How to Write Fast Numerical Code

Spring 2011

Lecture 19

Instructor: Markus Püschel

TA: Georg Ofenbeck

ETH

Eidgenössische Technische Hochschule Zürich

Swiss Federal Institute of Technology Zurich

Schedule

May 2011

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Today

Lecture

Project meetings

Project presentations

- 10 minutes each
- random order
- random speaker

Final project paper and code due:
A week or so (exact date still TBD) after semester end

SIMD Extensions and SSE

- **Overview**
- **SSE family, floating point, and x87**
- **SSE intrinsics**
- ***Compiler vectorization***

References:

Intel icc manual (currently 12.0) → Creating parallel applications → Automatic vectorization

http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/cpp/lin/index.htm

Compiler Vectorization

- Compiler flags
- Aliasing
- Proper code style
- Alignment

Compiler Flags (icc 12.0)

Linux* OS and Mac OS* X	Windows* OS	Description
-vec -no-vec	/Qvec /Qvec-	Enables or disables vectorization and transformations enabled for vectorization. Vectorization is enabled by default. To disable, use -no-vec (Linux* and MacOS* X) or /Qvec- (Windows*) option. Supported on IA-32 and Intel® 64 architectures only.
-vec-report	/Qvec-report	Controls the diagnostic messages from the vectorizer. See Vectorization Report.
-simd -no-simd	/Qsimd /Qsimd-	Controls user-mandated (SIMD) vectorization. User-mandated (SIMD) vectorization is enabled by default. Use the -no-simd (Linux* or MacOS* X) or /Qsimd- (Windows*) option to disable SIMD transformations for vectorization.

Architecture flags:

Linux: `-x`*`Host`* ⊃ `-m`*`Host`*

Windows: `/Qx`*`Host`* ⊃ `/Qarch:`*`Host`*

`Host` **in {SSE2, SSE3, SSSE3, SSE4.1, SSE4.2}**

Default: `-mSSE2, /Qarch:SSE2`

How Do I Know the Compiler Vectorized?

- `vec-report` **(previous slide)**
- **Look at assembly:** `mulps, addps, xxxps`
- **Generate assembly with source code annotation:**
  - Visual Studio + icc: /Fas
  - icc on Linux/Mac: -S

Example

```
void myadd(float *a, float *b, const int n) {
  for (int i = 0; i< n; i++)
    a[i] = a[i] + b[i];
}
```

unvectorized: /Qvec-

```
<more>
;;;    a[i] = a[i] + b[i];
movss      xmm0, DWORD PTR [rcx+rax*4]
addss      xmm0, DWORD PTR [rdx+rax*4]
movss      DWORD PTR [rcx+rax*4], xmm0
<more>
```

vectorized:

```
<more>
;;;    a[i] = a[i] + b[i];
movss      xmm0, DWORD PTR [rcx+r11*4]
addss      xmm0, DWORD PTR [rdx+r11*4]
movss      DWORD PTR [rcx+r11*4], xmm0
…
movups     xmm0, XMMWORD PTR [rdx+r10*4]
movups     xmm1, XMMWORD PTR [16+rdx+r10*4]
addps      xmm0, XMMWORD PTR [rcx+r10*4]
addps      xmm1, XMMWORD PTR [16+rcx+r10*4]
movaps     XMMWORD PTR [rcx+r10*4], xmm0
movaps     XMMWORD PTR [16+rcx+r10*4], xmm1
<more>
```

why this? (movss / addss / movss)

why everything twice?
why movups and movaps?

unaligned (movups) — aligned (movaps)

Aliasing

```
for (i = 0; i < n; i++)
  a[i] = a[i] + b[i];
```

Cannot be vectorized in a straightforward way due to potential aliasing.

However, in this case compiler can insert runtime check:

```
if (a + n < b || b + n < a)
  /* vectorized loop */
  ...
else
  /* serial loop */
  ...
```

Removing Aliasing

- **Globally with compiler flag:**
  - `-fno-alias, /Oa`
  - `-fargument-noalias, /Qalias-args-` (function arguments only)
- **For one loop: pragma**

```
void add(float *a, float *b, int n) {
  #pragma ivdep
  for (i = 0; i < n; i++)
    a[i] = a[i] + b[i];
}
```

- **For specific arrays: restrict (needs compiler flag `–restrict, /Qrestrict`)**

```
void add(float *restrict a, float *restrict b, int n) {
for (i = 0; i < n; i++)
    a[i] = a[i] + b[i];
}
```

Proper Code Style

- **Use countable loops = number of iterations known at runtime**
  - *Number of iterations is a:*
    constant
    loop invariant term
    linear function of outermost loop indices

- **Countable or not?**

```
for (i = 0; i < n; i++)
  a[i] = a[i] + b[i];
```

```
void vsum(float *a, float *b, float *c) {
  int i = 0;

  while (a[i] > 0.0) {
    a[i] = b[i] * c[i];
    i++;
  }
}
```

Proper Code Style

- **Use arrays, structs of arrays, not arrays of structs**
- **Ideally: unit stride access in innermost loop**

```
void mmm1(float *a, float *b, float *c) {
  int N = 100;
  int i, j, k;

  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < N; k++)
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

```
void mmm2(float *a, float *b, float *c) {
  int N = 100;
  int i, j, k;

  for (i = 0; i < N; i++)
    for (k = 0; k < N; k++)
      for (j = 0; j < N; j++)
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Alignment

```
float x[1024];
int i;

for (i = 0; i < 1024; i++)
   x[i] = 1;
```

Cannot be vectorized in a straightforward way since x may not be aligned

However, the compiler can peel the loop to extract aligned part:

```
float x[1024];
int i;

peel = x & 0x0f; /* x mod 16 */
if (peel != 0) {
  peel = 16 - peel;
  /* initial segment */
  for (i = 0; i < peel; i++)
    x[i] = 1;
}
/* 16-byte aligned access */
for (i = peel; i < 1024; i++)
  x[i] = 1;
```

Ensuring Alignment

- **Align arrays to 16-byte boundaries (see earlier discussion)**
- **If compiler cannot analyze:**
  - Use pragma for loops

```
float x[1024];
int i;

#pragma vector aligned
for (i = 0; i < 1024; i++)
   x[i] = 1;
```

  - For specific arrays:
    `__assume_aligned(a, 16);`

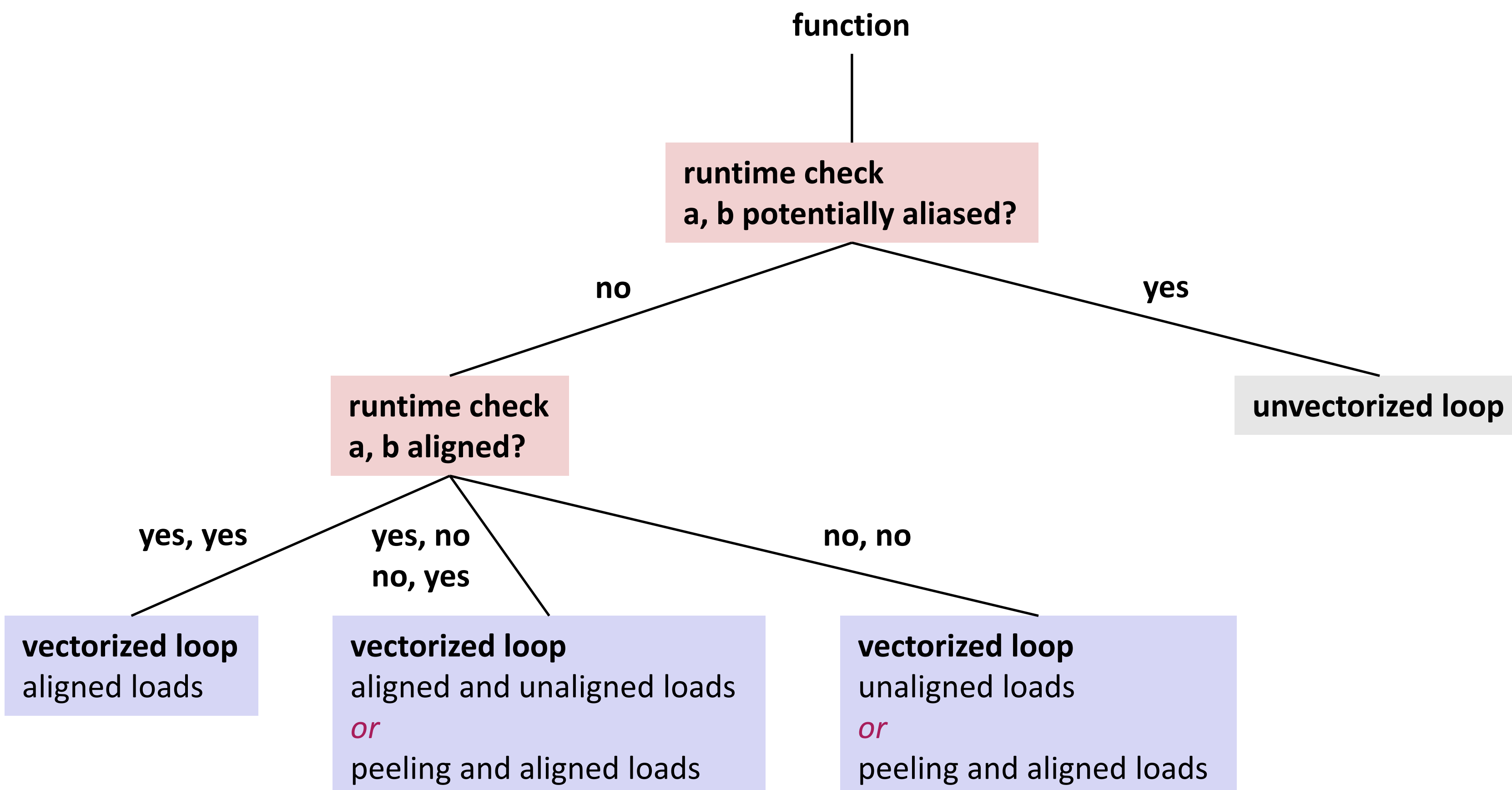
```
void myadd(float *a, float *b, const int n) {
  for (int i = 0; i< n; i++)
    a[i] = a[i] + b[i];
}
```

Assume:

- **No aliasing information**
- **No alignment information**

Can compiler vectorize?

Yes: Through versioning



Compiler Vectorization

- **Read manual**

Linear Transforms

- Overview
- Discrete Fourier transform
- Fast Fourier transforms
- Optimized implementation and autotuning (FFTW)
- Automatic implementation and optimization (Spiral)

Linear Transforms

- Very important class of functions: signal processing, scientific computing, …
- *Mathematically:* Change of basis = Multiplication by a fixed matrix T

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = y = Tx$$

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

$$T = [t_{k,\ell}]_{0 \le k,\ell < n}$$

Output *Input*

- Equivalent definition: Summation form

$$y_k = \sum_{\ell=0}^{n-1} t_{k,\ell} x_\ell, \quad 0 \le k < n$$

Smallest Relevant Example

Transform (matrix):

$$T = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

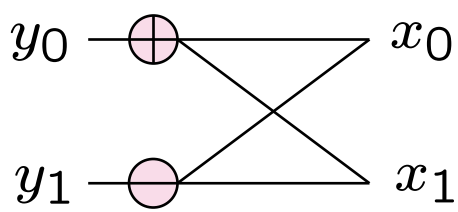
Computation:

$$y = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} x$$

or

$$\begin{aligned} y_0 &= x_0 + x_1 \\ y_1 &= x_0 - x_1 \end{aligned}$$

As graph (direct acyclic graph or DAG):



called a butterfly

http://charlottesmartypants.blogspot.com/ 2011_02_01_archive.html

Transforms: Examples

- **A few dozen transforms are relevant**
- **Some examples**

$$\mathbf{DFT}_n = [e^{-2k\ell\pi i/n}]_{0\le k,\ell<n}$$

$$\mathbf{RDFT}_n = [r_{k\ell}]_{0\le k,\ell<n}, \quad r_{k\ell} = \begin{cases} \cos\frac{2\pi k\ell}{n}, & k \le \lfloor\frac{n}{2}\rfloor \\ -\sin\frac{2\pi k\ell}{n}, & k > \lfloor\frac{n}{2}\rfloor \end{cases}$$

universal tool

$$\mathbf{DHT} = \left[\cos(2k\ell\pi/n) + \sin(2k\ell\pi/n)\right]_{0\le k,\ell<n}$$

$$\mathbf{WHT}_n = \begin{bmatrix} \mathbf{WHT}_{n/2} & \mathbf{WHT}_{n/2} \\ \mathbf{WHT}_{n/2} & -\mathbf{WHT}_{n/2} \end{bmatrix}, \quad \mathbf{WHT}_2 = \mathbf{DFT}_2$$

$$\mathbf{IMDCT}_n = \left[\cos((2k+1)(2\ell+1+n)\pi/4n)\right]_{0\le k<2n, 0\le\ell<n}$$

MPEG

$$\mathbf{DCT\text{-}2}_n = \left[\cos(k(2\ell+1)\pi/2n)\right]_{0\le k,\ell<n}$$

JPEG

$$\mathbf{DCT\text{-}3}_n = \mathbf{DCT\text{-}2}_n^T \quad \text{(transpose)}$$

$$\mathbf{DCT\text{-}4}_n = \left[\cos((2k+1)(2\ell+1)\pi/4n)\right]_{0\le k,\ell<n}$$

Blackboard

- **Discrete Fourier transform (DFT)**
- **Transform algorithms**
- **Fast Fourier transform, size 4**