Optimizacija upotrebe memorije

Pretpostavke za efikasan pristup memoriji

- Prilikom obrade veće količine podataka, optimalan je sekvencijalni pristup
- Ovo izlazi izvan okvira principa lokaliteta
- Memorijski kontroleri mogu da započnu i održavaju sekvencijalni protok podataka blizu maksimalnih parametara memorije
- Održavanje maksimalnog protoka usklađeno sa instrukcijama i metodama programiranja prilagođenim protočnoj obradi su način da se postignu optimalne performanse

Primer: raspored struktura u memoriji

- Struktura/klasa: osnovna jedinica apstrakcije i organizacije
- C: struct, C++: struct/class: Java: class, Rust: struct
- Grupiše raznorodne podatke vezane za neki entitet
- U realnim situacijama, programi obrađuju skupove struktura: nizovi, skupovi, mape

Osnovni raspored: niz struktura

- AoS (Array of Structures)
- Strukture, veličine diktirane zbirnom veličinom elemenata i poravnanjem, sekvencijalno složene u memoriji
- Ako je potrebno paralelno obrađivati podskupove strukture, javlja se problem prikupljanja vrednosti: elementima se mora pristupati na preskok, što je pogubno po efikasnost pristupa

AoS: 3D vektori

- Tri vrednosti, pokretni zarez, jednostruka preciznost:

```
#[derive(Clone, Copy)]
struct Node {
    x: f32,
    y: f32,
    z: f32,
}
let mut nodes =
    [Node { x: 0.0, y: 0.0, z: 0.0 }; 1024];
```

- Izračunavamo intenzitet vektora: $d = \sqrt{x^2 + y^2 + z^2}$
- Paralelizovano u grupama po 16 za masovno izračunavanje
- Može se videti da već posle prvih nekoliko pristupa izlazimo van tekuće linije keša

Alternativa: SoA

- = „Structure of Arrays“

```
struct Node1 {
    x: [f32; 1024],
    y: [f32; 1024],
    z: [f32; 1024],
}
let mut nodes1 =
    Node1 { x: [0.0; 1024], y: …, z: … };
```

- Ovog puta, kad zahvatamo grupe po 16 to će biti iz sukcesivnih memorijskih lokacija

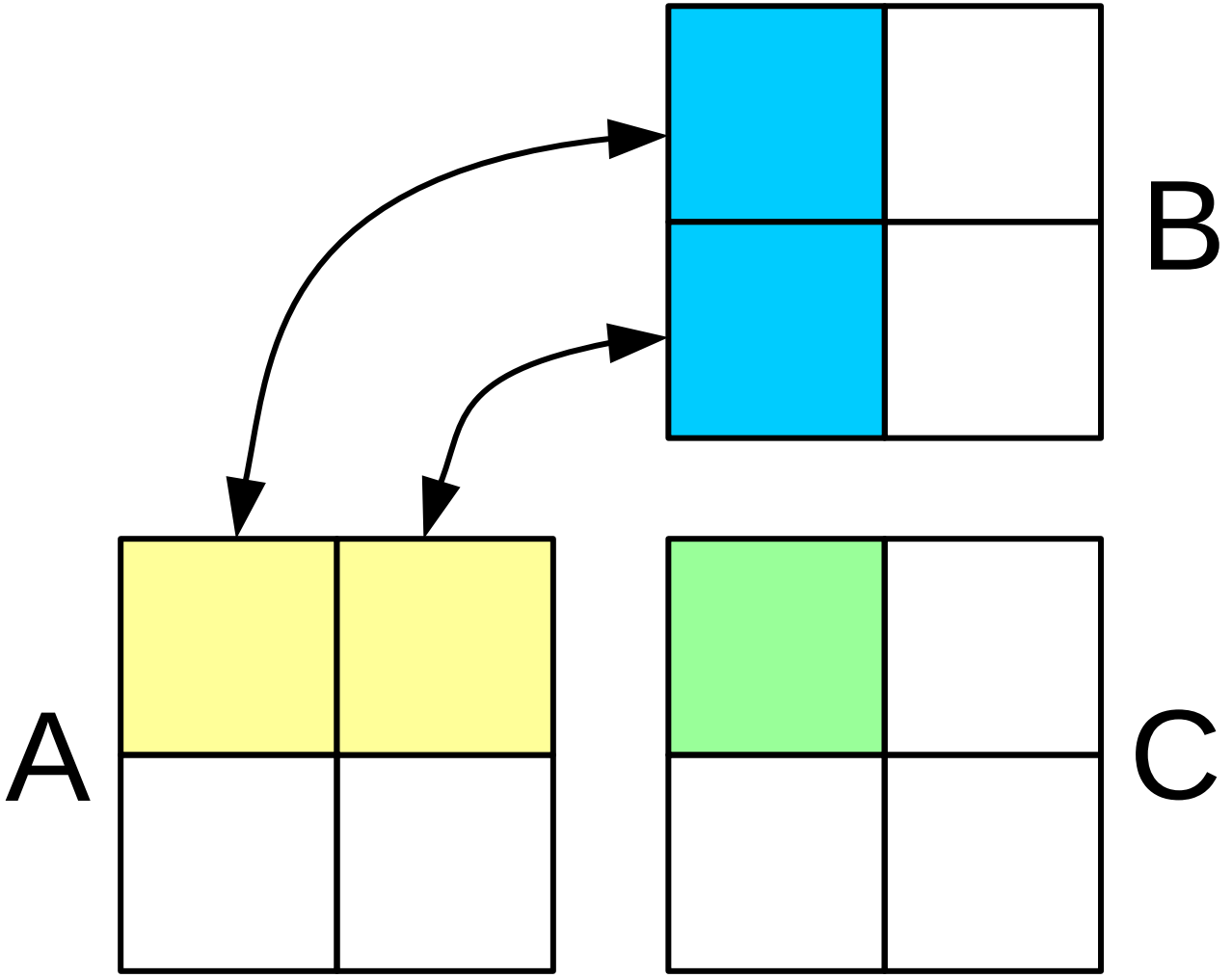
Problemi sa SoA

- Ne beznačajno, skraćenica se sudara sa bar jednom postojećom (Service-oriented architecture)
- Programski jezici su projektovani da koriste AoS, SoA je teško automatizovati i podrška postoji samo u eksperimentalnim jezicima
- Iz tog razloga, sve mora da se piše ručno, što predstavlja dodatno kognitivno opterećenje

Primer: množenje matrica

- Naročito: velikih matrica
- Široko rasprostranjeno u raznim granama primenjene matematike, fundamentalno u linearnoj algebri
- Kod velikog broja programskih jezika, elementi matrice se smeštaju sekvencijalno po vrstama
- Ovo je suboptimalno kod različitih operacija, pa i množenja

Postupak množenja (matematički)



Postupak množenja (programski)

- Najprostije: preslikava se matematička definicija
  - redom po vrstama, pa:
  - za svaku kolonu:
  - nađi odgovarajući element iz druge matrice, pomnoži i uvrsti u zbir
- Ovo znači da se elementima druge matrice pristupa na preskok, što dovodi do usporenja zbog neefikasnosti

Postupak množenja (optimizovan)

- Svaka kolona druge matrice se kopira u sekvencijalni vektor
- Onda se taj vektor upotrebi za izračunavanje svih vrednosti kolone rezultujuće matrice
- Ovo znači da se elementima i prve matrice i privremenog vektora pristupa sekvencijalno: optimalno za performanse
- U merenjima ovaj metod je bar 30% brži za matrice srednje veličine

Vektorske instrukcije: istorija

- Rano u razvoju računarskih sistema uočeno je da kod određenih klasa problema jedna instrukcija može da se primeni na veći skup podataka
- Modelovanje, linearna algebra
- Prva realizacija: vektorski procesori i specijalizovani računari
- Cray-1, -2, X-MP, Y-MP
- Sa razvojem mikroprocesora isplativost specijalizovanih mašina je prestala

Vektorska podrška: x86, početak

- MMX, 1997, za 32-bitnu arhitekturu (tada jedino postojeću)
- 8 64-bitnih registara (mm0–mm7), isključivo celobrojne operacije, registri su preslikani na registre matematičkog koprocesora
- Ovo poslednje je predstavljalo poseban problem zbog nemogućnosti da se koriste MMX i FP operacije istovremeno

Vektorska podrška: x86, SSE

- = „Streaming SIMD Extensions“
- 1999, Pentium III, i dalje 32-bitni procesor
- Operacije u pokretnom zarezu u jednostrukoj preciznosti (32 bita)
- 8 128-bitnih registara (xmm0–xmm7)

Vektorska podrška: x86_64, SSE2

- AMD-ova revizija arhitekture za 64-bitni rad
- Dodatni registri, xmm8–xmm15
- Dodatne celobrojne instrukcije, tako da MMX više nema naročitog smisla osim zbog kompatibilnosti
- SSE2 je minimum koji se može očekivati od bilo kog x86_64 procesora

Dalji razvoj vektorske podrške za x86_64

- SSE3, SSE4, AVX, AVX2, AVX-512
- Još registara, još širi registri: prvo ymm0–ymm15 (256 bita), pa zmm0-zmm31 (512 bita)
- Još instrukcija
- Zbog širine registara potrošnja energije postaje primetna kod izvršavanja izvesnih tipova instrukcija, pa procesor počinje da spušta takt ako se istovremeno sa vektorskim izvršavaju i drugi tokovi instrukcija

Problemi sa vektorskim radom

- Kao i kod optimizacije pristupa memoriji, nije lako automatizovati u opštem slučaju
- Tamo gde je moguće, postupak se zove *autovektorizacija*, i svi moderni prevodioci za C, C++ i Rust (između ostalih) imaju podršku
- Sedam generacija podrške (samo za Intelove i kompatibilne procesore) stvara problem distribucije univerzalnih izvršnih programa i detekcije podrške za napredne instrukcije
- Alternativa (RISC-V): univerzalne vektorske instrukcije, koje se mogu prilagoditi širini i dubini vektorskog skupa registara