Future of Vectorization

Georg Zitzlsberger

georg.zitzlsberger@vsb.cz

IT4Innovations
national01$#&0
supercomputing
center@#01%101

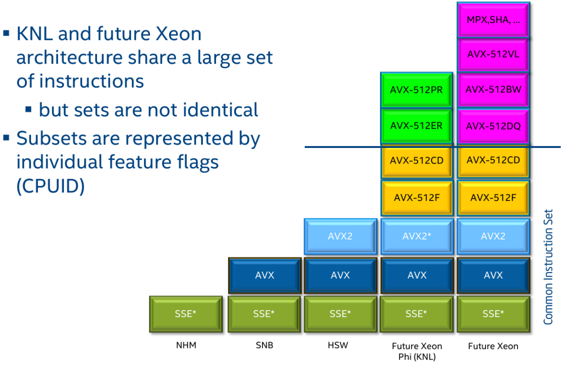
5th of July 2017

Agenda

- Future of Vectorization
- Why Intel Software Development Emulator
- Getting Started
- Histogram Tool
- SIMD Mask Profiling
- Calculating FLOP
- Memory Footprint
- Which SDE Data is Important?

Future of Vectorization

- Remember AVX-512?



- KNL and future Xeon architecture share a large set of instructions
  - but sets are not identical
- Subsets are represented by individual feature flags (CPUID)

(Image: Intel)

- Larger vectors and more features (e.g. masking)
- How can we validate their efficiency now?
- Do I really need a physical system?

⇒ Intel Software Development Emulator

Why Intel Software Development Emulator

- Emulate a wide range of Intel microarchitectures - present and future
- Alternative to measure FLOPs and understand effects of SIMD instructions like:
  - Are the latest SIMD instructions used?
  - How many elements of vectors are used (masking)?
  - What are estimated speedups even there's no access to HW?
- Memory footprint of application

Getting Started

- Latest version of Intel Software Development Emulator is 8.4.
- Easy to use:

```
$ sde [options] -- application [app_options]
```

  ⇒ No recompilation and no debug information required!
- Some interesting microarchitectures[1] (used as `[options]`):
  - `-hsw`: Haswell (all SIMD up to AVX2)
  - `-skx`: Skylake Server (CORE-AVX512)
  - `-knl`: Knights Landing (MIC-AVX512)
- Only instructions will be emulated missing on the underlying architecture
- Numerical FP results are identical to native execution
- Help via `sde -help` or `sde -long-help`

---

[1] KNC is not supported

Histogram Tool

- Which instructions types are executed by an application (esp. which SIMD extension)?
- The histogram tool counts the instructions by groups (like `*avx256` or `*avx512`)
- Enabled by option `-mix`:

  ```
  $ sde -mix -- application [app_options]
  ```

  Output can be found in file `sde-mix-out.txt`[2]
- Statistics are provided by:
  - Thread
  - Function per thread
  - Summary of entire run
- Use `-mix_filter_rtn` to filter a single routine

---

[2] Can be changed by option `-omix <mix.out>`

Histogram Tool - Example

Compile and run SDE:

```
> g++ vec.cpp -O2 -mavx2 -ftree-vectorize -o vec

> c++filt _Z3vecPdS_S_
vec(double*, double*, double*)

> sde64 -hsw -mix -mix_filter_rtn "_Z3vecPdS_S_"  -- ./vec
```

File `vec.cpp`:

```
...
__attribute__ ((noinline))
void vec(double *a, double *b, double *c)
{
    int i;
    for (i = 0; i < 100000; ++i) {
        c[i] = a[i] * b[i];
    }
}
...
```

File `sde-mix-out.txt` (simplified):

```
...
FN: vec(double*, double*, double*) IMG: vec
 AVX vmovupd xmm0, {mem}
 AVX vmovupd xmm1, {mem}
 AVX vinsertf128 ymm0, ymm0, {mem}
 AVX vinsertf128 ymm1, ymm1, {mem}, 0x1
 AVX vmulpd ymm0, ymm0, ymm1
 AVX vmovups xmmword ptr {mem}, xmm0
 AVX vextractf128 {mem}, ymm0, 0x1
BASE add rax, 0x20    % =4
BASE cmp rax, 0xc3500 % =100000
BASE jnz 0x558ee45c9780
...
*avx128 75001
*avx256 100000
...
```

Information on instruction groups like `*avx128` or `*avx256` is here

SIMD Mask Profiling

- For AVX512 (KNL and SKX) the instruction set allows direct masking of SIMD elements
- Masking reduces the advantage of SIMD but sometimes is necessary
- SDE can profile the amount of masked elements to help understand the impact
- Option `-dyn_mask_profile` creates file `sde-dyn-mask-profile.txt`
- Summary shows masked computations and data transfers (`dataxfer`, `scatter` and `gather`)

SIMD Mask Profiling - Example

File `mask.cpp`:

```
...
void mask(double * restrict r, double *a,
          double *b, double *c, bool *msk)
{
    int i ;
    for (i = 0; i < N; ++i) {
        if (msk[i])
            // FMA
            r[i] = a[i] * b[i] + c[i];
        else
            // FMA
            r[i] = c[i] * b[i] + a[i];
    }
}
...
```

Compile and run SDE:

```
> icpc mask.cpp -xmic-avx512 -o mask

> sde64 -knl -dyn_mask_profile -- ./mask
```

File `sde-dyn-mask-profile.txt` (simplified):

```
...
<summarytable>
 mask     cat    vec-length #elements element_s element_t|icount comp_count %max-comp
  masked  dataxfer 512b     8elem     64b       fp       |1250   6666       66.660
  masked  mask     512b     8elem     64b       fp       |1250   6666       66.660
 unmasked dataxfer 64b      1elem     64b       fp       |1      1          100.000
 unmasked dataxfer 512b     8elem     64b       fp       |1250   10000      100.000
 unmasked dataxfer 512b     16elem    32b       fp       |3750   60000      100.000
 unmasked mask     64b      1elem     64b       fp       |1      1          100.000
 unmasked mask     512b     8elem     64b       fp       |1250   10000      100.000
</summarytable>
...
```

SIMD Mask Profiling - Example cont'd

. . . and detailed histogram (`popcount*`) of elements computed:

```
...
<instruction-details>
 <IP> 0x400bf7 </IP>
 <disassembly> vfmadd213pd zmm5{k1}, zmm4, zmmword ptr [rcx+rax*8] </disassembly>
 <source-location>
    <img> /home/zit0029/lab/SDE/mask </img>
    <routine> _Z4maskPdS_S_S_Pb </routine>
 </source-location>
 <dynamic-stats>
    <execution-counts> 1250 </execution-counts>
    <computation-count> 6666 </computation-count>
    <percent-of-max-computation>  66.660 </percent-of-max-computation>
    <scalarish> 0 </scalarish>
    <popcount>
       <popcount5> 834 </popcount5>
       <popcount6> 416 </popcount6>
    </popcount>
 </dynamic-stats>
</instruction-details>
...
```

Calculating FLOP

- SDE can help to calculate the effective FLOPs of an application even if simulated
- Follow these steps:
  1. Use histogram tool (instruction mix) to get sum of all `*elements_fp_`[FP$_{type}$]`_`[$\#_{elements}$] where FP$_{type}$ = `[float|double]` and $\#_{elements}$ are SIMD elements processed.
  2. Add the FMA operations (one operation of the FMA was already considered above):
     Search for `VFMADD...` and `VFMSUB...`, like `VFMADD213PD_YMMqq_YMMqq_YMMqq`
  3. Consider masking (for AVX512):
     - 3.1 Use mask profiling from SDE and add the `comp_count` counters with category `mask` and `element_t` being `fp`
     - 3.2 Same as for unmasked FMA, add 2nd FLOP of the masked variants (`VFMADD...MASK...`)

The full guide can be found ▸ here

Memory Footprint

- SDE can record how often a cache-line (64 byte) was referenced (load or store).
- Its simple but can be helpful to see changes of number of cache lines being used.
- Option `-footprint` creates `sde-footprint.txt`:
  Lists:
  1. # of cache line loads and stores
  2. # of pages touched by loads and stores
     But requires `-footprint_page_size` option for page size.

Which SDE Data is Important?

The following information from SDE could be important for you:

- Ratio of SIMD instructions to non-SIMD:

$$\mathrm{R}_{SIMD} = \frac{\texttt{*sse-packed} + \texttt{*avx128} + \texttt{*avx256} + \texttt{*avx512}}{\texttt{*total}}$$

- The ratio further decreases with more masked SIMD operations:

$$Comp_{count} \leq Inst_{count} * VL$$

- Calculating the FLOP of a routine can help estimating the impact of SIMD:
  Comparing FLOPs[3] by normalizing with scalar vs. vectorized overall instruction count (FLOP per instruction)

---

[3] Compilers could change FLOPs slightly for different compilations

Lab Time!