IF3130
Sistem Paralel dan Terdistribusi
Shared Address Space Model – Vectorization

Achmad Imam Kistijantoro (imam@staff.stei.itb.ac.id)
Robithoh Annur (robithoh@stafff.stei.itb.ac.id)
Andreas Bara Timur (bara@staff.stei.itb.ac.id)

Februari 2023

Vectorization

- Prosesor memiliki unit vector khusus yang dapat melakukan load data dan beroperasi pada lebih dari 1 elemen data pada saat bersamaan
  - MMX instructions, AVX, AVX2, AVX512

- Bermanfaat pada aplikasi yang computation bound (performance dibatasi oleh kemampuan prosesor mengeksekusi operasi)
- Kebanyakan aplikasi bersifat memory bound (memory bandwidth menjadi batasan kecepatan)
- Namun vectorization dapat memberikan kinerja tambahan dengan little effort

Vectorization: Single Instruction Multiple Data

- Sebuah vector add instruction dapat menggantikan 8 operasi add instruction
- Jumlah elemen data yang dapat diproses dalam 1 instruksi disebut *vector length*.
- Ukuran dari vector unit (dinyatakan dalam bit) disebut sebagai *vector width*. Contoh: instruksi AVX512 memiliki width 512 bit (8 elemen data double precision).

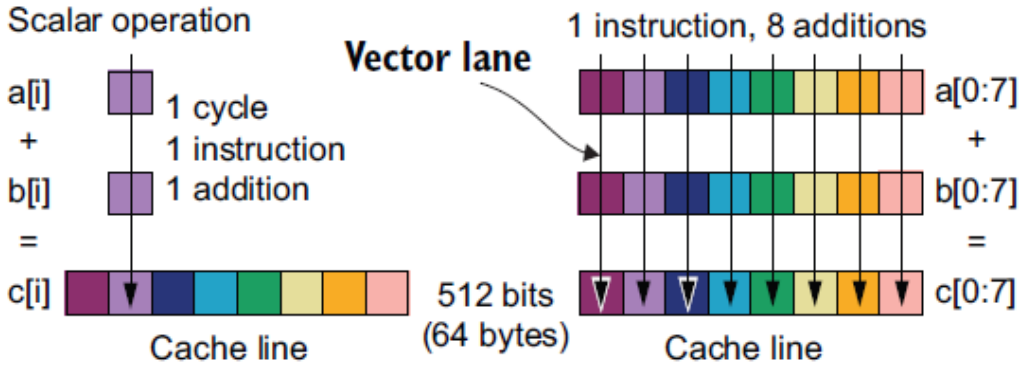


Figure 6.1 A scalar operation does a single double-precision addition in one cycle. It takes eight cycles to process a 64-byte cache line. In comparison, a vector operation on a 512-bit vector unit can process all eight double-precision values in one cycle.

vectorization

- Vectorization menggunakan kombinasi hardware dan software component
  - Generate vector instructions: oleh compiler atau manual dinyatakan via intrinsics/assembler code
  - Match instructions dengan hardware/vector unit pada processor. Instruksi yang tidak di-support hardware akan gagal dieksekusi (e.g. AVX512 tidak dapat berjalan pada prosesor yang lama)

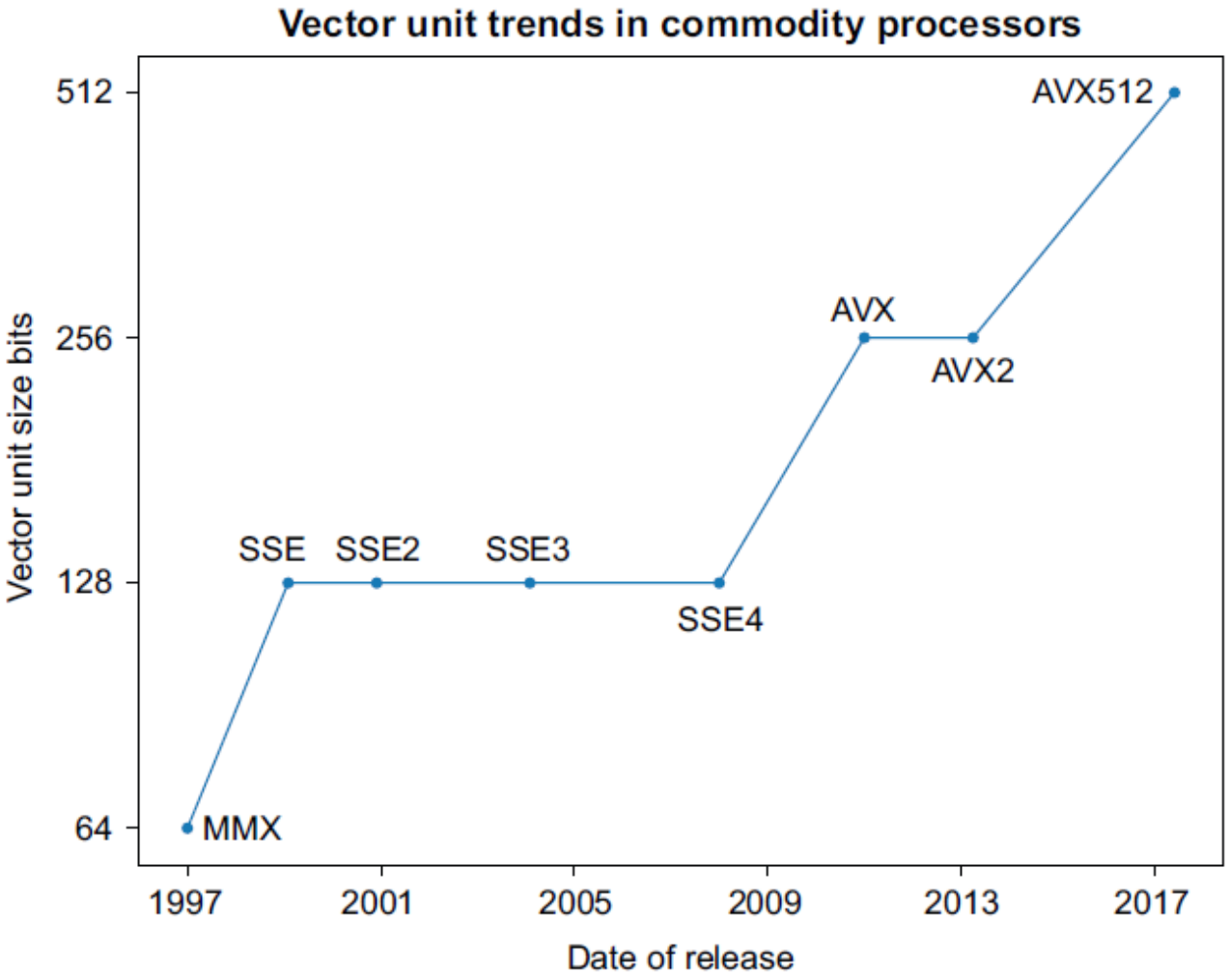
- *Compiler harus sesuai/menggunakan versi yang terbaru yang mensupport hardware terbaru*

Dukungan hardware

Release	Functionality
MMX (trademark with no official meaning)	Targeted towards the graphics market, but GPUs soon took over this function. Vector units shifted their focus to computation rather than graphics. AMD released its version under the name 3DNow! with single-precision support.
SSE (Streaming SIMD Extensions)	First Intel vector unit to offer floating-point operations with single-precision support
SSE2	Double-precision support added
AVX (Advanced Vector Extensions)	Twice the vector length. AMD added a fused multiply-add FMA vector instruction in its competing hardware, effectively doubling the performance for some loops.
AVX2	Intel added a fused multiply-add (FMA) to its vector processor.
AVX512	First offered on the Knights Landing processor; it came to the main-line multi-core processor hardware lineup in 2017. From the years 2018 and on, Intel and AMD (Advanced Micro Devices, Inc.) have created multiple variants of AVX512 as incremental improvements to vector hardware architectures.

Dukungan Hardware

- 8 register mm (64 bit) -- MMX
- 16 register xmm (128 bit) – SSE-SSE4
- 16 register ymm (256 bit) – AVX, AVX2
- 32 register zmm (512 bit) – AVX512


Vector unit trends in commodity processors
512
256
128
64
Vector unit size bits
AVX512
AVX
AVX2
SSE
SSE2
SSE3
SSE4
MMX
1997
2001
2005
2009
2013
2017
Date of release

Teknik Vectorization

- Menggunakan Optimized libraries
- Fitur Auto-vectorization dari compiler
- Menambahkan hints to the compiler
- Menggunakan Vector intrinsics
- Menggunakan Assembler instructions

Optimized Libraries

- Banyak library yang sudah disediakan dengan memanfaatkan vector instruction dari processor yang digunakan:

- **BLAS** (Basic Linear Algebra System)—A base component of high-performance
- linear algebra software
- **LAPACK**—A linear algebra package
- **SCALAPACK**—A scalable linear algebra package
- **FFT** (Fast Fourier transform)—Various implementation packages available
- Sparse Solvers—Various implementations of sparse solvers available

Auto vectorization

- Paling umum digunakan, karena memerlukan usaha paling kecil
- Goal: membuat kode yang memudahkan compiler mendeteksi bagian yang dapat di-vektorisasi
- Termasuk dengan menambahkan `restrict`, untuk C dan `__restrict`, `__restrict__` pada C++
  - Untuk menyatakan objek yg ditunjuk pointer dengan qualifier restrict tidak akan diacu dengan menggunakan pointer lain
- Flag compiler pada GCC:
  - -ftree-vectorize
  - -fstrict-aliasing
  - -fopt-info-vec-optimized

https://gcc.gnu.org/projects/tree-ssa/vectorization.html

Compiler	Strict aliasing	Vectorization	Floating-point flags

GCC, G++, GFortran v9	`-fstrict-aliasing`	`-ftree-vectorize`
		`-march=native`
		`-mtune=native`
		ver 8.0+:
		`-mprefer-vector`
		`-width=512`
			`-fno-trapping-math`
			`-fno-math-errno`

Clang v9	`-fstrict-aliasing`	`-fvectorize`
		`-march=native`
		`-mtune=native`
			`-fno-math-errno`

Intel icc v19	`-ansi-alias`	`-restrict`
		`-xHost`
		`-vecabi=cmdtarget`
		ver 18.0+:
		`-qopt-zmm-usage=high`	

MSVC	Not implemented	On by default	

IBM XLC v16	`-qalias=ansi`
	`-qalias=restrict`
		`-qsimd=auto`
		`-qhot`
		`-qarch=pwr9`
		`-qtune=pwr9`	

Kenapa autovectorization gagal?

- Data dependencies
- Penyebab lain
  - Alignment
  - Function call di dalam blok loop
  - Complex control flow/conditional branches
  - Loop not countable
  - Mixed data types
  - Non unit stride between elements
  - Loop body too complex
  - Vectorization seems inefficient

Data dependency

- Flow dependency: A variable within the loop is read after being written, known as a read-after-write (RAW)
- Anti-flow dependency: A variable within the loop is written after being read, known as a write-after-read (WAR)
- Output dependency: A variable is written to more than once in the loop

RAW

```
a = 40
b = 21
c = a+2
```

```
b = 40
a = b + 1
b = 21
```

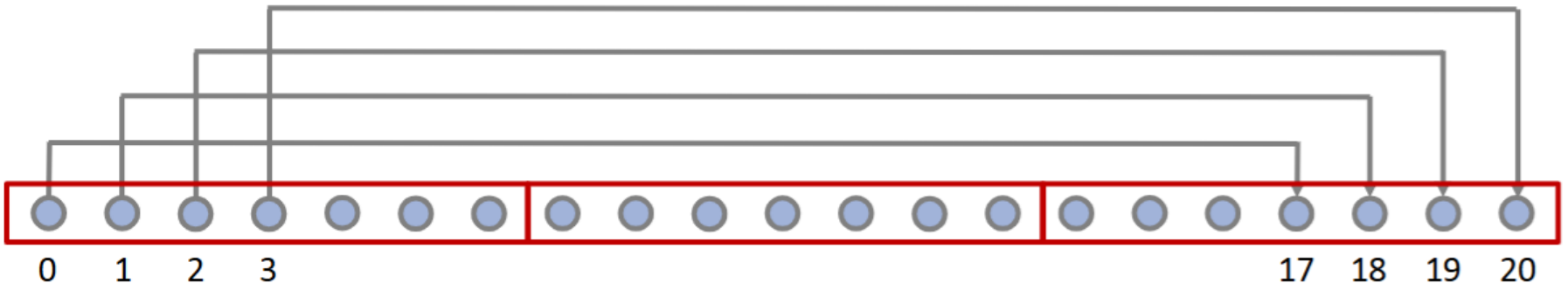
WAR

Loop-carried dependencies

- Dependencies yang terjadi antar loop

```
void lcd_ex(float* a, float* b, size_t n, float c1, float c2) {
  size_t i;
  for (i = 0; i < n; i++) {
    a[i] = c1 * a[i + 17] + c2 * b[i];
  }
}
```

- Kode ini tidak bisa diparalelisasi, namun bisa di-vectorisasi, jika panjang vector lebih kecil dari dependencies



Loop not countable

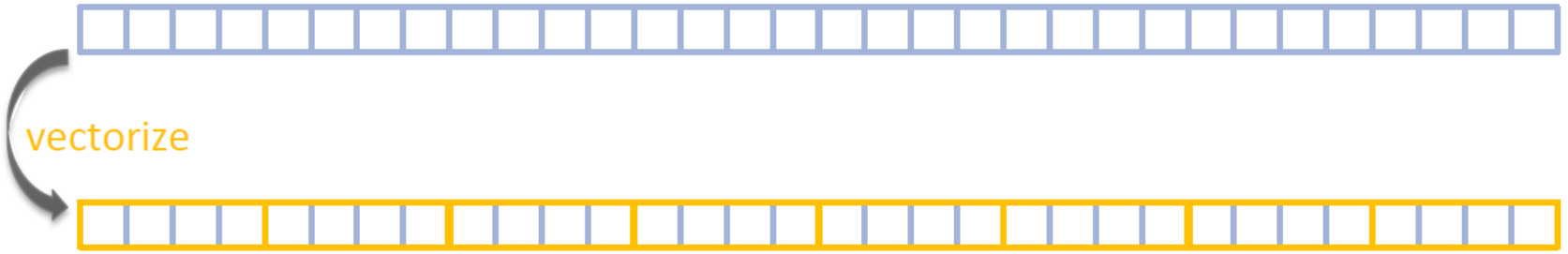
```
typedef struct {
  float* data;
  size_t size;
} vec_t;

void vec_eltwise_product(vec_t* a, vec_t* b, vec_t* c) {
  size_t i;
  for (i = 0; i < a->size; i++) {
    c->data[i] = a->data[i] * b->data[i];
  }
}
```

Hints compiler – openmp simd

- Memecah loop menjadi blok yg muat dalam simd vector register
- Tidak memparalelkan loop

- #pragma omp simd reduction(min:nama_var)
- #pragma omp simd private(myvar)

```
void sprod(float *a, float *b, int n) {
  float sum = 0.0f;
#pragma omp simd reduction(+:sum)
  for (int k=0;k<n;k++)
    sum += a[k](b[k]
  return sum
```



Simd loop clause

- safelen(length): max number of iterations that can run concurrently without breaking dependencies
- Simdlen(length): preferred length of SIMD registers

Misaligned loop

- Akses data pada iterasi mungkin tidak align
- Operasi vector memerlukan data yang align dengan ukuran vector.
  - AVX512 menggunakan alignment 64 byte

Misaligned loop

- Contoh:

```
float a[N];
for(i=0;i<N;i++)
  a[i]=1.;
```

Alignment variable a adalah 4, bukan 64, sehingga pada avx512 compiler akan mensplit loop menjadi

```
for(i=0,i_al=0; (&a[i]%64!=0);i++) {
  a[i]=1.;
  i_al=i+1;
}
for(i=i_al;i<N;i++)
  a[i]=1.;
```

Misaligned loop

- Peel loop: a loop to execute for misaligned data so that the main loop would then have aligned data
- Remainder loop: a loop that executes after the main loop to handle a partial set of data that is too small for a full vector length
- Pendefinisian variable agar aligned dengan ukuran vector
  - `double a[N] __((aligned(64)))`

Vector intrinsics

- C menyediakan low level vector intrinsics untuk operasi vector
  - Deklarasi vector variable: m128, m128i, m128h
  - Operasi load, store
  - Operasi aritmetika

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html