

# Auto-vectorization

Adam Kosiorek

May 16, 2015

## Vectorization

What can be vectorized?



# What is vectorization?

*Vectorization = Loop unrolling + packed SIMD instructions (1)*

Loop unrolling: easy

SIMD: SSE, AVX, Nano etc.

How to get them:

- ▶ write assembly
- ▶ compiler intrinsics
- ▶ special purpose language extensions eg. OpenCL, CUDA
- ▶ vectorizing compiler + guidelines

# The simplest case

```
int A[1024], B[1024], C[1024];  
// initialize A and B  
for(int i = 0; i < 1024; ++i)  
    C[i] = A[i] * B[i];
```

## vectorized:

- ▶ -O2 and beyond
- ▶ speedup  $\in [2, 8]$

## not vectorized:

- ▶ -O0, -O1, -Og, -g, -no-vec
- ▶ operates on single entry at a time
- ▶ slow

# Why was it vectorized?

Properties needed for vectorization:

- ▶ countable
- ▶ single entry, single exit
- ▶ no branching
- ▶ the innermost loop
- ▶ no function calls
- ▶ no data dependencies

# Countable

**(a)** countable:

```
for (i = 0; i < MAX; ++i)
    C[i] = A[i] * B[i];
```

**(b)** uncountable:

```
for (i = 0; i < MAX; ++i)
{
    C[i] = A[i] * B[i];
    if (A[i] < B[i])
        MAX--;
}
```

Index cannot depend on the loop execution!

## Single entry, single exit

**(a)** single exit:

```
for (i = 0; i < MAX; ++i)
    C[i] = A[i] * B[i];
```

**(b)** multiple exits:

```
for (i = 0; i < MAX; ++i)
{
    C[i] = A[i] * B[i];
    if (A[i] < B[i])
        break;
}
```

Vectorization could “skip” the termination condition!



# No branching

**(a)** no branching?:

```
for(i = 0; i < MAX; ++i)
    if(A[i] != 0)
        C[i] = A[i];
    else
        C[i] = B[i];
```

**(b)** branching:

```
for(i = 0; i < MAX; ++i)
    switch(i % 3) {
        case 0: C[i] = A[i];
                break;
        case 1: C[i] = B[i];
                break;
        default: C[i] = 0;
    }
```

If statements implemented by “masking assignment” are ok.

# Innermost loops

```
for(int i = 0; i < 10; ++i)
  for(j = 0; j < MAX; ++j)
    C[i][j] = A[i][j] * B[i][j];
```

- ▶ Only j-loop vectorized
- ▶ Outer-loop vectorization inefficient
- ▶ Can be enforced with `#pragma SIMD`
- ▶ Outer-loop vectorization possible after loop interchange

# Function Calls

```
int compute(int a, int b);  
...  
for(i = 0; i < MAX; ++i)  
    C[i] = compute(A[i], B[i]);
```

works if the function:

- ▶ can be inlined
- ▶ is declared as a vector function:

```
__attribute__((vector))  
int compute(int a, int b);
```

- ▶ is compiler intrinsic function
- ▶ is one of the math functions: sin, cos, exp, pow, etc.

a

a