Program Optimization – SIMD

simon.marechal@synacktiv.com

2023/2024

What is SIMD?

What is SIMD?

SIMD

`single instruction, multiple data`

On Intel CPUs

- MMX (1997) with 8, 64 bits, registers (shared with the FP unit)
- SSE (1999) with 8, 128 bits, registers
- SSE2 (2000), SSE3 (2004), SSE4 (2006), incremental updates to SSE
- AVX (2011), with 16, 256 bits, registers
- AVX-512 (2013), with 32, 512 bits, registers

Support

- processors with the *Core* commercial name support these ;
- *Pentium* / *Celeron* do not
- that's why you will need fallback algorithms for older / less capable processors :'(

Compiler support

- most major compiler support the instruction set
- must specify the target architecture though
- usually only works for very simple algorithms

What does it mean

Multiple data in a single register

In a 128 bits register, you can have:

- 2 x 64 bits values (`uint64_t` or `double`) ;
- 4 x 32 bits values (`uint32_t` or `float`) ;
- 8 x 16 bits values (`uint16_t`) ;
- 16 x 8 bits values (`uint8_t`).

And then perform the same operation on all of them at once.

Motivating example

Physics simulation, where, for every *tick*, we do need to update velocity and position:

$$V_t \leftarrow V_{t-1} + A_{t-1} \tag{1}$$
$$P_t \leftarrow P_{t-1} + V_{t-1} \tag{2}$$

In a 3D world, that means 6 additions for every simulation element.

In standard C

```
void add(struct P *l, const struct P *r)
{
  l->x += r->x;
  l->y += r->y;
  l->z += r->z;
  l->w += r->w;
}
...
for (int i = 0; i < OBJECTS; i++)
{
  add(&objects[i].position, &objects[i].velocity);
  add(&objects[i].velocity, &objects[i].acceleration);
}
```

Generated code, -O1, 9.2s

```
0000000000001155 <add>:
    movsd   (%rdi),%xmm0
    addsd   (%rsi),%xmm0
    movsd   %xmm0,(%rdi)
    movsd   0x8(%rdi),%xmm0
    addsd   0x8(%rsi),%xmm0
    movsd   %xmm0,0x8(%rdi)
    movsd   0x10(%rdi),%xmm0
    addsd   0x10(%rsi),%xmm0
    movsd   %xmm0,0x10(%rdi)
    movsd   0x18(%rdi),%xmm0
    addsd   0x18(%rsi),%xmm0
    movsd   %xmm0,0x18(%rdi)
```

Generated code, -O2, 8.2s

Inlined code, operation reordering:

```
movsd   0x20(%rax),%xmm3
movsd   (%rax),%xmm0
add     $0x60,%rax
movsd   -0x38(%rax),%xmm2
movsd   -0x30(%rax),%xmm1
addsd   %xmm3,%xmm0
movsd   -0x48(%rax),%xmm4
addsd   -0x20(%rax),%xmm3
movsd   %xmm0,-0x60(%rax)
movsd   -0x58(%rax),%xmm0
movsd   %xmm3,-0x40(%rax)
addsd   %xmm2,%xmm0
addsd   -0x18(%rax),%xmm2
```

```
movsd   %xmm0,-0x58(%rax)
movsd   -0x50(%rax),%xmm0
movsd   %xmm2,-0x38(%rax)
addsd   %xmm1,%xmm0
addsd   -0x10(%rax),%xmm1
movsd   %xmm0,-0x50(%rax)
movsd   -0x28(%rax),%xmm0
movsd   %xmm1,-0x30(%rax)
addsd   %xmm0,%xmm4
addsd   -0x8(%rax),%xmm0
movsd   %xmm4,-0x48(%rax)
movsd   %xmm0,-0x28(%rax)
...
```

Generated code, -O3, 3.9s

Actually use vector instructions:

```
movupd 0x30(%rdx),%xmm7
movupd 0x10(%rdx),%xmm0
add    $0x60,%rdx
movupd -0x10(%rdx),%xmm2
movupd -0x40(%rdx),%xmm6
movupd -0x60(%rdx),%xmm1
addpd  %xmm7,%xmm0
movupd -0x20(%rdx),%xmm3
addpd  %xmm7,%xmm2
movupd -0x40(%rdx),%xmm7
addpd  %xmm6,%xmm3
```

```
addpd  %xmm7,%xmm1
addpd  %xmm2,%xmm0
addpd  %xmm3,%xmm1
movups %xmm0,-0x50(%rdx)
movupd -0x10(%rdx),%xmm0
movups %xmm1,-0x60(%rdx)
movupd -0x20(%rdx),%xmm1
addpd  %xmm2,%xmm0
addpd  %xmm3,%xmm1
movups %xmm0,-0x30(%rdx)
movups %xmm1,-0x40(%rdx)
```

Generated code, `-O3 -march=native`, 3.8s

Wider registers:

```
vmovupd 0x20(%rdx),%ymm0
vmovupd 0x40(%rdx),%ymm1
add     $0x60,%rdx
vaddpd %ymm0,%ymm1,%ymm2
vaddpd -0x60(%rdx),%ymm0,%ymm0
vaddpd %ymm2,%ymm1,%ymm1
vaddpd %ymm2,%ymm0,%ymm0
vmovupd %ymm1,-0x40(%rdx)
vmovupd %ymm0,-0x60(%rdx)
```

Caveat

- Optimizations are VERY fragile!
- most of the time, do not even fire

Task: password cracker

Idea

Adapt your program so that it tries at least 4 plain texts in parallel.
From:

```
00  40 40 40 40  40 40 40 80  00 00 00 00  00 00 00 00
...
```

To:

```
00  40 40 40 40  40 40 40 40  40 40 40 40  40 40 40 40
00  40 40 40 80  40 40 41 80  40 40 42 80  40 40 43 80
...
```

QUESTIONS?

Thank you for your attention

SYNACKTIV DIGITAL SECURITY