

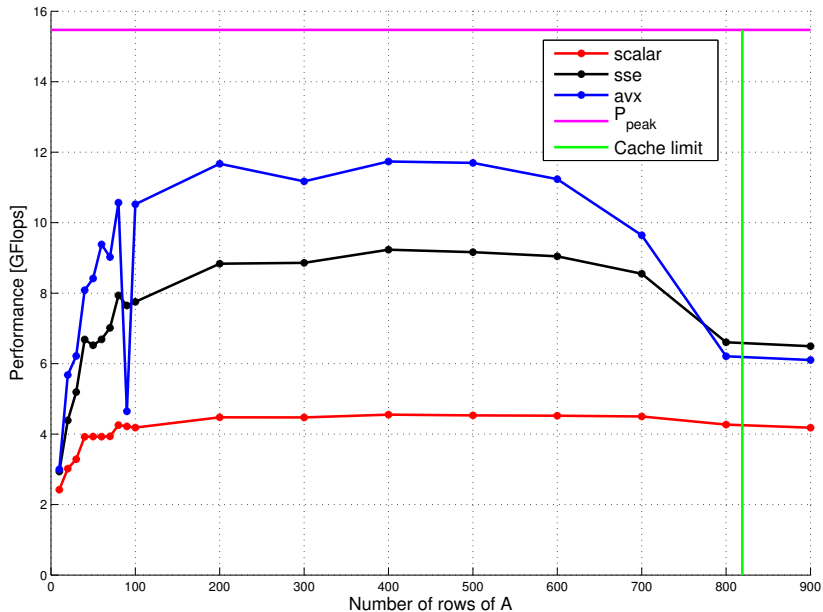
Auto-vectorization

Adam Kosiorek

Supervised by: Prof. M. Gerndt

June 3, 2015

Why do we need vectorization?



What is vectorization?

Vectorization = Loop unrolling + packed SIMD instructions (1)

Loop unrolling: manually or by compiler

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

```
double 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

The simplest case - vectorization

default:

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

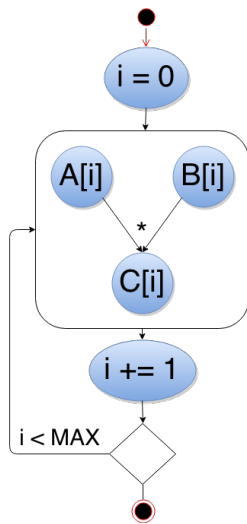
unrolled:

```
for (i = 0; i < MAX; i+=4) {
    C[i] = A[i] * B[i];
    C[i+1] = A[i+1] * B[i+2];
    C[i+2] = A[i+1] * B[i+2];
    C[i+3] = A[i+3] * B[i+3];
}
```

vectorized: (with AVX intrinsics)

```
_mm256* mA = (_mm256*)A;
_mm256* mB = (_mm256*)B;
_mm256* mC = (_mm256*)C;
for (i = 0; i < MAX/4; ++i)
    mC[i] = _mm256_add_pd(mA[i], mB[i]);
```

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]);
```

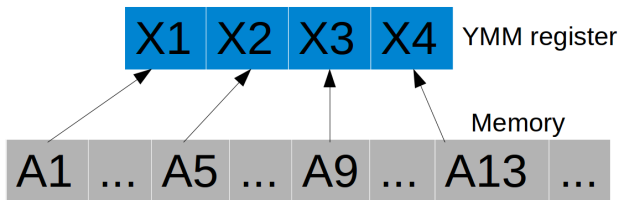
works if the function:

- ▶ 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.

Non-contiguous Memory Access



- ▶ each entry loaded separately
- ▶ happens with non-unit stride and indirect addressing e.g.

$$C[i] = C[i] * A[i * 2];$$
$$C[i] = C[i] * A[B[i]];$$

Data Dependencies

Read-after-write (flow dependency): ✗

```
for (i=1; i<MAX; ++i)
    A[i] = A[i-1] + 1;
```

Write-after-read (anti-dependency): ✓

```
for (i=1; i<MAX; ++i)
    A[i-1] = A[i] + 1;
```

Write-after-write (output dependency): ✗

```
for (i=0; i<MAX; ++i)
    A[i - i%2] = A[i] * B[i];
```

Reduction: ✓

```
double sum = 0;
for (i=0; i<MAX; ++i)
    sum = sum + A[i];
```

Assumed Data Dependencies - Pointer Aliasing

```
void compute(int* A, int* B, int* C, int N) {  
    for(int i = 1; i < N; ++i)  
        C[i] = A[i-1] + B[i];  
}
```

- C/C++ makes no assumptions about pointers:

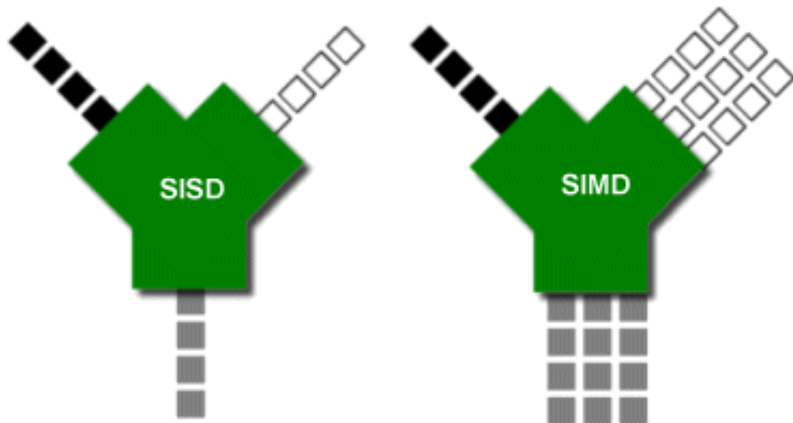
```
compute(a, b, a);
```

is legal!

References

1. A Guide to Vectorization with Intel C++ Compilers
<https://software.intel.com/en-us/articles/a-guide-to-auto-vectorization-with-intel-c-compilers>
2. Intrinsics Guide
<https://software.intel.com/sites/landingpage/IntrinsicsGuide/>

Questions?



- Instructions
- Data
- Results