Image filtering using MMX technology

Report about motivation article

vasko.anton@gmail.com

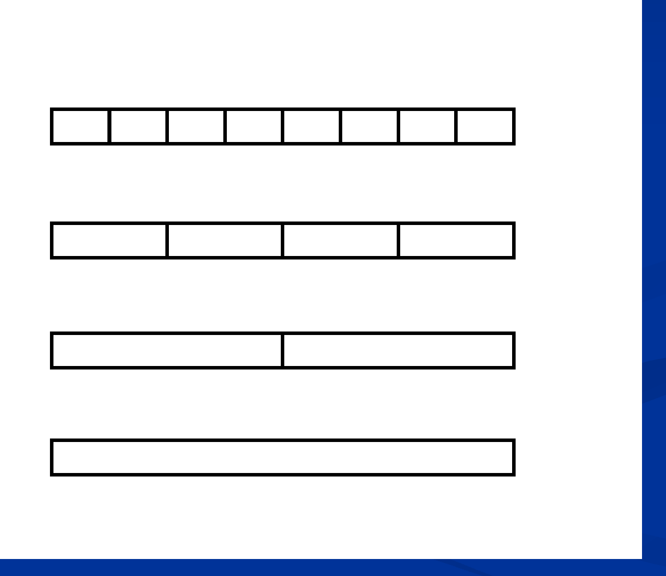
Content

- Brief overview of MMX technology
- Sample assignment
- MMX optimization of filtering
- Conclusion

MMX – data types

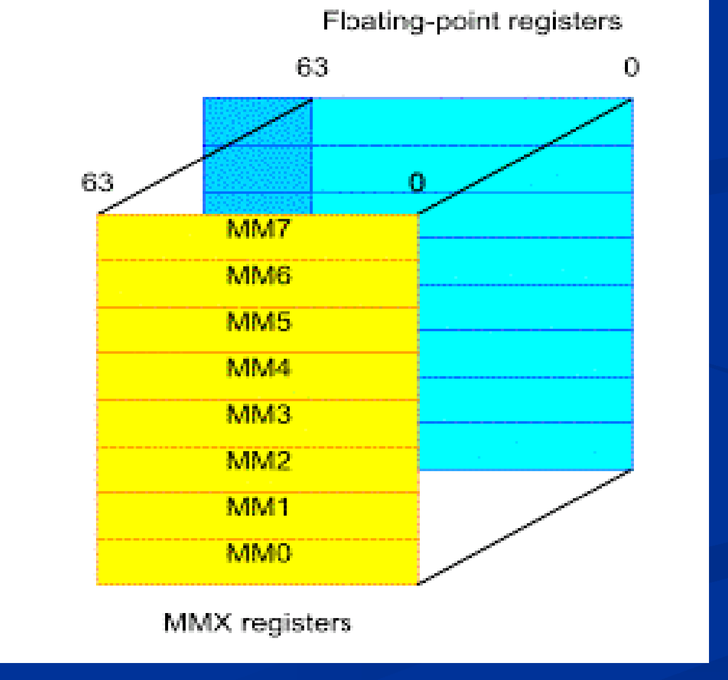
4 new data types (64 bits wide)

1. packed byte
2. packed word
3. packed doubleword
4. quadword



MMX - registers

- 8 new registers MM0 – MM7, 64 bits wide
- physically mirrored in FP stack



MMX - instructions

57 new instructions:

1. arithmetic (padd, psub, pmul ...)
2. comparison (pcmpeq, pcmpgt)
3. conversion (pack,punpck)
4. logical (pand, pandn, por, pxor)
5. shift (psll, psrl, psra)
6. data transfer (movq, movd)
7. state management (emms)

Reasons for MMX optimization analysis

- Image filtering is computational expensive (e.g. for picture of size 256x256 and separable 3x3 filter 256 * 256 * (3+3) = 393 216 multiplication is needed)
- Often used in multimedia
- MMX = MultiMedia eXtension
- Is image filtering suitable candidate for MMX optimization?

Sample assignment

- 2D image with 8-bit color values
- 3x3 separable filter kernel
- Filter coefficient are signed 8 bit with sum of 64 – normalization is shifting (instead of divide)

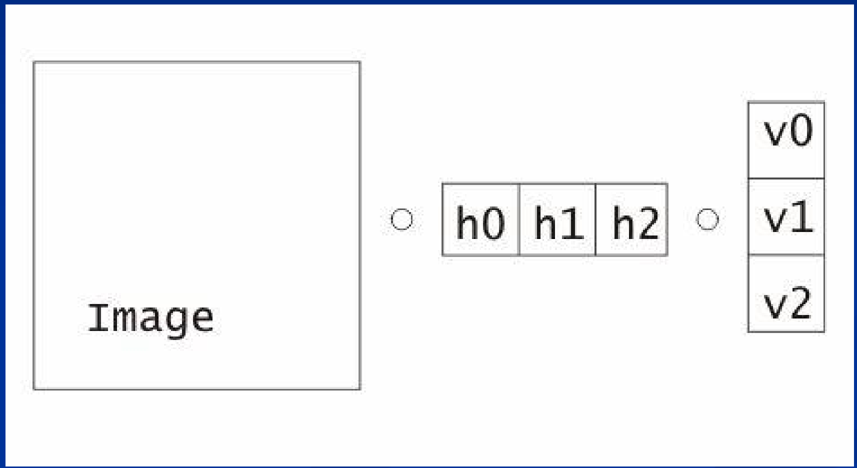
Basic Strategy

- MMX multiplying is on packed words (16-bits) =>

1. Read 8-bit pixels
2. Unpack them to 16-bits
3. Multiply by the filter coefficients (already preformatted into 16-bit data)
4. Sum products
5. Produce normalized result using arithmetic right shift
6. Convert back from word to byte format

Filter operations

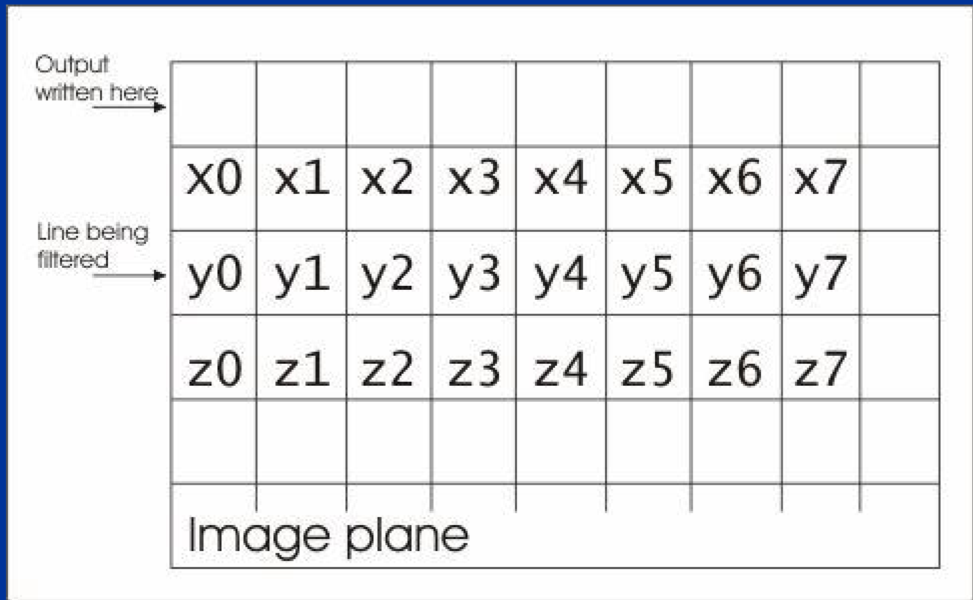
- Filter operations:



- 2 steps – horizontal and vertical filtering
- Let start with the vertical one, because it is easier and fits very nicely the parallelism of the MMX instructions

Vertical filter strategy

- [y0’, y1’, y2’, y3’] = [x0, x1, x2, x3] * v0 +
  [y0, y1, y2, y3] * v1 +
  [z0, z1, z2, z3] * v2



Unscheduled code

```
; Vertical pass of a 3x3 separable image filter
; Assume:
;   esi points to line of X's
;   edi points to output line (the line before X's
;   edx contains the line-to-line "stride"
;   memv0 is the memory location containing [v0,v0,v0,v0] (16 bit values)
;   memv1 is the memory location containing [v1,v1,v1,v1] (16 bit values)
;   mm6 contains zero
;   mm7 contains [v2,v2,v2,v2] (16 bit values)
; This code does the four low-order pixels in a group of eight.
; To do the high-order pixels use the same code with punpckhbw
; instead of punpcklbw
;
movq        mm0, [esi]          ; Load X's
punpcklbw   mm0, mm6            ; Unpack with zeros to get words
pmullw      mm0, memv0          ; Multiply by v0
movq        mm1, [esi + edx]    ; Load Y's
punpcklbw   mm1, mm6            ; Unpack with zeros to get words
pmullw      mm1, memv1          ; Multiply by v1
movq        mm2, [esi + 2* edx] ; Load Z's
punpcklbw   mm2, mm6            ; Unpack with zeros to get words
pmullw      mm2, mm7            ; Multiply with v1
paddsw      mm0, mm1            ; Accumulate
paddsw      mm0, mm2            ; Finish accumulation
psraw       mm0,6               ; Normalize
packuswb    mm0, mm0            ; Pack into four low-order bytes
movd        [edi], mm0          ; Write result into memory
```

Improvements

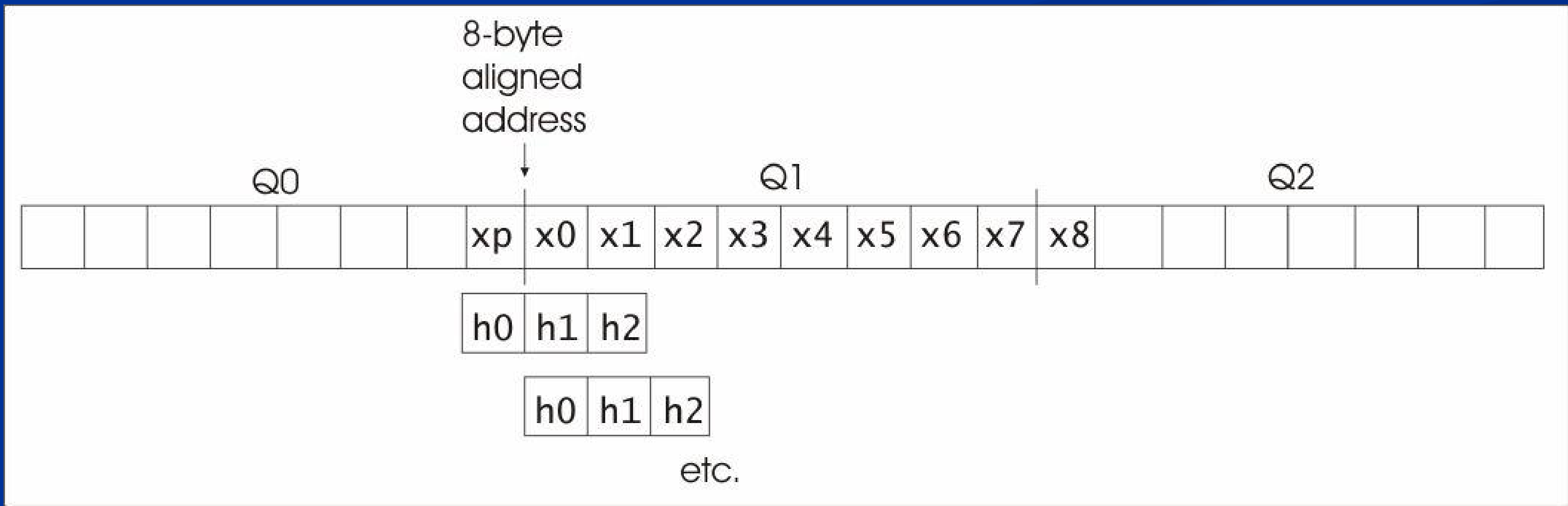
- Resources analyze – 1 register for unpacking, 3 registers for 3 image lines =>
- Unwind the loop twice (in the x direction) – 6+1 registers, and
- Interleave two copies of the code (Software-pipelining technique)
- Schedule - it is possible to obtain perfect pairing (without stalls) of this code (not shown)

Vertical filter summary

- Operating on 4 (not 8) pixels in parallel
- Operating on the original pixels – writing result 2 lines above the original
- This shifts the image but can be compensated elsewhere, e.g. in horizontal filter
- Efficient utilization of processor's L1 cache

Horizontal filter strategy

- [x0’, x1’, x2’, x3’, x4’, x5’, x6’, x7’] =
  h0 * [xp, x0, x1, x2, x3, x4, x5, x6] +
  h1 * [x0, x1, x2, x3, x4, x5, x6, x7] +
  h2 * [x1, x2, x3, x4, x5, x6, x7, x8]



Synthesis of the sets

- Not everything in memory can be aligned
  =>
- [x0, x1, x2, x3, x4, x5, x6, x7] = Q1
- [xp, x0, x1, x2, x3, x4, x5, x6] =
  (Q0>>56) | (Q1<<8)
- [x1, x2, x3, x4, x5, x6, x7, x8] =
  (Q2<<56) | (Q1>>8)

Unscheduled code (1)

```
; Horizontal pass of a 3x3 separable image filter
; Assume:
;    esi points to beginning of input line
;    edi points to beginnig of output line
;    ecx is an offset within the line
;    memv0 is the memory location containing [v0,v0,v0,v0] (16 bit values)
;    memv1 is the memory location containing [v1,v1,v1,v1] (16 bit values)
;    memv2 is the memory location containing [v2,v2,v2,v2] (16 bit values)
;    mm6 contains zero
;
movq        mm0, [esi + 8* ecx -8]   ; Load Q0
movq        mm1, [esi + 8* ecx]      ; Load Q1
movq        mm2, mm1                 ; Make a copy of mm1
movq        mm3,mm1                  ; Make another copy of mm1
movq        mm0, [esi + 8* ecx +8]   ; Load Q2
psrlq       mm0, 56                  ; Q0>>56
psllq       mm2, 8                   ; Q1<<8
por         mm0, mm2                 ; mm0 now has [x6, .., xp]
psllq       mm4, 56                  ; Q2<<56
psrlq       mm3, 8                   ; Q1>>8
por         mm4, mm3                 ; mm4 now has [x8, .., x1]
movq        mm2, mm0                 ; make a copy of the set
movq        mm3, mm1                 ; 1. set: mm0, mm1, mm4
movq        mm5, mm4                 ; 2. set: mm2, mm3, mm5
```

Unscheduled code (2)

```
;
; Low 4 pixels
;
punpcklbw    mm0, mm6           ; Unpack with zeros to get words
pmullw       mm0, memv0         ; Multiply by v0
punpcklbw    mm1, mm6           ; Unpack with zeros to get words
pmullw       mm1, memv1         ; Multiply by v1
punpcklbw    mm4, mm6           ; Unpack with zeros to get words
pmullw       mm4, mm7           ; Multiply with v1
paddsw       mm0, mm1           ; Accumulate
paddsw       mm0, mm4           ; Finish accumulation
psraw        mm0,6              ; Normalize
;
; High 4 pixels
;
punpckhbw    mm2, mm6           ; Unpack with zeros to get words
pmullw       mm2, memv0         ; Multiply by v0
punpckhbw    mm3, mm6           ; Unpack with zeros to get words
pmullw       mm3, memv1         ; Multiply by v1
punpckhbw    mm5, mm6           ; Unpack with zeros to get words
pmullw       mm5, mm7           ; Multiply with v1
paddsw       mm2, mm3           ; Accumulate
paddsw       mm2, mm5           ; Finish accumulation
psraw        mm2,6              ; Normalize
;
packuswb     mm0, mm2           ; Pack to bytes
movd         [edi +8* ecx], mm0 ; Write result into memory
```

Horizontal filter summary

- 2 sets of input – once to filter low-order 4 pixels, once for the high-order 4 pixels
- It is possible to obtain perfect pairing (without stalls) of the code (not shown)

Conclusion

- Image filtering is rewarding topic for MMX optimization
- My current work – extension of these ideas:

1. From 2D (image filtering) to 3D (volume filtering)
2. Bigger kernel (5x5x5, 7x7x7)
3. Floating-point calculations

Bibliography

1. Intel Corporation: The Complete Guide to MMX Technology, McGraw-Hill, Inc.

Thanks for your attention !

vasko.anton@gmail.com