Pokročilé asemblery

Přednáška 7 – AVX a AVX-2 instrukce

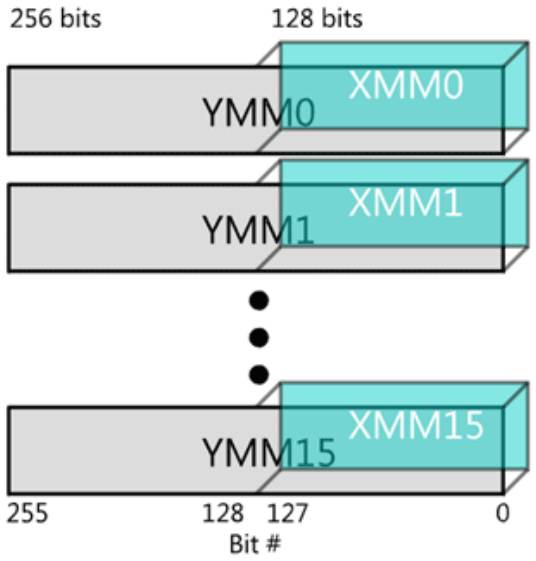
Pár slov úvodem

- Intel® Advanced Vector Extensions (dále Intel AVX)
- Registry rozšířené z 128bitů na 256bitů (předpokládá se rozšíření až na 1024 bitů, již existuje AVX-512)
- Až 4 operandy (většino se používají dva zdrojové a jeden cílový)
- Uvolněné požadavky na zarovnání operandů

- Změna kódovacího schématu (VEX) -> zjednodušené rozšiřování instrukci do budoucna
  - Umožnuje zakódovat více registrů

AVX – běhové prostředí

- Hardware podporující AVX instrukce obsahuje 16 256-bitových registrů YMM0-YMM15
- Kontrolní registr MXCSR
- Většina instrukcí z SSE je rozšířená i do instrukční sady AVX např. mulps xmm0,xmm1 -> vmulps ymm0, ymm0, ymm1

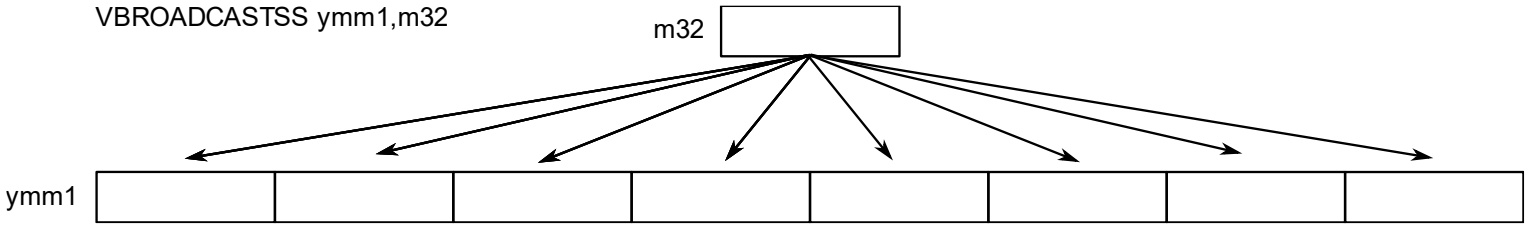


https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

AVX – broadcast instrukce

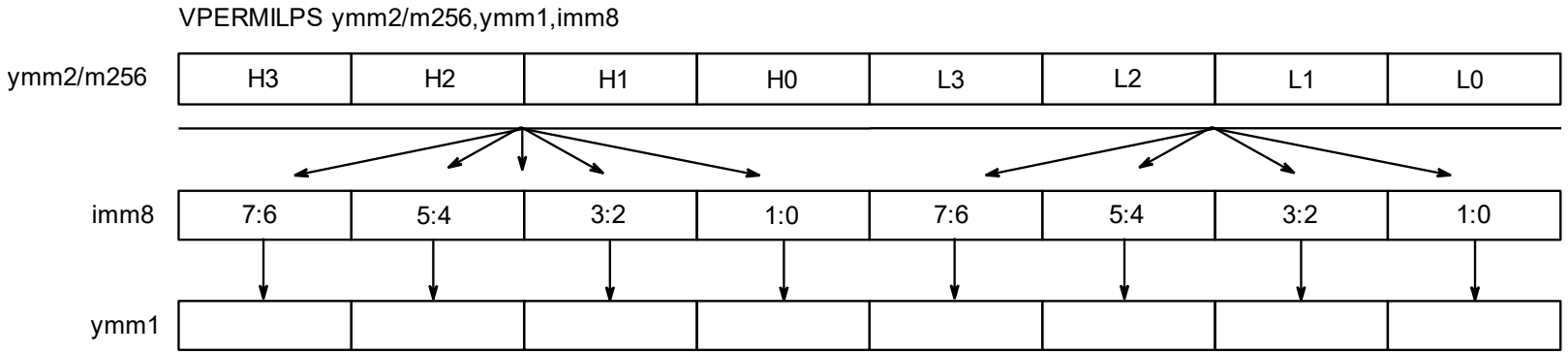
Zkratka pro *float*: SPFP- single precision floating-point
Zkratka pro *double*: SPFP- double precision floating-point

- **vbroadcastss** – rozšíří SPFP z paměťového místa do registru
- **vbroadcastsd** – rozšíří DPFP z paměťového místa do registru
- **vbroadcastf128** – přesune 128 bitovou *packed* hodnotu z paměti do horní a dolní části registru YMMx
- **vpbroadcast(b | w | d | q)** – přesune b | w | d | q z paměti do všech elementů cílového YMM registru



AVX – instrukce pro permutace

- Permutace slouží k přesunům nebo replikaci elementů *packed* datového typu.
- **vperm2f128** – permutace *packed* hodnoty z prvního zdrojové a druhé zdrojového operandu na základě konstanty určené třetím operandem
- **vpermilps** – permutace SPFP hodnoty pomocí konstanty určené druhým zdrojovým operandem
- **vpermilpd** - permutace DPFP hodnoty pomocí konstanty určené druhým zdrojovým operandem
- **vperm2i128, vpermq, vpermps, vpermpd, vpermd**



AVX – maskované přesuny

- **vmaskmovps** – podmíněně zkopíruje SPFP elementy z druhého zdrojového operandu do cílového v závislosti na kontrolní masce určené prvním operandem
- **vmaskmovpd**– podmíněně zkopíruje DPFP elementy z druhého zdrojového operandu do cílového v závislosti na kontrolní masce určené prvním operandem

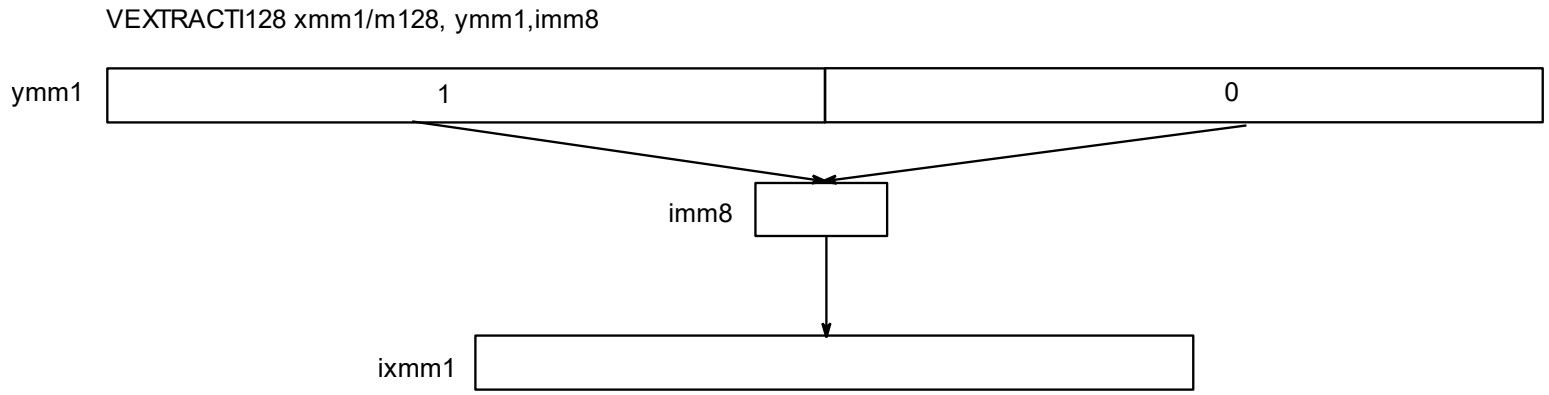
- **vpmaskmovd**- kopírování doubleword elementu
- **vpmaskmovq-** kopírování quadword elementu

Příklad:
VMASKMOVPD - 256-bit load

DEST[63:0] ← IF (SRC1[63]) Load_64(mem) ELSE 0
DEST[127:64] ← IF (SRC1[127]) Load_64(mem + 8) ELSE 0
DEST[195:128] ← IF (SRC1[191]) Load_64(mem + 16) ELSE 0
DEST[255:196] ← IF (SRC1[255]) Load_64(mem + 24) ELSE 0

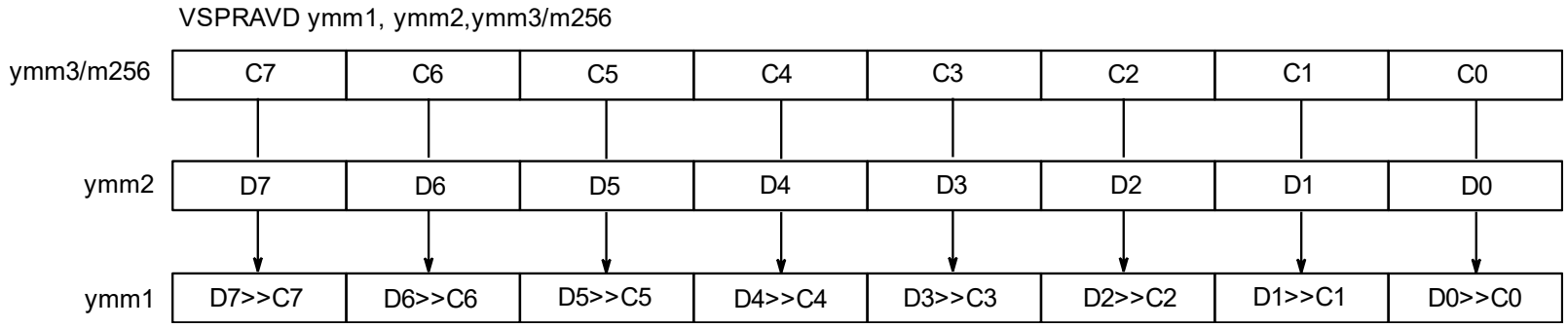
AVX – instrukce pro vložení a extrakci

- **vextracti128** – extrakce 128bitové horní nebo dolní *integer packed* hodnoty ze zdrojového operandu na cílové 128bitové místo
- **vinserti128** – vložení 128bitové *packed* hodnoty ze zdrojového operandu do horní nebo dolní poloviny cílového operandu



AVX – shifty individuální

- **vpsllv(d|q)** – posune každý *doubleword* nebo *quadword* doleva o počet bitů ve druhém zdrojovém operandu
- **vpsravd** - posune každý *doubleword* nebo *quadword* doprava o počet bitů ve druhém zdrojovém operandu, přičemž zachovává znaménkový bit
- **vpsrlv(d|q)** – posune každý *doubleword* nebo *quadword* doleva o počet bitů ve druhém zdrojovém operandu



AVX – gather instrukce

Instrukce shromáždění obsahují instrukce, které podmíněně kopírují datové elementy z paměti do registru XMM a YMM.

- Využití pro data v nesouvislé paměti
- Instrukce pracuje s VSIB (Vector Scale Index Base) adresováním, které se při nepřímém adresování skládá z:

Base—A general-purpose register that points to the start of an array in memory.

Scale—The array element size scale factor (1, 2, 4, or 8).

Index—A vector register (XMM or YMM) that contains the signed doubleword or signed quadword array indices.

Displacement—An optional fixed offset from the start of the array.

- Příklad: vgatherdps xmm0 , [esi+xmm1*4] , xmm2

AVX – FMA instrukce

- **FMA** (Fused Multiply Add) instrukce – jedná se o instrukce, které realizují operaci násobení a sčítání s důrazem na rychlost a přesnost. Při výpočtu vzniká zaokrouhlovací chyba až při výpočtu výsledného výrazu.
- Instrukce realizuje výrazy:
  a = (b * c) + d
  a = (b * c) – d
  a = -(b * c) + d
  a = -(b * c) - d

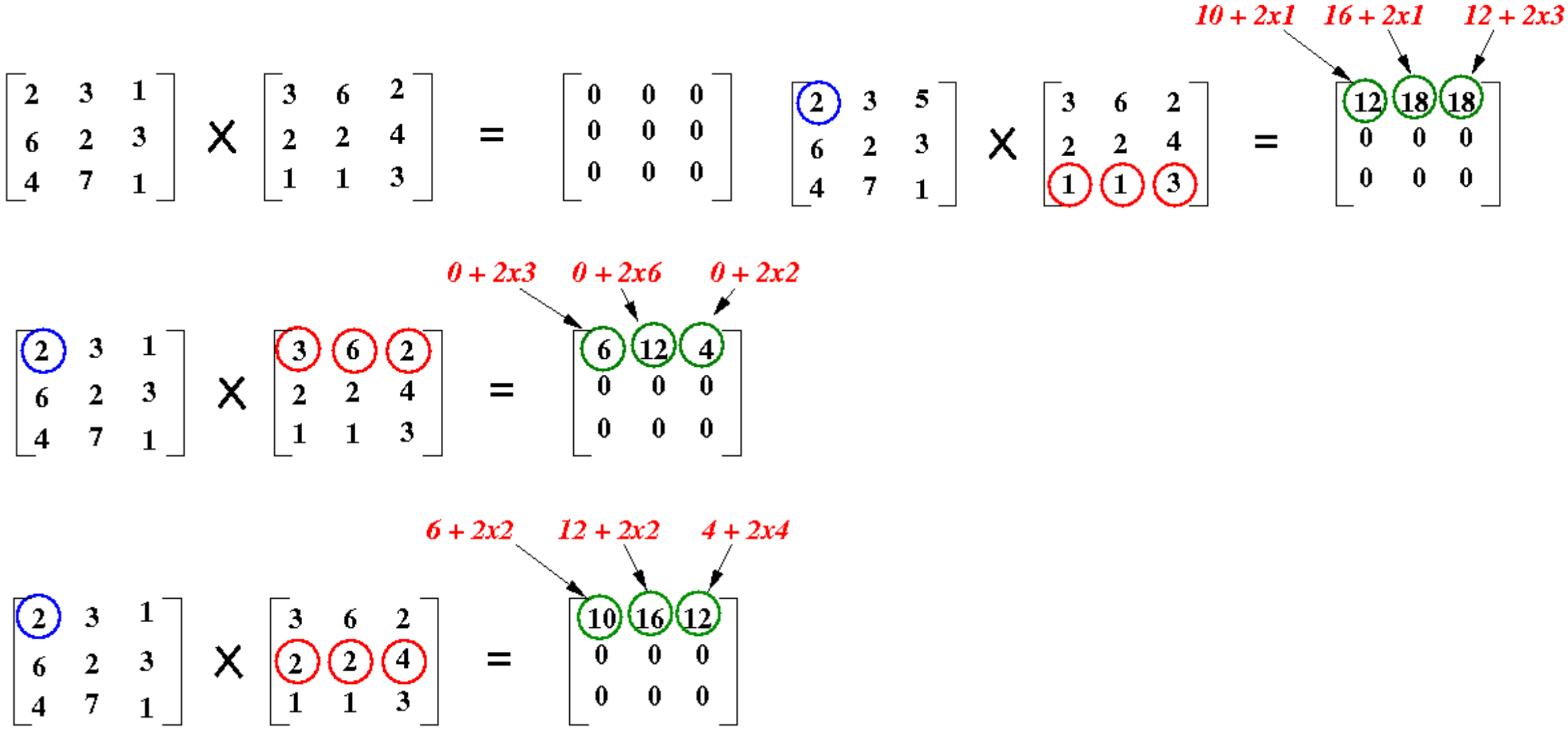
Příklad:

vfmadd132(pd|ps|sd|ss) des = src1 * src3 + src2
vfmadd213(pd|ps|sd|ss) des = src2 * src1 + src3
vfmadd231(pd|ps|sd|ss) des = src2 * src3 + src1

AVX – Obecné instrukce

- **andn** – logický and s invertovaným prvním operandem a druhým operandem
- **bextr** – extrakce spojité bitové části prvního operandu, počáteční pozice a délka je daná druhým operandem
- **blsi** – extrakce nastaveného bitu s nejnižším indexem, výsledný bit se uloží do cílového registru
- **lzcnt** – spočíá počet prvních nul ve zdrojovém registru
- **rdrand** – uloží do cílového registru (16bitový, 32bitový nebo 64bitový) náhodně vygenerované číslo

AVX – násobení matice



http://www.mathcs.emory.edu/~cheung/Courses/561/Syllabus/90-parallel/SIMD.html

AVX – FMA instrukce

```
//Read the eight rows of Matrix B into ymm registers
ymm8 = _mm256_load_ps((float *) (pMatrix2));

ymm9 = _mm256_load_ps((float *) (pMatrix2 + 1*8));
ymm10 = _mm256_load_ps((float *) (pMatrix2 + 2*8));
ymm11 = _mm256_load_ps((float *) (pMatrix2 + 3*8));
ymm12 = _mm256_load_ps((float *) (pMatrix2 + 4*8));
ymm13 = _mm256_load_ps((float *) (pMatrix2 + 5*8));
ymm14 = _mm256_load_ps((float *) (pMatrix2 + 6*8));
ymm15 = _mm256_load_ps((float *) (pMatrix2 + 7*8));
//Broadcast each element of Matrix A Row 1 into a ymm
register
ymm0 = _mm256_broadcast_ss(pIn);
ymm1 = _mm256_broadcast_ss(pIn + 1);
ymm2 = _mm256_broadcast_ss(pIn + 2);
ymm3 = _mm256_broadcast_ss(pIn + 3);
ymm4 = _mm256_broadcast_ss(pIn + 4);
ymm5 = _mm256_broadcast_ss(pIn + 5);
ymm6 = _mm256_broadcast_ss(pIn + 6);
ymm7 = _mm256_broadcast_ss(pIn + 7);
//Multiply A11 times Row 1 of Matrix B
ymm0 = _mm256_mul_ps(ymm0, ymm8);
//Multiply A12 times Row 2 of Matrix B
ymm1 = _mm256_mul_ps(ymm1, ymm9);
//Create the first partial sum
ymm0 = _mm256_add_ps(ymm0, ymm1);
//Repeat for A13, A14, and Rows 3, 4 of Matrix B
ymm2 = _mm256_mul_ps(ymm2, ymm10);
ymm3 = _mm256_mul_ps(ymm3, ymm11);
ymm2 = _mm256_add_ps(ymm2, ymm3);
```

https://software.intel.com/en-us/articles/benefits-of-intel-avx-for-small-matrices

N-body

- Velké množství částic různého druhu např.:
  - Pohyb vesmírných těles
  - Elektrony v polovodičích
  - Pohyb částic prachu ve vzduchu

- Částice na sebe vzájemně působí, tj. vzájemně se ovlivňují.
- Analyticky řešitelný je pouze problém dvou těles, problém tří a více těles nikoliv.

- Problém n-částic lze charakterizovat řešením rovnice:

$$U(n_j) = \sum_{i=0, i \neq j}^{N} F(n_j, n_i) \, ; i, j \in N$$

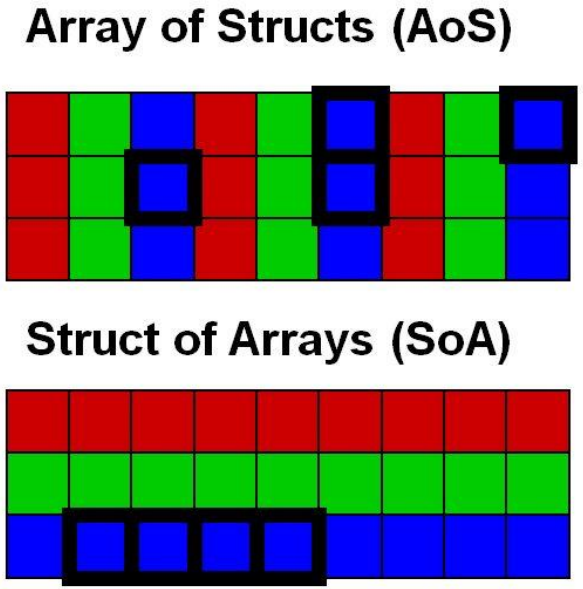
,kde U je výsledná fyzikální veličina vzniklá simulací všech dvou částicových iterací

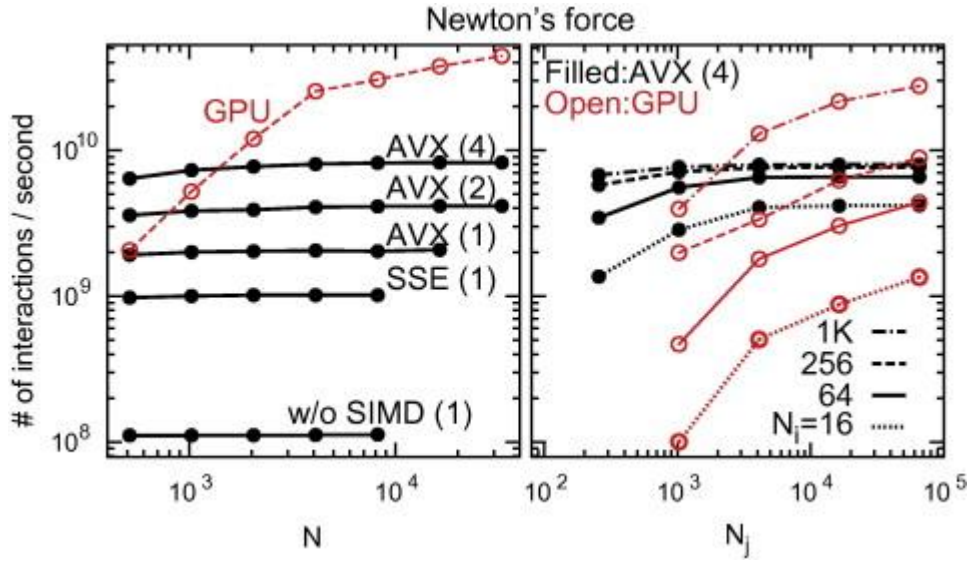
- Z Newtonova gravitačního zákonu:

$$F(x_j) = \sum_{i=0, i \neq j}^{N} g m_j m_i \frac{x_j - x\_i}{|x_j - x_i|^3}$$

N-body 2

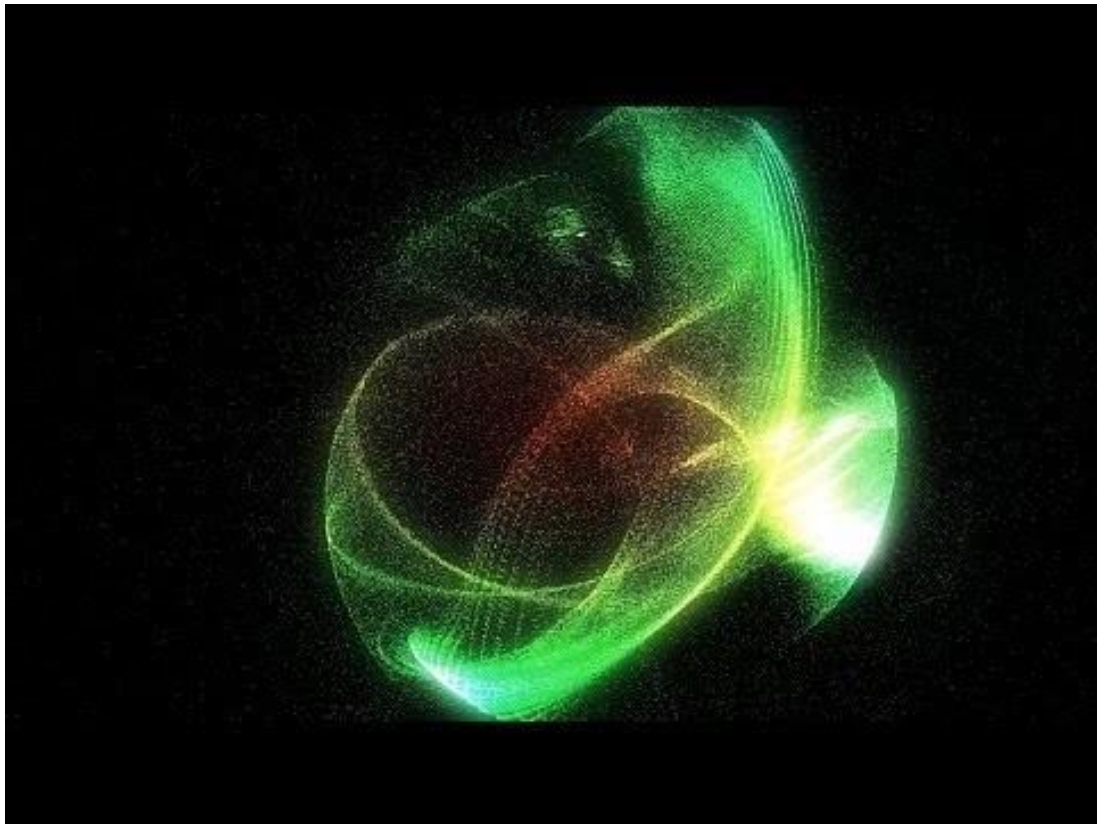
- SOA (Structure of Arrays) vs AOS (Array of structures) – Pole struktur vs Struktura polí
- Z hlediska optimálního návrhu aplikace se jedná o velmi podstatnou věc





http://asc.ziti.uni-heidelberg.de/en/node/18

N-body 3



N-body 3 - cvičení

- Úkolem bude akcelerovat algoritmus kolize koulí

1. Vypočítat, zda došlo ke kolizi tzn. na základě euklidovské vzdálenosti určit, zda je hodnota menší než dvojnásobek poloměru koule
2. Pokud není -> pokračuj stejným směrem
3. Pokud je:
    1. Normalizovaný skalární součin vektoru pro každou kouli
    2. Provedeme výpočet hybnosti
    3. $P = \frac{2 \cdot m1 \cdot m2 \cdot (a1 - a2)}{m1 + m2}$
    4. Aktualizujeme vektor pohybu