AVX-512 in FFmpeg

Kieran Kunhya <kieran@kunhya.com>
What is AVX-512?

• New SIMD (Single Instruction Multiple Data) Instruction Set for Intel since 2017, and more recently AMD CPUs
• 512-bit register sizes (zmm)
• Many new instructions
• Opmasks
• Comparison instructions
• Many things not so interesting in multimedia (e.g. cryptography, neural networks)

• Lots of fancy words, but high schoolers have written assembly in FFmpeg!
Why is this relevant now?

• AVX-512 has been around since 2017. Why is it relevant now?

• Old Skylake (server) systems had large performance throttling when using larger registers. AVX-512 remained unused in multimedia

• Could still use new instructions with smaller registers
  • Can be beneficial in some cases (see later)

• Ice Lake (10th and 11th Gen Intel) first to have no throttling
How to get started?

- Intel have removed AVX-512 support in consumer processors (from 12\textsuperscript{th} Gen) 😞
- Available in AMD Zen 4
- Still exists in Intel Server CPUs (Xeon). Available from cloud providers

- Easiest way is to buy an Intel NUC (11\textsuperscript{th} Gen)
Existing work in multimedia using AVX-512

- dav1d project added AVX-512 support
- AV1 decoding, particularly beneficial owing to large block sizes
  - 10-20% faster *overall* decoding
- Classic FFmpeg/x264 approach to assembly
- No intrinsics, nor inline assembly
- Detect CPU capabilities and set function pointers to appropriate functions
- Messy Venn Diagram of capabilities, but two in practice we care about
CPU_FLAGS in FFmpeg

int av_get_cpu_flags(void);

#define AV_CPU_FLAG_AVX512  0x100000 ///< AVX-512 functions: requires OS support even if YMM/ZMM registers aren't used

#define AV_CPU_FLAG_AVX512_ICL  0x200000 ///< F/CD/BW/DQ/VL/VNNI/IFMA/VBMI/VBMI2/VPOPCNDQ/BIT ALG/GFNI/VAES/VPCLMULQDQ
Lanes

- Older AVX ymm registers are split into lanes
- Instructions (mainly) operate in lanes
- Can be tricky to move data between lanes
- Limitation on AVX2 code
K-mask registers

• A new set of registers \( k_0 - k_7 \) that allow the destination register to remain unchanged or set to zero

• For example, addition, but only some values:
  • paddw zmm1{\( k_1 \)}, zmm2, zmm3

• kmovX instructions to manipulate k-mask registers
vpermb

• Byte shuffles (permute) are the one of the most important instructions in multimedia
• Similar to existing pshufb instruction but cross-lane
• Need to use k-masks with vpermb to zero out a byte
• e.g for zigzag scan
• Also, VPERMT2B, permute from two registers
Variable shifts

- `vpsrlvw/vpsrlvd/vpsrlvq` – variable right shift (logical)
- `vpsllvw/vpsllvd/vpsllvq` – variable left shift (logical)
- (letter soup, especially arithmetic shifts!)
- Historically had to use multiple instructions and various trickery to achieve right shifts. Had many limitations.
- Faster than multiply for left shifts
vpternlogd

• The kitchen sink of instructions!
• Allows a programmable truth table to be implemented per bit input in each register
• Can replace up to eight instructions!
• e.g vpternlogd zmm0, zmm1, zmm2, 0xca
  • zmm0 = zmm0 ? zmm1 : zmm2; (for each bit)
• [Website Link](http://0x80.pl/articles/avx512-ternary-functions.html)

<table>
<thead>
<tr>
<th>inputs</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>constant</td>
<td>0xca</td>
</tr>
</tbody>
</table>
Example: v210enc (1)

• Takes 3x 8-bit samples, extends to 10-bit and packs into 32-bits

```
.loop:
  movu    ym1, [yq + 2*widthq]
  vinserti32x4 m1, [uq + 1*widthq], 2
  vinserti32x4 m1, [vq + 1*widthq], 3
  vpermb   m1, m2, m1 ; uyvx yuyx vyux vvux
  CLIPUB m1, m4, m5
  pmaddubsw m0, m1, m3 ; shift high and low samples of each dword and mask out other bits
  pslld m1, 4 ; shift center sample of each dword
  vpternlogd m0, m1, m6, 0xd8 ; C?B:A ; merge and mask out bad bits from B
  movu [dstq], m0
  [...]  
jl .loop
```
Example: v210enc (2)

- Benchmarks (decicyles)
  - Skylake
    - v210_planar_pack_8_c: 2373.5
    - [...]  
    - v210_planar_pack_8_avx2: 194.0
    - v210_planar_pack_8_avx512: 174.0
    - vptestnlogd on a shorter ymm register
  - Ice Lake
    - v210_planar_pack_8_c: 2743.6
    - v210_planar_pack_8_avx2: 246.6
    - v210_planar_pack_8_avx512: 238.6
    - v210_planar_pack_8_avx512icl: 122.1
    - vpermb and zmm nearly twice as fast as avx512 ymm, and more than twenty times faster than C!
What AVX-512 code next?

• Anything involving line/frame based processing
  • e.g filters, scalers etc.
• Comparisons
• `vpternlogd` in many places (e.g 3-way boolean)
• Also change variable shifts in many places

• Intel manual is very verbose (useful in some cases)
• [https://www.officedaytime.com/simd512e/](https://www.officedaytime.com/simd512e/)
Any questions?