HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2020.05.31.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

SIMD feldolgozás

Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék
ghorvath@hit.bme.hu, belso@hit.bme.hu

Ez a videó a BME Mérnökinformatikus BSc képzésének a tananyaga.

Személyes használatra bárki szabadon megnézheti és birtokolhatja.

A BME Villamosmérnöki és Informatikai Karán kívüli képzésekben csak a Kar engedélyével lehet felhasználni!

FLYNN TAXONÓMIA

- Flynn felosztása az utasítások és adatok viszonya szerint:

- **SISD** (single instruction, single data):
  - Egy utasítássorozat végrehajtása skalár adatokon
  - Ezt tanultuk eddig
- **SIMD** (single instruction, multiple data)
  - Egy utasítássorozat több adaton végez műveletet egyszerre
  - Vektorprocesszorok/tömbprocesszorok, stb.
- **MIMD** (multiple instruction, multiple data)
  - Több utasítássorozat több adaton dolgozik
  - Multiprocesszoros rendszerek
- **MISD** (multiple instruction, single data)
  - Hibatűrő rendszerekben

Vektorprocesszorok

VEKTORPROCESSZOROK

- Klasszikus skalár adattípus és műveletek mellett / helyett
  - Vektor adattípus
  - Vektorkezelő utasítások
- Minden modern szuperszámítógép rendelkezik ilyen képességgel

PÉLDA

- C program:

```
for (i=0; i<64; i++)
    C[i] = A[i] + B[i];
```

- Klasszikus (skalár) megoldás:

```
R4←64
loop:
    D1 ← MEM[R1]
    D2 ← MEM[R2]
    D3 ← D1 + D2
    MEM[R3] ← D3
    R1 ← R1 + 8
    R2 ← R2 + 8
    R3 ← R3 + 8
    R4 ← R4 - 1
    JUMP loop IF R4!=0
```

Vektoros megoldás:

```
VLR ← 64
V1 ← MEM[R1]
V2 ← MEM[R2]
V3 ← V1 + V2
MEM[R3] ← V3
```

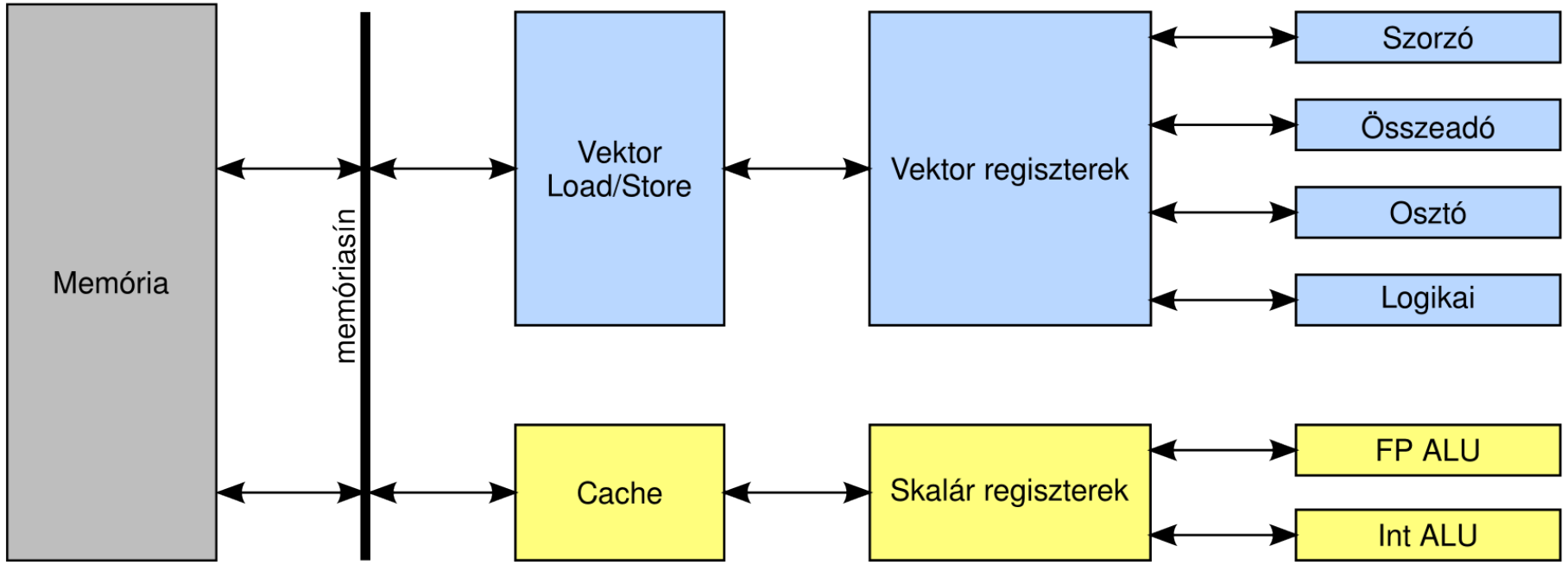
PÉLDA

- Miért is jobb vektorprocesszorral?
  - Rövidebb, tömörebb kód
  - **Nincs szükség ciklusra!**
    - Mi a baj a ciklussal?
      - Minden körben újra és újra
        - Le kell hívni
        - Dekódolni
        - Végrehajtani

        a ciklusmag utasításait.
      - Minden körben procedurális egymásrahatás
      - 64-szer kell elágazásbecslést végezni!
  - **A vektorműveletek implicit feltételezik, hogy a vektorelemek függetlenek**
    - Sok műveleti egységgel vagy/és igen mély pipeline-al nagy teljesítmény érhető el

VEKTORPROCESSZOROK FELÉPÍTÉSE

- Típusok:
  - Regiszter-regiszter
  - Memória-memória
- Mostantól csak regiszter-regiszter:



VEKTORPROCESSZOROK MŰKÖDÉSE

- Miért csak a skalár operandusoknak jár cache?
  - Vektorok-műveleteken máshogy gyorsítunk:
    → Spéci memóriakezeléssel
- Miért lassú egy skalár memóriaolvasás?
  - Cím kiadás → adat megérkezés között több órajelciklus
- Miért szerencsések a vektor memóriaműveletek?
  - Vektor beolvasás: fix (nagy) számú memóriaművelet
  - Kiadom az egyik elem címét
  - Következő ciklusban még nem jön meg, de attól még kiadhatom egy másik elem címét
  - …, stb. → pipeline-szerű megoldás
  - Ehhez kell, hogy
    az egymás utáni elemek külön memória-bankban legyenek

FÉSÜS CÍMZÉS

Órajel	Bank					
	0	1	2	3	4	5
1		15636				
2		Fogl.	15640			
3		Fogl.	Fogl.	15644		
4		Fogl.	Fogl.	Fogl.	15648	
5		adat[0]	Fogl.	Fogl.	Fogl.	15652
6	15656		adat[1]	Fogl.	Fogl.	Fogl.
7	Fogl.	15660		adat[2]	Fogl.	Fogl.
8	Fogl.	Fogl.	15664		adat[3]	Fogl.
9	Fogl.	Fogl.	Fogl.	15668		adat[4]
10	adat[5]	Fogl.	Fogl.	Fogl.	15672	
11		adat[6]	Fogl.	Fogl.	Fogl.	15676
12	15680		adat[7]	Fogl.	Fogl.	Fogl.
13	Fogl.	15684		adat[8]	Fogl.	Fogl.

MŰVELETI EGYSÉGEK

- Vektorműveletek gyorsítása:
  - **Műveleti egységek többszörözésével**
    - Vektorelemek függetlenek
    - pl. 4 műveleti egység → egyszerre 4 elem számolása
  - **Mély adat-pipeline alkalmazásával**
    - Adatpipeline? Az mi?
    - Lebegőpontos számok: Szám = $(-1)^s * c * 2^q$
    - Példa: lebegőpontos összeadás (4 fázis):
      - Ellenőrizzük, valamely operandus nulla-e
      - A 2 operandus azonos karakterisztikára hozása
      - Összeadás elvégzése
      - Eredmény normál alakra hozása

ADAT-PIPELINE

- Példa: lebegőpontos szorzás (5 fázis)
  - Ellenőrizzük, valamely operandus nulla-e
  - Karakterisztikákat összeadjuk
  - Mantisszákat összeszorozzuk
  - Eredmény előjel bitjét előállítjuk
  - Eredményt normál alakra hozzuk
- Ha egy vektorelemre az első fázis kész, rögtön kezdi a következő elemre is az első fázist → futószalag

ADAT-PIPELINE

	1	2	3	4	5	6	7	8	9	10	11
V2[0] ← V0[0]+V1[0]	A0	A1	A2	A3							
V2[1] ← V0[1]+V1[1]		A0	A1	A2	A3						
V2[2] ← V0[2]+V1[2]			A0	A1	A2	A3					
V2[3] ← V0[3]+V1[3]				A0	A1	A2	A3				
V2[4] ← V0[4]+V1[4]					A0	A1	A2	A3			
V2[5] ← V0[5]+V1[5]						A0	A1	A2	A3		
V2[6] ← V0[6]+V1[6]							A0	A1	A2	A3	
V2[7] ← V0[7]+V1[7]								A0	A1	A2	A3

ADAT-PIPELINE

- 2 műveleti egységgel:

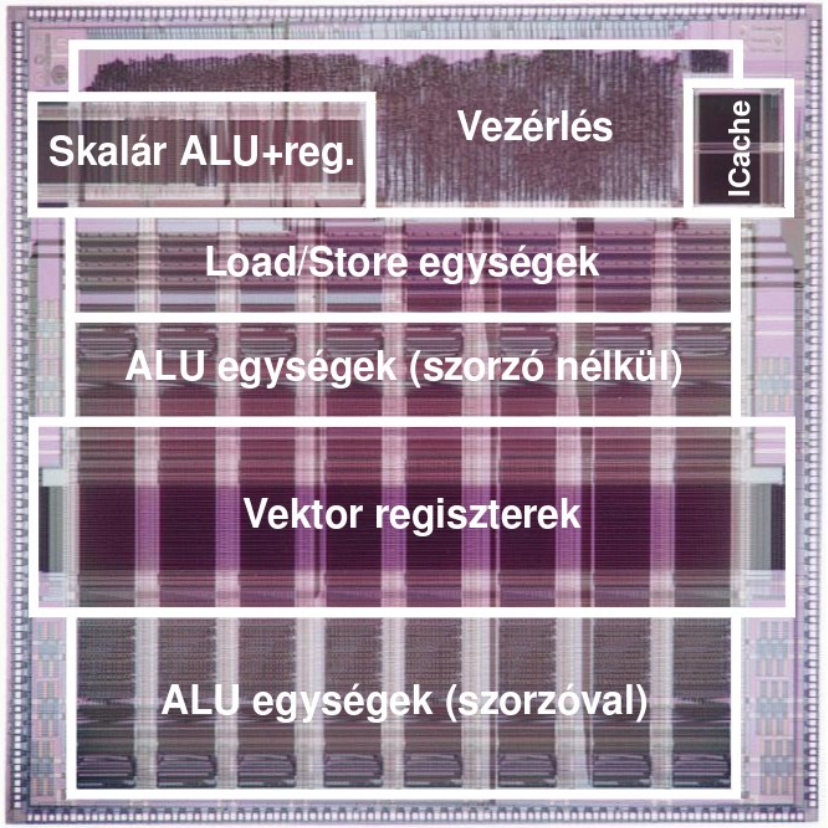
	1	2	3	4	5	6	7	8	9	10	11
V2[0] ← V0[0]+V1[0]	A0	A1	A2	A3							
V2[1] ← V0[1]+V1[1]	A0	A1	A2	A3							
V2[2] ← V0[2]+V1[2]		A0	A1	A2	A3						
V2[3] ← V0[3]+V1[3]		A0	A1	A2	A3						
V2[4] ← V0[4]+V1[4]			A0	A1	A2	A3					
V2[5] ← V0[5]+V1[5]			A0	A1	A2	A3					
V2[6] ← V0[6]+V1[6]				A0	A1	A2	A3				
V2[7] ← V0[7]+V1[7]				A0	A1	A2	A3				

ADAT-PIPELINE

- Nagyon fontos:
  - Ez egy olyan pipeline, melyben
    - **Nincs egymásrahatás!**
      → Nem kell egymásrahatást vizsgáló és kezelő logika
      → Emiatt tetszőleges mély lehet
      → Emiatt tetszőleges széles lehet
  - Egyetlen korlátozó tényező:
    - Hány részfázisra tudunk bontani egy aritmetikai műveletet

PÉLDA 1.

- Berkely T0 (Torrent-0, 1995)
- 32 hosszú regiszterek
- 1 sáv: 4 elem kezelése (ábrán 8 függőleges sáv)
- Minden sávban 2 ALU
- … de csak az egyik tud szorozni

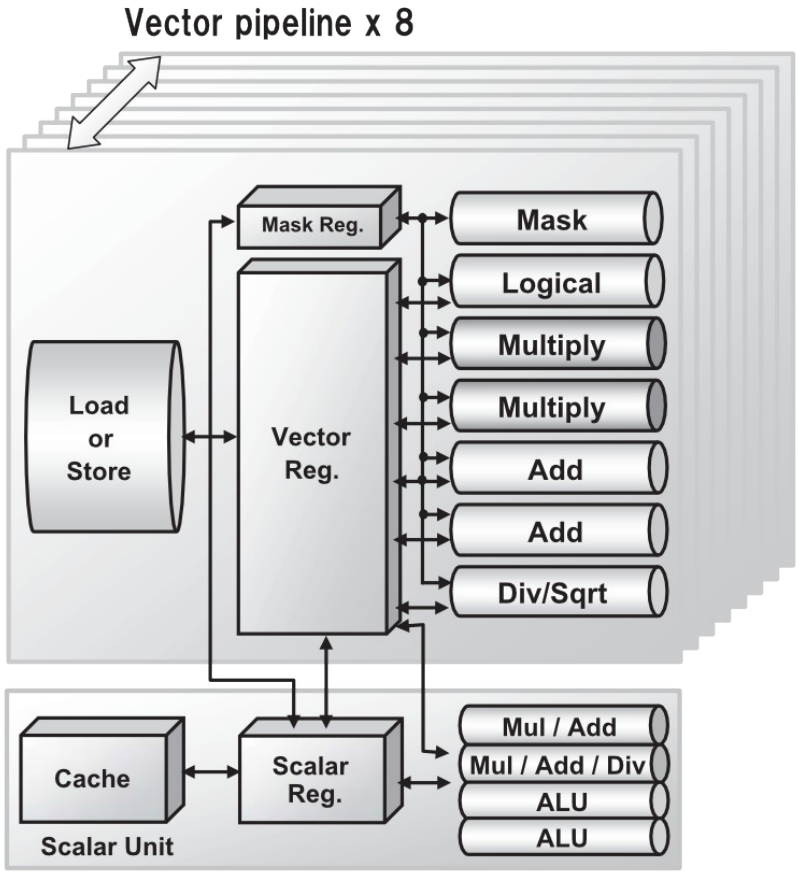


PÉLDA 2.

- NEC SX-9 szuperszámítógép processzorra
- 2008-as év leggyorsabb szuperszámítógépe
- Német meteorológiai központ ezzel dolgozik (2 db)
- 2011-ben: 976 db vektorprocesszor + 31 TB memória
- 1 processzor:
  - 256 hosszú vektorokat kezel
  - 8 pipeline szervezésű műveleti egysége van
- Vektor ALU: 3.2 GHz
- Skalár egység (4 utas out-of-order): 1.6 GHz

PÉLDA 2.

- A NEC SX-9 felépítése:



Vektorprocesszorok jellegzetes megoldásai

STRIP-MINING

- Támogatott vektor méret hardveresen kötött:
  MVL (Maximum Vector Length)
- Ritkán kell pont ekkora vektor
  - Ha kisebbel számolunk:
    - VLR beállítása (Vector Length Register)
    - Kisebb VLR → kisebb futási idő
  - Ha nagyobbal számolunk:
    - Vektorunk MVL méretű darabokra szeletelése
    - Minden szeletre a művelet végrehajtása

  → ez a **Strip-mining**

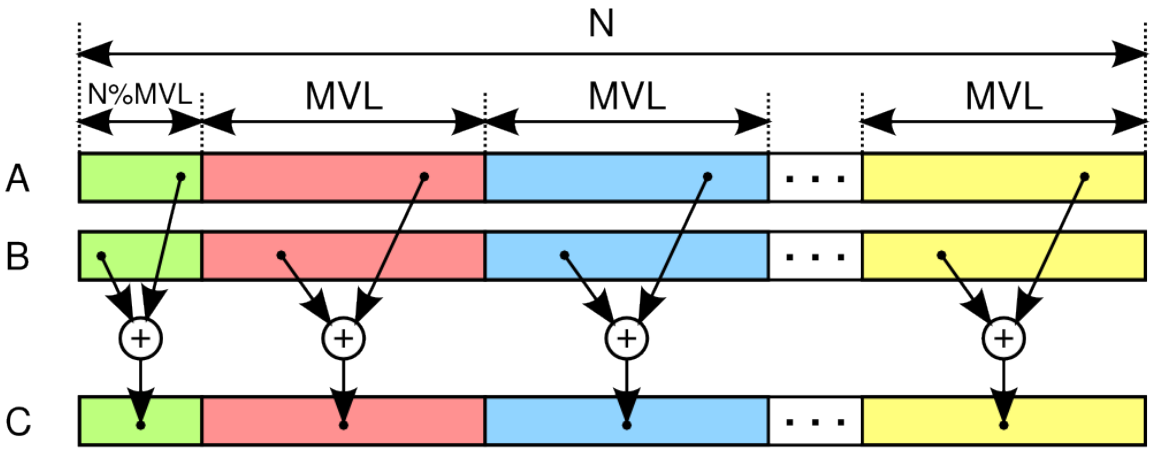
STRIP-MINING PÉLDA

- C kód:

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

Vektorizálva:

```
    VLR ← R0 % MVL
loop:
    V1 ← MEM[R1]
    V2 ← MEM[R2]
    V3 ← V1 + V2
    MEM[R3] ← V3
    R4 ← VLR * 8
    R1 ← R1 + R4
    R2 ← R2 + R4
    R3 ← R3 + R4
    R0 ← R0 - VLR
    VLR ← MVL
    JUMP loop IF R0!=0
```



FELTÉTELES VEKTORMŰVELETEK

- Maszk regiszter:
- Ahol =0, arra az elemre a köv. vektorművelet nem vonatkozik
- C program:

```
for (i=0; i<N; i++)
    if (B[i]>0)
        C[i] = A[i] / B[i];
```

Vektorprocesszorral:

```
V1 ← MEM[R1]
MASK ← V1>0
V0 ← MEM[R0]
V2 ← V0 / V1
MEM[R2] ← V2
```

- Kétféleképpen valósítható meg:
  - **Naív**: Vektor ALU mindig minden elemre számol, de a maszkolt elemek eredményét nem tárolja
  - **Hatékony**: A Load/Store és az ALU a maszkolt elemeket kihagyja → gyorsabb végrehajtás

MŰVELET LÁNCOLÁS

- RAW egymásrahatások vektorprocesszorban is vannak:
- `V1 ← MEM[R1]`
  `V3 ← V1 + V2`
  `V5 ← V3 * V4`

- A forwarding vektorprocesszoros megfelelője:
  - **Művelet láncolás** (vector chaining)
  - Függő utasításnak nem kell megvárnia az előző utasítás végét
  - Ahogy V1 első eleme megvan, V3 első eleme el is kezdhető
  - Amint V3 első eleme megvan, V5 első eleme el is kezdhető

MŰVELETLÁNCOLÁS PÉLDA

Utasítás	1	2	3	4	5	6	7	8	9
V1[0]←MEM[R1+0]	T0	T1							
V1[1]←MEM[R1+8]	T0	T1							
V1[2]←MEM[R1+16]		T0	T1						
V1[3]←MEM[R1+24]		T0	T1						
V1[4]←MEM[R1+32]			T0	T1					
V1[5]←MEM[R1+40]			T0	T1					
V3[0]←V1[0]+V2[0]			A0	A1	A2				
V3[1]←V1[1]+V2[1]			A0	A1	A2				
V1[6]←MEM[R1+48]				T0	T1				
V1[7]←MEM[R1+56]				T0	T1				
V3[2]←V1[2]+V2[2]				A0	A1	A2			
V3[3]←V1[3]+V2[3]				A0	A1	A2			

...

MŰVELETLÁNCOLÁS PÉLDA

Utasítás	1	2	3	4	5	6	7	8	9
V1[8]←MEM[R1+64]					T0	T1			
V1[9]←MEM[R1+72]					T0	T1			
V3[4]←V1[4] + V2[4]					A0	A1	A2		
V3[5]←V1[5] + V2[5]					A0	A1	A2		
V1[10]←MEM[R1+80]						T0	T1		
V1[11]←MEM[R1+88]						T0	T1		
V3[6]←V1[6] + V2[6]						A0	A1	A2	
V3[7]←V1[7] + V2[7]						A0	A1	A2	
V5[0]←V3[0] * V4[0]						M0	M1	M2	M3
V5[1]←V3[1] * V4[1]						M0	M1	M2	M3

...

SIMD utasításkészlet-kiegészítések

SIMD UTASÍTÁSKÉSZLET KIEGÉSZÍTÉSEK

- Vektor utasítások otthoni használatra is hasznosak
  - Képfeldolgozási célokra
  - 3D grafikai alkalmazásokban / játékokban
  - Egyszerű tudományos alapfeladatok is jól vektorizálhatók
- Számos processzor támogat vektorműveleteket
- De attól még nem lesznek vektorprocesszorok!
  - Nagyon kicsi a vektorméret (legjobb: 256 bites)
  - Nincs VLR
  - Nincs maszk regiszter
  - Nincs művelet láncolás
  - Nincs adat-pipeline
    - Annyi műveleti egység van, amekkora a vektorméret

SIMD UTASÍTÁSKÉSZLET KIEGÉSZÍTÉSEK

- Utasítások fajtái:
  - Vektor-vektor műveletek:
    - Inter-vektor: 2 vektor között. Eredmény: vektor
      - pl. 2 vektor összeadása
    - Intra-vektor: 1 vektor elemein. Eredmény: skalár
      - pl. vektor elemeinek összeadása
    - Vektorelemek átrendezése (shuffling)
  - Skalár-vektor műveletek:
    - pl. minden elem megszorzása egy skalárral
  - Vektor load/store műveletek:
    - Memória ↔ vektorregiszterek közötti adatmozgatás

SIMD UTASÍTÁSKÉSZLET KIEGÉSZÍTÉSEK

Vektor kieg.	Utasításkészlet	Vekt.reg. száma	...hossza	Elemek típusa
MMX	x86	8	64 bit	Int: 8x8, 4x16, 2x32 bit
3DNow	x86	8	64 bit	Float: 2x32 bit
SSE	x86/x64	8	128 bit	Float: 4x32 bit
SSE2-4	x86/x64	8/16	128 bit	Int: 16x8, 8x16, 4x32 bit. Float: 4x32, 2x64 bit
AVX	x86/x64	16	256 bit	Float: 8x32, 4x64 bit
AltiVec	Power	32	128 bit	Int: 16x8, 8x16, 4x32 bit Float: 4x32 bit
NEON	ARM	32/16	64/128 bit	Int: 8x8, 4x16, 2x32 bit Float: 2x32 bit

SIMD MŰVELETEK MAGAS SZINTŰ NYELVEN

- Az SIMD műveletek magas szintű nyelvből is elérhetők!
  → „intrinsic” utasítások
  - Platformfüggő
- Használat:
  - Megfelelő C header fájlok beemelésével
    - Vektor adatípusok jelennek meg (`__m128`, `float32x4_t`)
    - Vektorokon dolgozó függvények jelennek meg

PÉLDA SIMD UTASÍTÁSKÉSZLETRE

- Színtelítettség növelése, SIMD nélkül:

```
void saturate () {

    float r, g, b, p, val;

    for (int i=0; i<height*width; i++) {
        r = *srcR;
        g = *srcG;
        b = *srcB;

        p = sqrt (r*r + g*g + b*b);
        val = p + (r - p) * 1.5f;
        *dstR = val>255.0 ? 255.0 : val<0 ? 0 : val;
        val = p + (g - p) * 1.5f;
        *dstG = val>255.0 ? 255.0 : val<0 ? 0 : val;
        val = p + (b - p) * 1.5f;
        *dstB = val>255.0 ? 255.0 : val<0 ? 0 : val;

        srcR++; srcG++; srcB++;
        dstR++; dstG++; dstB++;
    }
}
```

SSE2 UTASÍTÁSKÉSZLETTEL

```
#include <xmmintrin.h>
void saturateSSE2 () {

    float p, val;
    __m128 r0, r1, r2, r3, r4;
    const __m128 r5 = {1.5f, 1.5f, 1.5f, 1.5f};
    const __m128 r6 = {0.0f, 0.0f, 0.0f, 0.0f};
    const __m128 r7 = {255f, 255f, 255f, 255f};

    for (int i=0; i<height*width; i+=4) {

        r1 = _mm_load_ps (srcR);
        r0 = r1;
        r0 = _mm_mul_ps (r0, r1);

        r2 = _mm_load_ps (srcG);
        r4 = r2;
        r4 = _mm_mul_ps (r4, r2);
        r0 = _mm_add_ps (r0, r4);

        r3 = _mm_load_ps (srcB);
        r4 = r3;
        r4 = _mm_mul_ps (r4, r3);
        r0 = _mm_add_ps (r0, r4);

        r0 = _mm_sqrt_ps (r0);

        r1 = _mm_sub_ps (r1, r0);
        r1 = _mm_mul_ps (r1, r5);
        r1 = _mm_add_ps (r1, r0);
        r1 = _mm_min_ps (r1, r7);
        r1 = _mm_max_ps (r1, r6);

        _mm_store_ps (dstR, r1);

        r2 = _mm_sub_ps (r2, r0);
        r2 = _mm_mul_ps (r2, r5);
        r2 = _mm_add_ps (r2, r0);
        r2 = _mm_min_ps (r2, r7);
        r2 = _mm_max_ps (r2, r6);

        _mm_store_ps (dstG, r2);

        r3 = _mm_sub_ps (r3, r0);
        r3 = _mm_mul_ps (r3, r5);
        r3 = _mm_add_ps (r3, r0);
        r3 = _mm_min_ps (r3, r7);
        r3 = _mm_max_ps (r3, r6);

        _mm_store_ps (dstB, r3);

        srcR+=4; srcG+=4; srcB+=4;
        dstR+=4; dstG+=4; dstB+=4;
    }
}
```

NEON UTASÍTÁSKÉSZLETTEL

```
#include <arm_neon.h>
void saturateNEON () {

    float p, val;
    float32x4_t r0, r1, r2, r3, r4;
    const float32x4_t r5 = vdupq_n_f32 (1.5f);
    const float32x4_t r6 = vdupq_n_f32 (0.0f);
    const float32x4_t r7 = vdupq_n_f32 (255.0f);

    for (i=0; i<height*width; i+=4) {

        r1 = vld1q_f32 (srcR);
        r0 = vmulq_f32 (r1, r1);

        r2 = vld1q_f32 (srcG);
        r4 = vmulq_f32 (r2, r2);
        r0 = vaddq_f32 (r0, r4);

        r3 = vld1q_f32 (srcB);
        r4 = vmulq_f32 (r3, r3);
        r0 = vaddq_f32 (r0, r4);

        r0 = vrecpeq_f32 (vrsqrteq_f32 (r0));
```

```
        r1 = vsubq_f32 (r1, r0);
        r1 = vmulq_f32 (r1, r5);
        r1 = vaddq_f32 (r1, r0);
        r1 = vminq_f32 (r1, r7);
        r1 = vmaxq_f32 (r1, r6);

        vst1q_f32 (dstR, r1);

        r2 = vsubq_f32 (r2, r0);
        r2 = vmulq_f32 (r2, r5);
        r2 = vaddq_f32 (r2, r0);
        r2 = vminq_f32 (r2, r7);
        r2 = vmaxq_f32 (r2, r6);

        vst1q_f32 (dstG, r2);

        r3 = vsubq_f32 (r3, r0);
        r3 = vmulq_f32 (r3, r5);
        r3 = vaddq_f32 (r3, r0);
        r3 = vminq_f32 (r3, r7);
        r3 = vmaxq_f32 (r3, r6);

        vst1q_f32 (dstB, r3);

        srcR+=4; srcG+=4; srcB+=4;
        dstR+=4; dstG+=4; dstB+=4;
    }
}
```

PÉLDA SIMD UTASÍTÁSKÉSZLETRE

- Futási idők:

Intel Core i7-2600	SIMD nélkül	15,166 ms
	SSE2	3,829 ms
	AVX	3,698 ms
Intel Pentium 4	SIMD nélkül	139,758 ms
	SSE2	36,355 ms
ARM Cortex A9	SIMD nélkül	155,012 ms
	NEON	44,026 ms

HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

MŰEGYETEM 1782