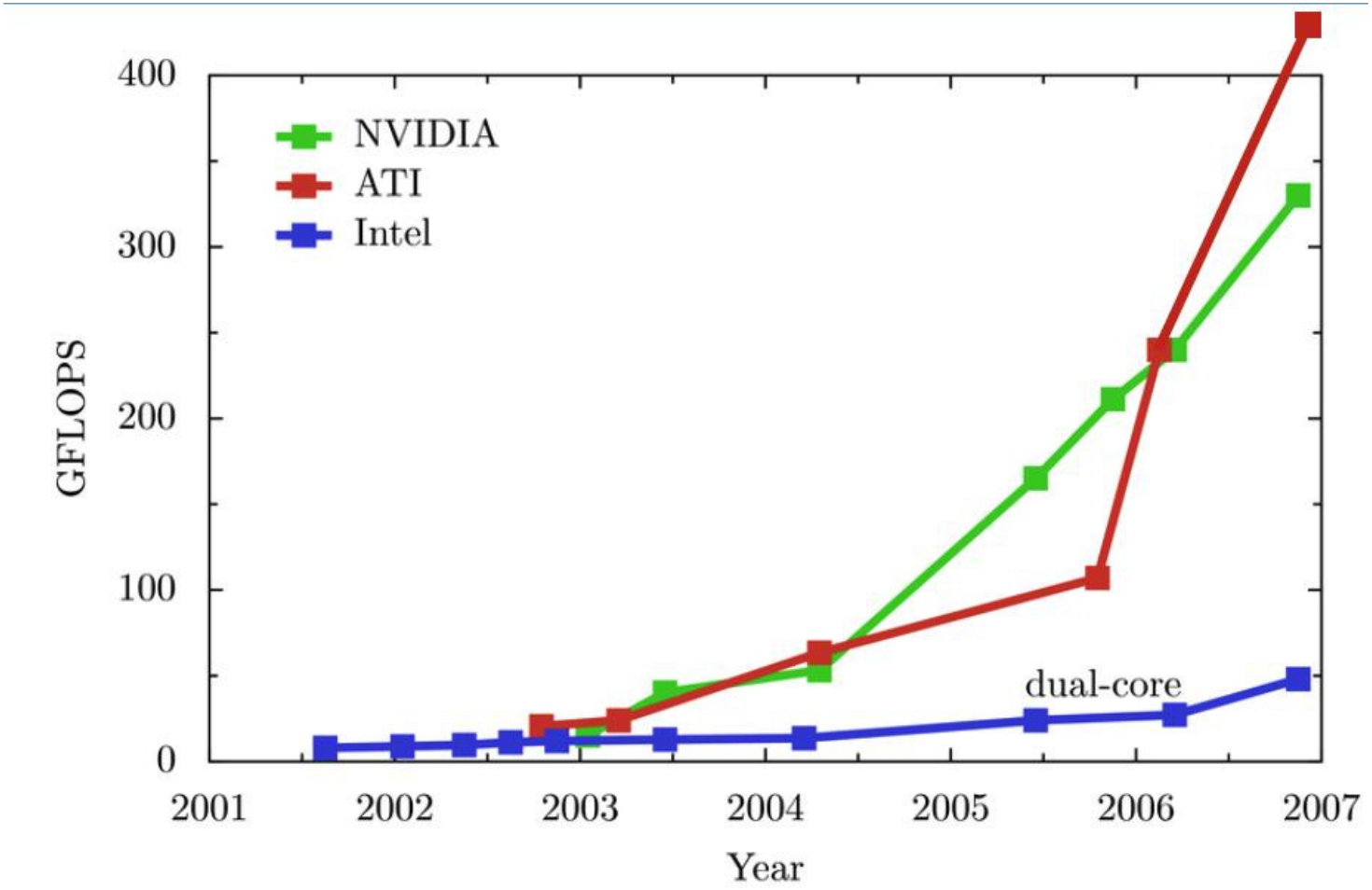
Adaptácia neurónovych sietí pomocou Kalmanovho filtra implementovaného na grafickom procesore

Peter Trebatický

Úvod

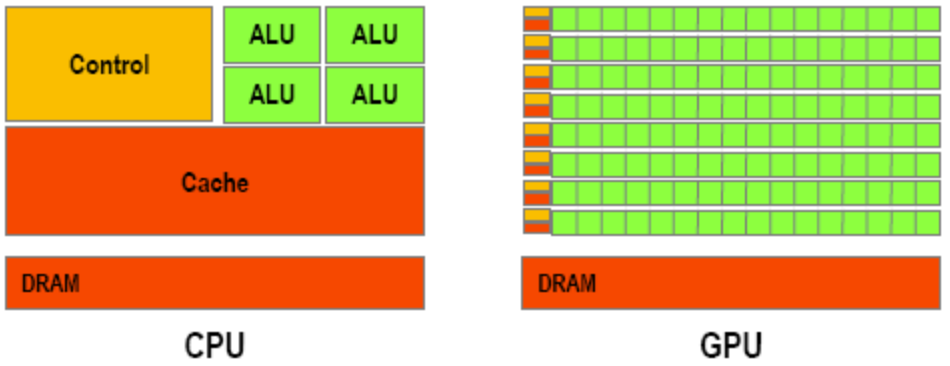
- Grafické procesorové jednotky na bežných grafických kartách sa vyvinuli v extrémne flexibilné a výkonné procesory
- Pôvodne pre urýchlenie matematických výpočtov pre zobrazovanie grafiky
- Kvôli rastúcej zložitosti ponúkaných algoritmov „natvrdo“ bola pridaná možnosť programovania
- GPGPU – General Purpose Computing on GPU
- NVidia Tesla nemá grafický výstup
- Grafická karta sa stáva koprocesorom

Porovnanie výpočtovej sily



Prečo sú GPU stále rýchlejšie?

- Je to vysoko paralelný výpočtový prostriedok
- Sú špecializované, je jednoduchšie pridať ďalšie tranzistory určené na výpočet a nie na cacheovanie dát a riadenie
- Trh s počítačovými hrami tlačí na vyšší výkon za nižšiu cenu
- Konkurencia podnecuie inováciu



Donedávna zložité

- Grafické karty boli vytvorené pre videohry
  - Neobvyklý programovací model
  - Jednotlivé pojmy späté s počítačovou grafikou
  - Veľmi obmedzené programovacie prostredie
- Problém nesúvisiaci s grafikou bolo treba pretransformovať na prácu s „grafikou“
  - Dáta => textúry
  - Algoritmy => skladanie obrazu (renderovacie prechody)
  - Dá sa zapisovať len na predom zvolené miesto (pixel)

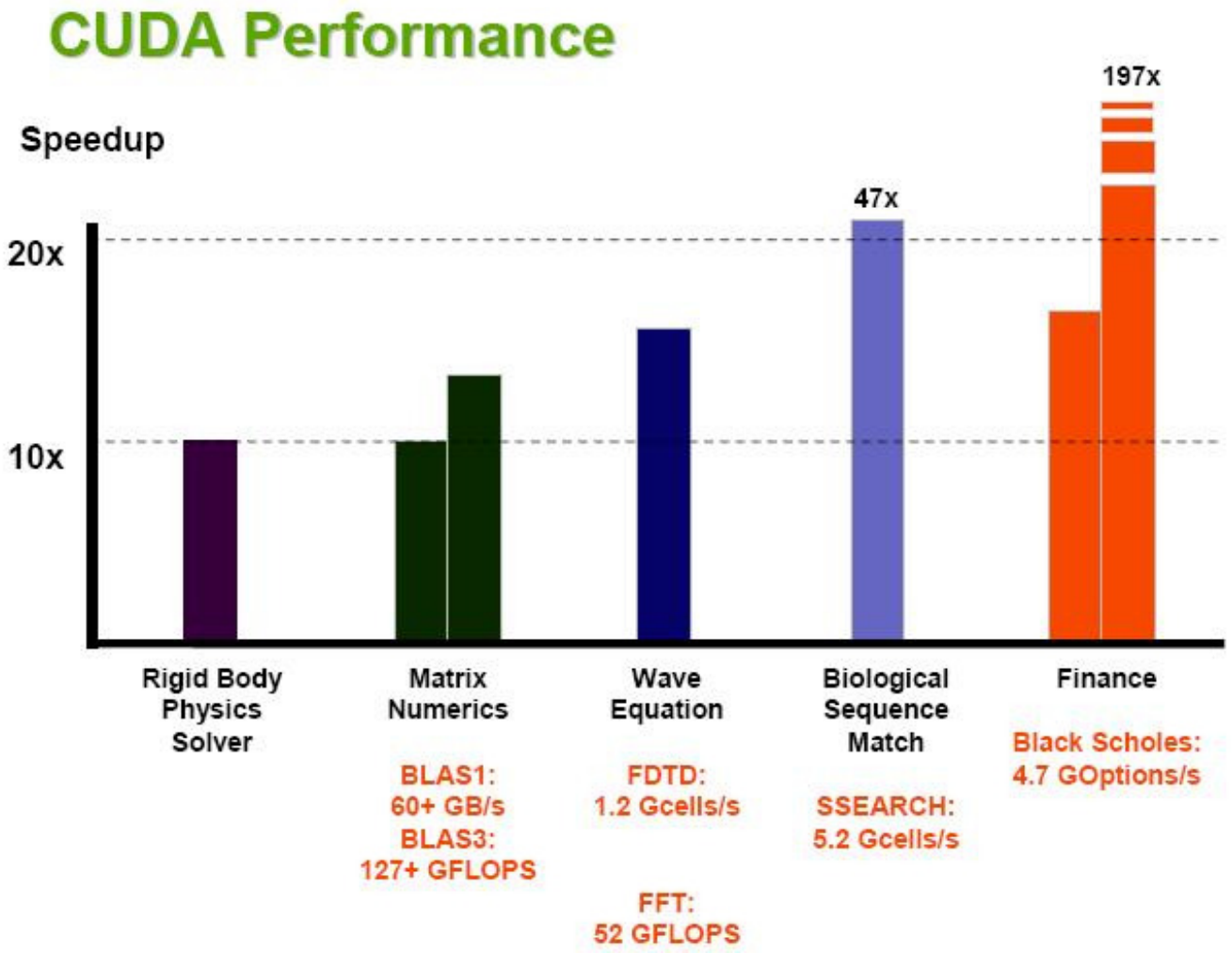
Jazyky

- API
  - Direct3D, OpenGL
- Využitie shaderov
  - HLSL, GLSL, Cg
- GPGPU jazyky
  - Skrývajú detaily D3D/OpenGL
  - CTM, CUDA, Brook, RapidMind

CUDA

- Hardvér aj softvér navrhnutý s ohľadom na možnosť všeobecných výpočtov na GPU
- Hardvér – plne dátovoparalelná architektúra
  - Spúšťanie vlákien (lightweight)
  - Globálne čítanie aj zápis
  - Cache paralelných dát
  - Skalárna architektúra
  - Celé čísla, bitové operácie (doteraz len 32b float)
- Softvér – programuje sa v C
  - Jazyk C s užitočnými rozšíreniami
    - __global__ void KernelFunc(...);
    - __device__ int GlobalVar;
    - __shared__ int SharedVar;
    - KernelFunc<<< 500, 128 >>>(...);

Zrýchlenie výpočtov oproti CPU



Prúdové spracovanie

- Prúdy
  - Skupina údajov vyžadujúcich podobné spracovanie
    - Prvky na mriežke, voxely, atď.
  - Poskytujú dátový paralelizmus
- Kernely
  - Funkcie aplikované na každý prvok v prúde
    - Transformácie, rovnice, ...
  - Malá závislosť medzi elementami
- Programujeme pre stovky procesorov
  - Bežia na nich kernely (spolupracujúce vlákna) spracúvajúce prúdy

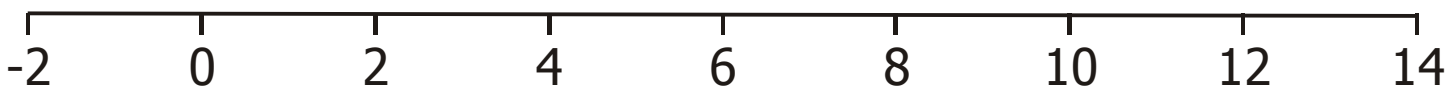
Možné operácie

- Mapovanie
  - map(A,f) aplikuje f na všetky prvky z prúdu A
  - Výsledok je v novom buffery
- Gather (čítanie)
  - p = a[i]
- Scatter
  - a[i] = p
  - zápis nielen na predom zvolené miesto, konflikty sa neriešia
- Redukcia (paralelná)
  - reduce(+,[3, 0, 4, 7])=14
  - +, *, min, max

Kalmanova filtrácia

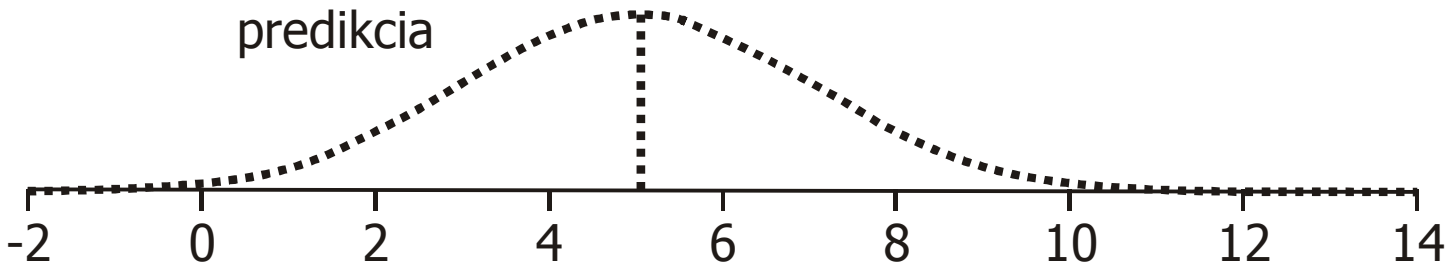
- Alternatíva ku gradientovým metódam trénovania neurónových sietí
- Jednoduchý princíp
- Stáva sa populárnym
- Známy niekoľko desaťročí
- Používaný v rôznych oblastiach

Kalmanov filter

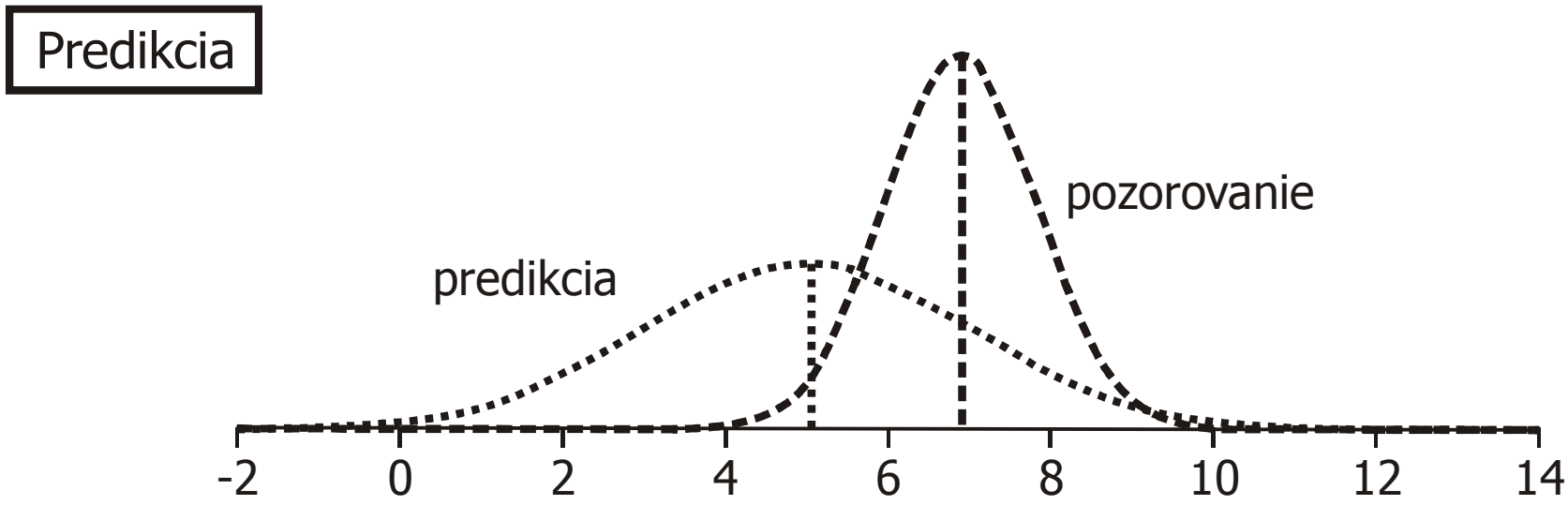


Kalmanov filter

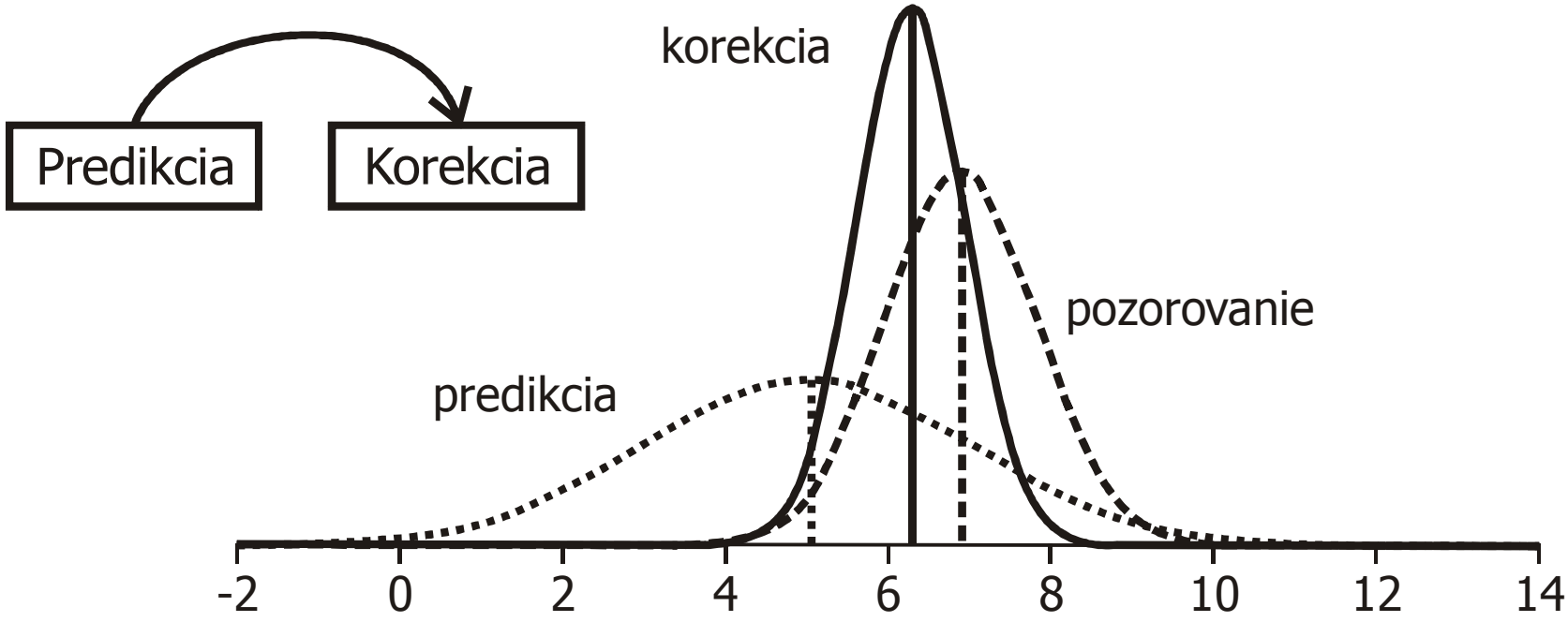
Predikcia



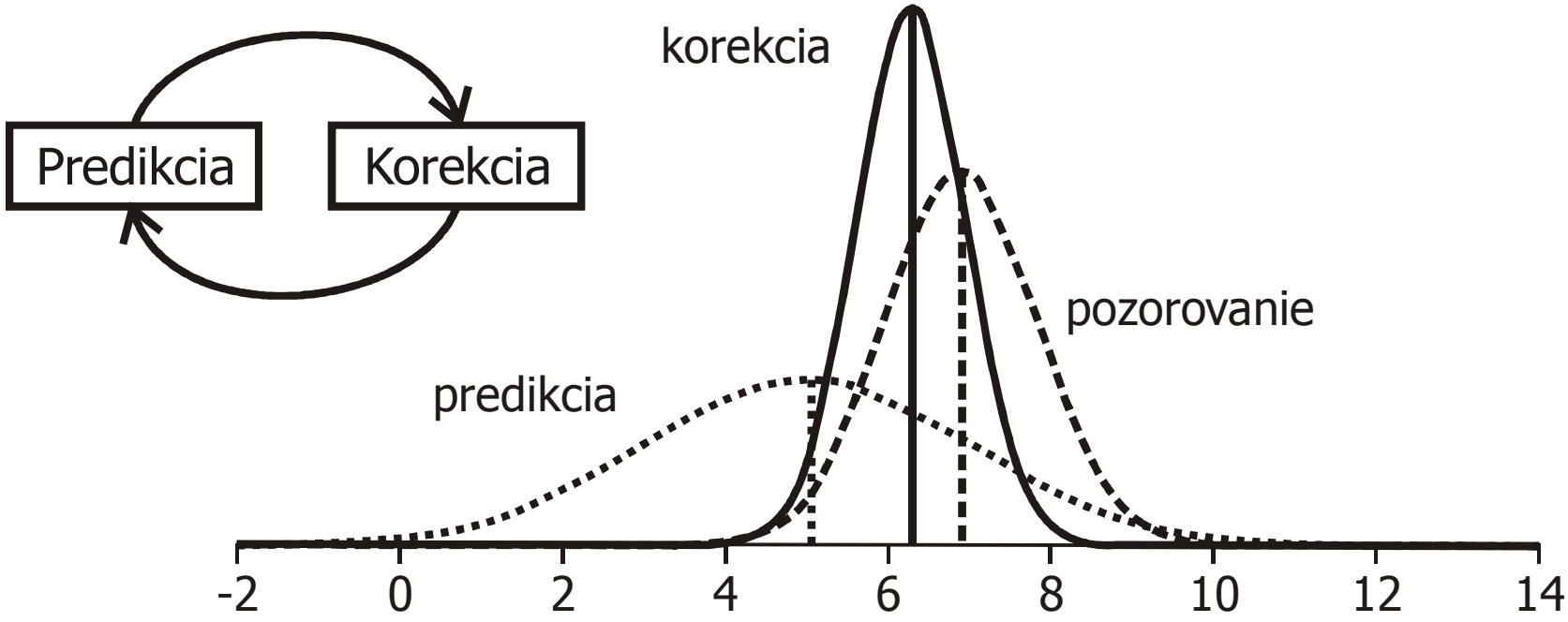
Kalmanov filter



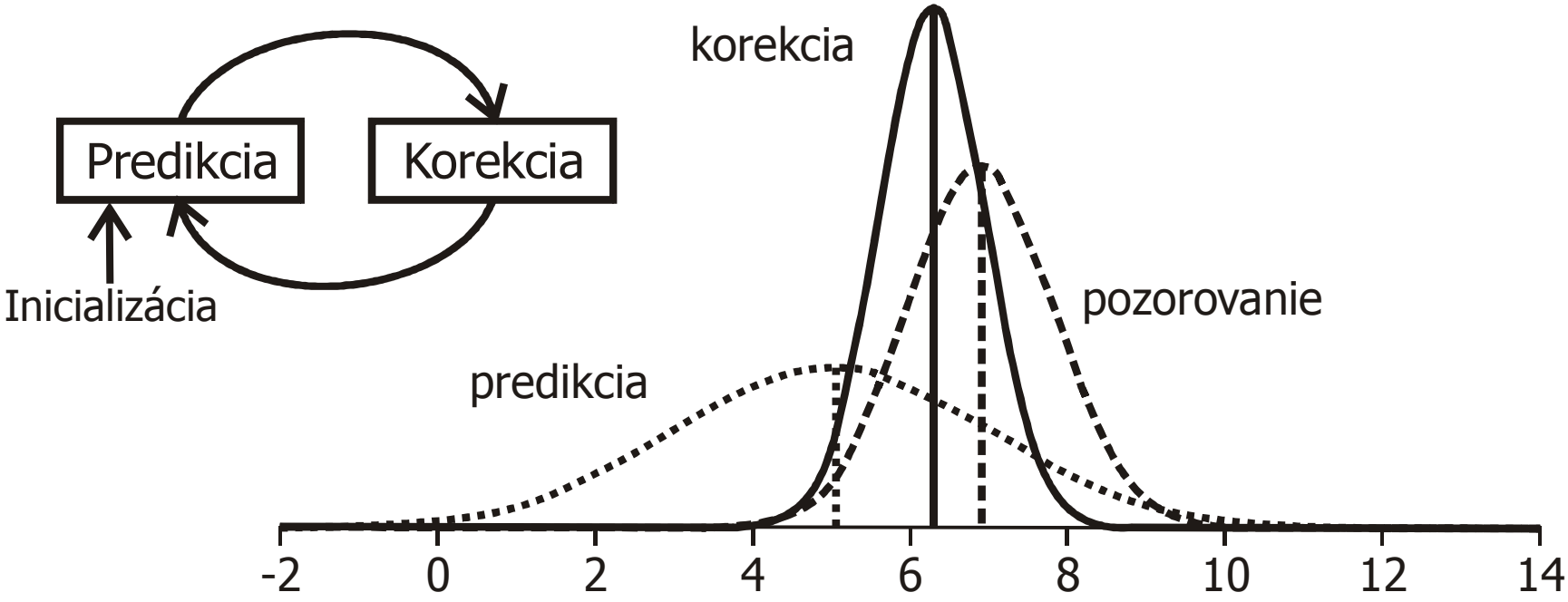
Kalmanov filter



Kalmanov filter



Kalmanov filter



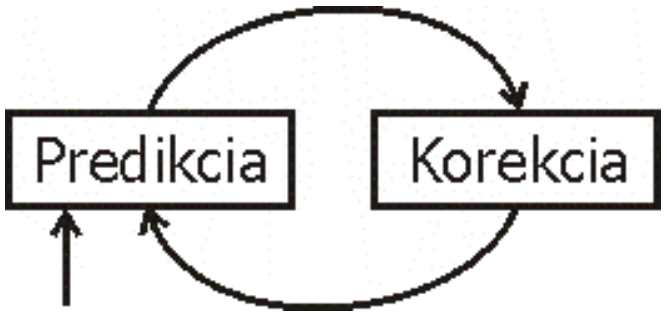
Kalmanov filter

- Sada matematických rovníc
- Pre *lineárne* dynamické systémy

  $x_{k+1} = F_{k+1,k}x_k + q_k$ *stavová rovnica*

  $y_k = H_k x_k + r_k$ *rovnica pozorovania*

  $q_k \sim N(0, Q_k)$; $r_k \sim N(0, R_k)$
- Prediktor-korektor
- Optimálny



Kalmanov filter

- Predikcia

$$\hat{\mathbf{x}}_k^- = \mathbf{F}_{k,k-1}\hat{\mathbf{x}}_{k-1}$$

$$\mathbf{P}_k^- = \mathbf{F}_{k,k-1}\mathbf{P}_{k-1}\mathbf{F}_{k,k-1}^T + \mathbf{Q}_{k-1}$$

- Korekcia

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k\right)^{-1}$$

$$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{H}_k\mathbf{x}_k^-\right)$$

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_k^-$$

Neurónová sieť ako dynamický systém

- Váhy siete predstavujú „stav“ systému

$$x_{k+1} = x_k + q_k$$

$$y_k = h_k(x_k, u_k, v_{k-1}) + r_k$$

- y_k je požadovaný výstup
- Odhad parametrov
- *Nelineárny* systém

Rozšírený Kalmanov filter (EKF)

- Taylorov rozvoj okolo aktuálne odhadovaného stavu
- Prvý člen rozvoja
- Linearizácia
- Môžeme použiť Kalmanov filter

Rozšírený Kalmanov filter (EKF)

- Rovnice:

$$\mathbf{K}_k = \mathbf{P}_k \mathbf{H}_k^T \left[\mathbf{H}_k \mathbf{P}_k \mathbf{H}_k^T + \mathbf{R}_k \right]^{-1}$$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + \mathbf{K}_k \left[\mathbf{y}_k - h\left(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{v}_{k-1} \right) \right]$$

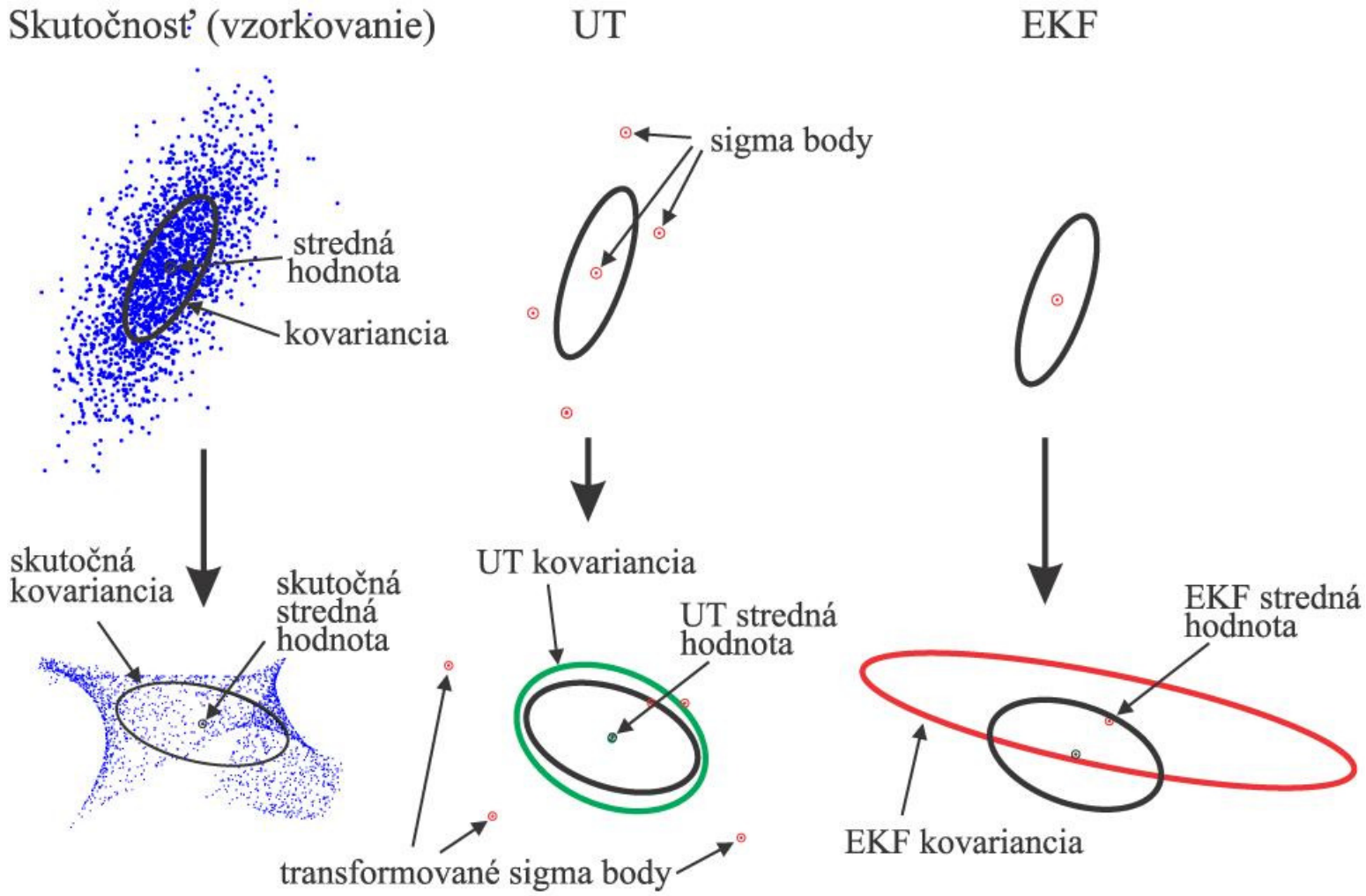
$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_k + \mathbf{Q}_k$$

- Rozmery matíc

$$\mathbf{K}_{n_x \times n_o}, \mathbf{P}_{n_x \times n_x}, \mathbf{Q}_{n_x \times n_x}, \mathbf{R}_{n_o \times n_o}, \mathbf{H}_{n_o \times n_x}$$

- Násobenie matíc
- Invertovanie symetrickej matice

Unscented Kalman Filter (UKF)



Unscented Kalman Filter (UKF)

- Sigma body
  - Malý počet
  - Deterministicky volené
  - Zachytávajú skutočné štatistiky
  - Prejdú nelineárnou transformáciou
- Nie sú potrebné derivácie
- Vyhodnotenia funkcií
- Nutnosť prešíriť vstupy sieťou s novými váhami (sigma bodmi)

Unscented Kalman Filter (UKF)

- Časová aktualizácia

$$\mathbf{x}_{[n_x \times 1]} = \hat{\mathbf{x}}$$

$$\mathbf{P}_{[n_x \times n_x]} = \hat{\mathbf{P}} + \mathbf{Q}$$

- Aktualizácia pozorovaním

$$\boldsymbol{\chi}_{[n_x \times 2n_x+1]} = \left[\mathbf{x} \quad \mathbf{x} + \gamma\sqrt{\mathbf{P}} \quad \mathbf{x} - \gamma\sqrt{\mathbf{P}} \right] \text{ (sigma body)}$$

$$\Upsilon_{[n_o \times 2n_x+1]} = \mathrm{g}(\boldsymbol{\chi}, \mathbf{u})$$

$$\mathbf{y}_{[n_x \times 1]} = \sum_{i=0}^{2n_x} W_i^{(m)} \Upsilon_i$$

$$\mathbf{P}_{yy[n_o \times n_o]} = \sum_{i=0}^{2n_x} W_i^{(c)} \left(\Upsilon_i - \mathbf{y} \right) \left(\Upsilon_i - \mathbf{y} \right)^T + \mathbf{R}$$

$$\mathbf{P}_{xy[n_x \times n_o]} = \sum_{i=0}^{2n_x} W_i^{(c)} \left(\boldsymbol{\chi}_i - \mathbf{x} \right) \left(\Upsilon_i - \mathbf{y} \right)^T$$

$$\mathbf{K}_{[n_x \times n_o]} = \mathbf{P}_{xy} \mathbf{P}_{yy}^T$$

$$\hat{\mathbf{x}} = \mathbf{x} + \mathbf{K} \left(\mathbf{y}_d - \mathbf{y} \right)$$

$$\hat{\mathbf{P}} = \mathbf{P} - \mathbf{K} \mathbf{P}_{yy} \mathbf{K}^T$$

Úzke miesta

- Násobenie matíc
- Invertovanie symetrickej matice
- Choleského dekompozícia

  $\mathbf{B} = \sqrt{\mathbf{A}}, \quad \mathbf{A} = \mathbf{B}\mathbf{B}^T, \ \mathbf{B}$ je dolná trojuholníková matica

  - Použijeme pri výpočte inverznej matice

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

$$\mathbf{B}\mathbf{B}^T\mathbf{A}^{-1} = \mathbf{I}$$

$$\mathbf{B}^T\mathbf{A}^{-1} = \mathbf{Z} \qquad \text{1. Substitúcia, 3. Spätný prechod}$$

$$\mathbf{B}\mathbf{Z} = \mathbf{I} \qquad \text{2. Dopredný prechod}$$

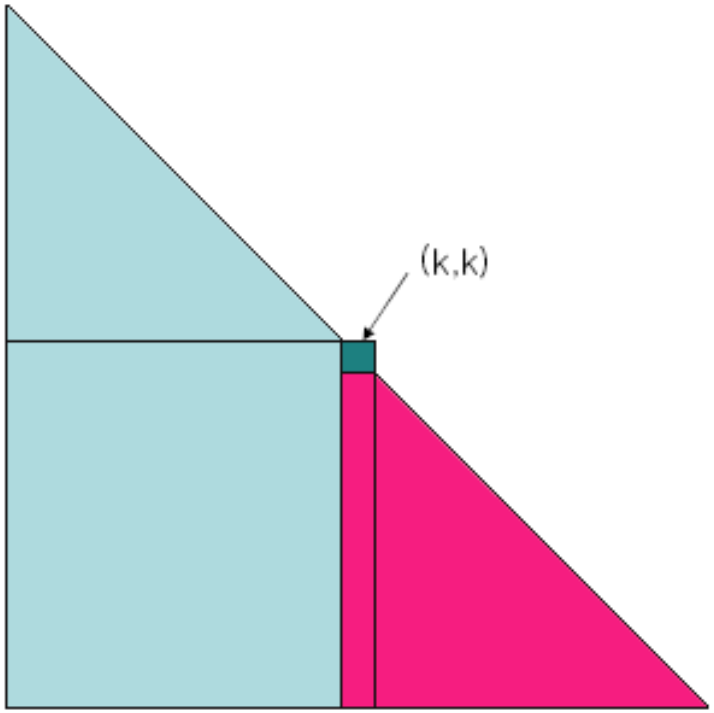
Násobenie matíc na GPU

- So staršou architektúrou problematické
  - Architektúra nebola vhodná
  - Veľa údajov sa znovupoužíva
  - Kombinácia CPU+GPU
- CUDA pre NVidia
  - poskytuje operácie BLAS 1 až 3
  - BLAS3 = násobenie matíc
- Brook+ bude pre ATI
  - Plánuje sa ACML začiatkom 2008

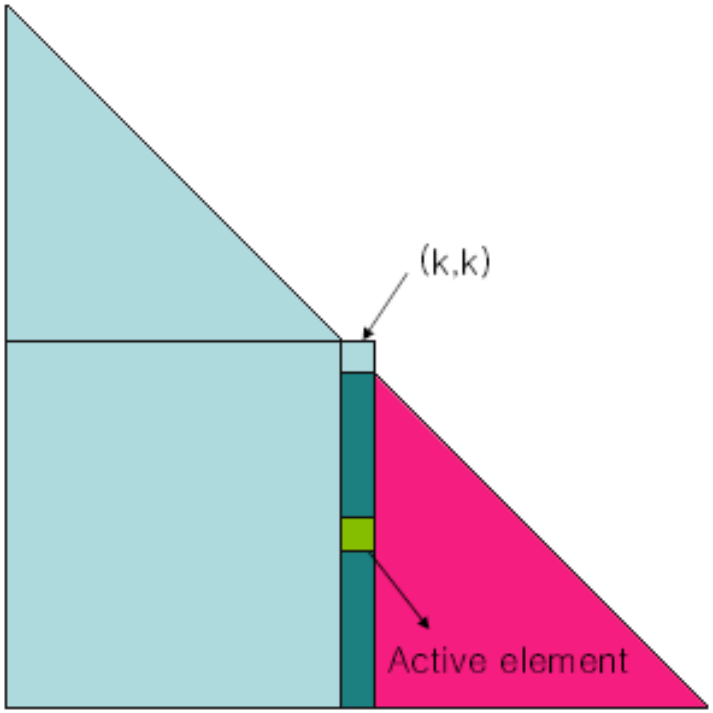
Choleského dekompozícia

for k = 1 to n-1 **do**
  A(k, k) = sqrt(A(k, k))
  A(k+1:n, k) = A(k+1:n, k)/A(k, k)
  for j = k+1 to n **do**
    A(j, k+1) = A(j, k+1) - A(j, 1:k)^T ×A(k+1, 1:k)
  end for
end for
A(n, n) = sqrt(A(n, n))

Choleského dekompozícia

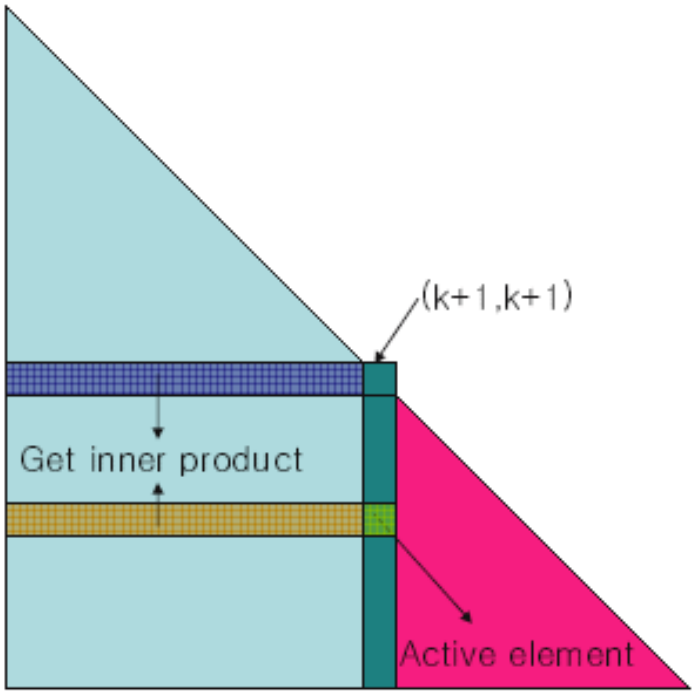


(a) Compute square root of the diagonal element at the kth column and row



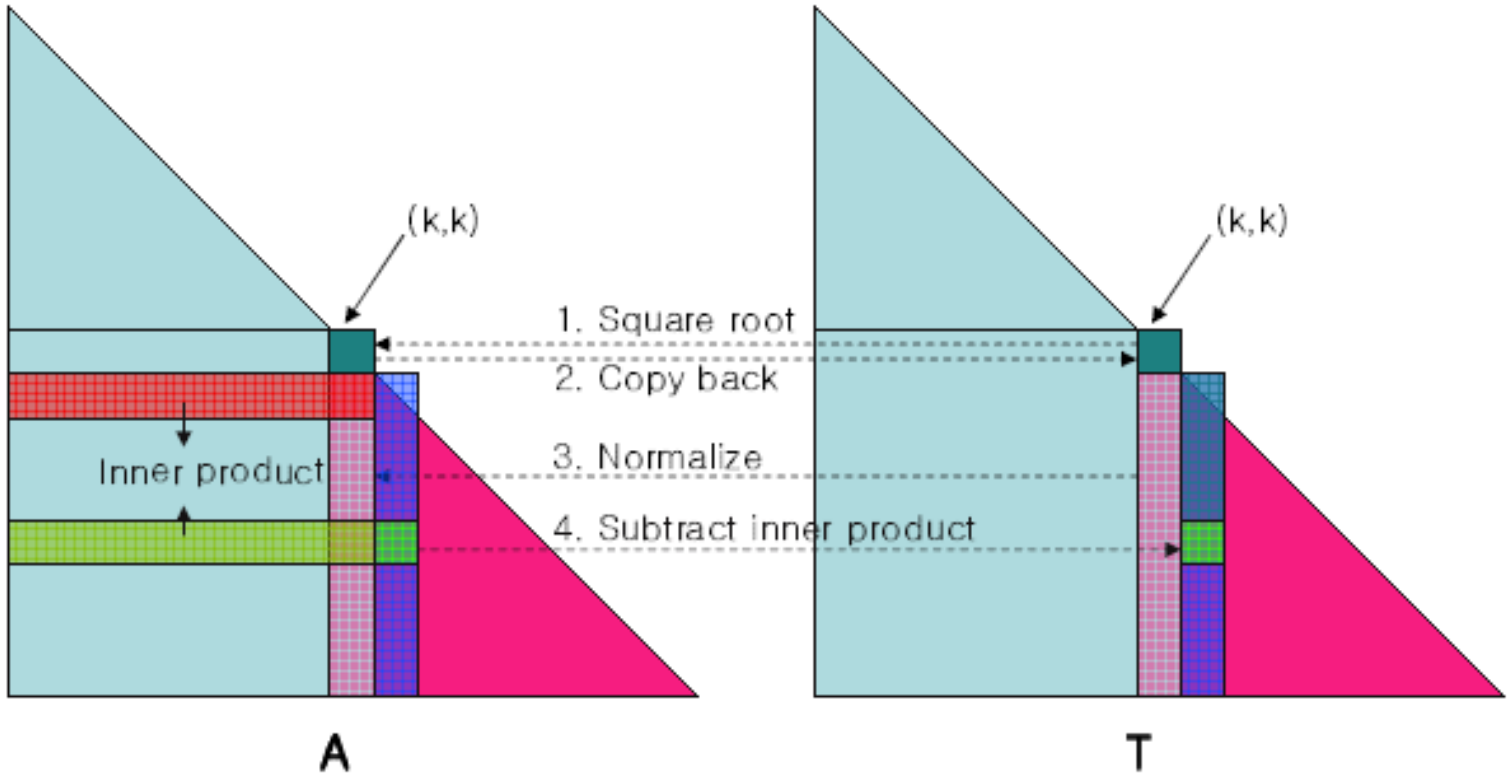
(b) Divide each element in the kth column below the diagonal by the diagonal element

Choleského dekompozícia



(c) **Inner product subtraction** process for inner product updating: Compute the inner product of the two sub-rows, the pivot and the active row, and then subtract the result from the active element

Choleského dekompozícia



(a) Inner product form: Only one additional copy for one element at each iteration is needed to avoid simultaneous reading and writing.

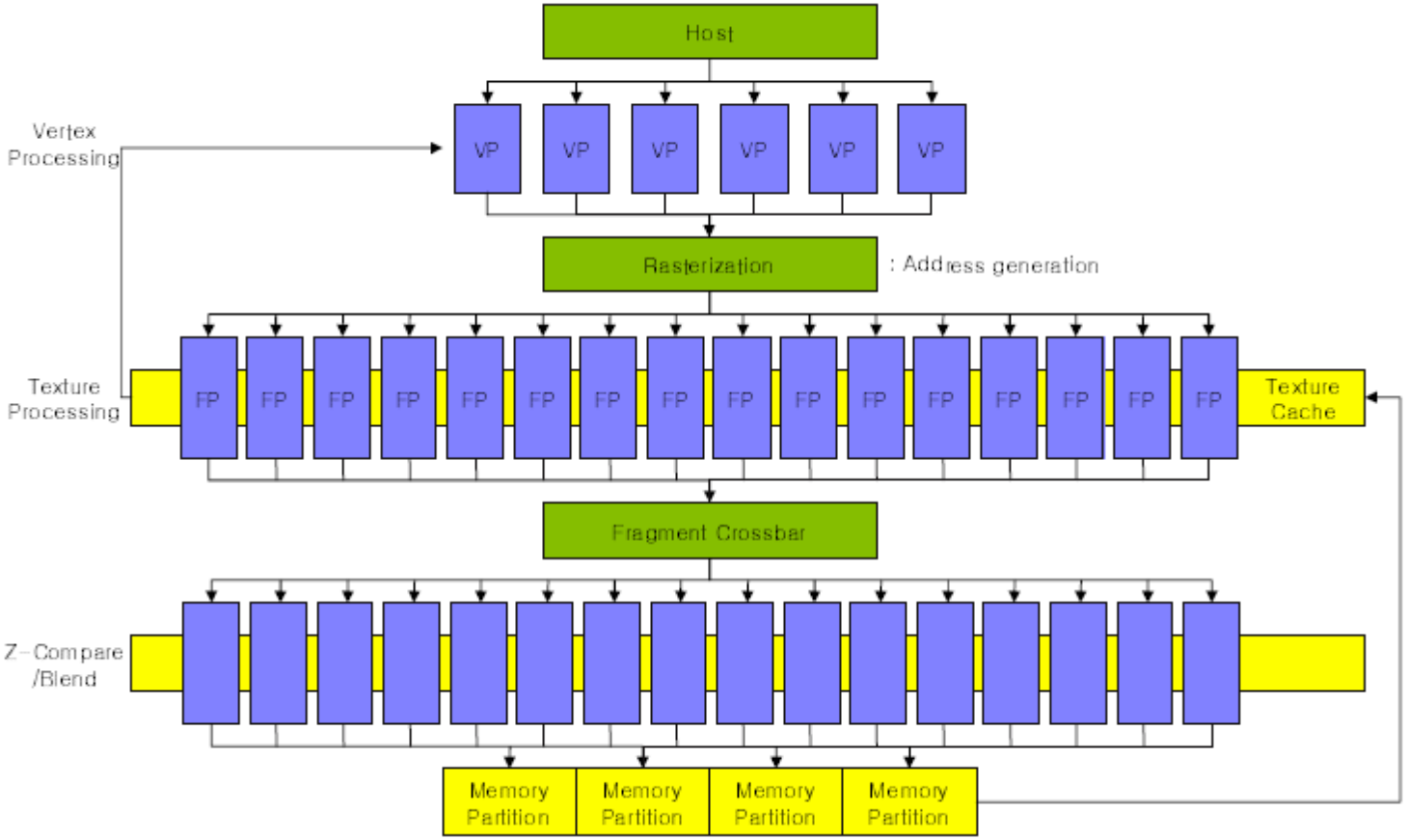
Obmedzenia

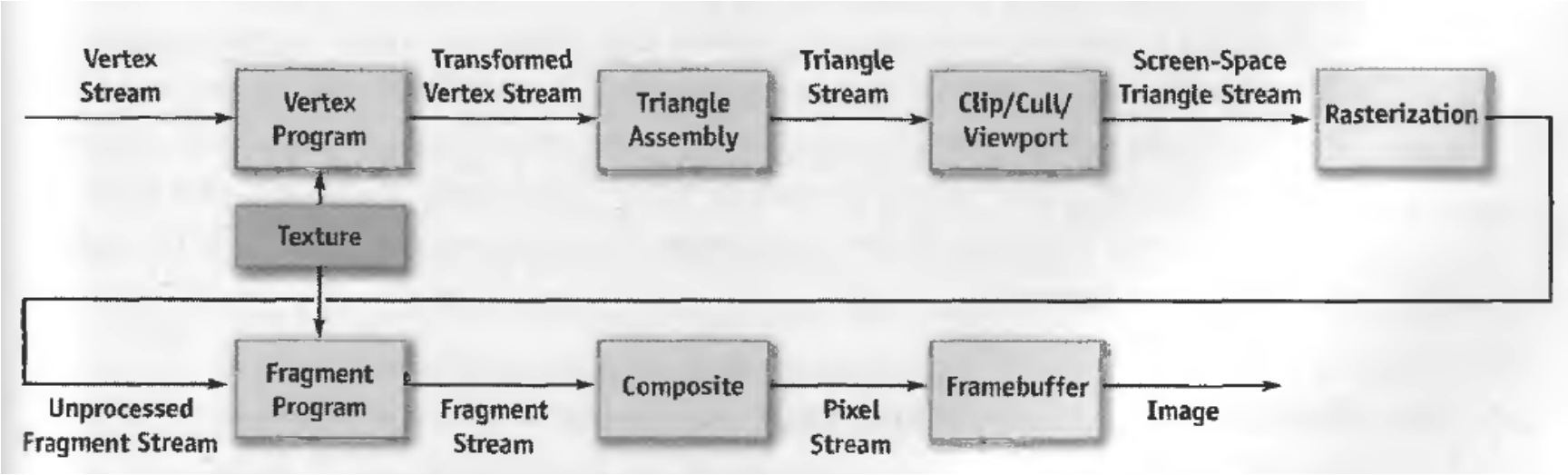
- Násobenie matíc len pre NVidiu
- Zatiaľ len 32 bitové floaty
- Koncom 2007 – začiatok 2008 double
  - Problém s priepustnosťou
- Choleského dekompozícia je numericky stabilná

Ďakujem za pozornosť

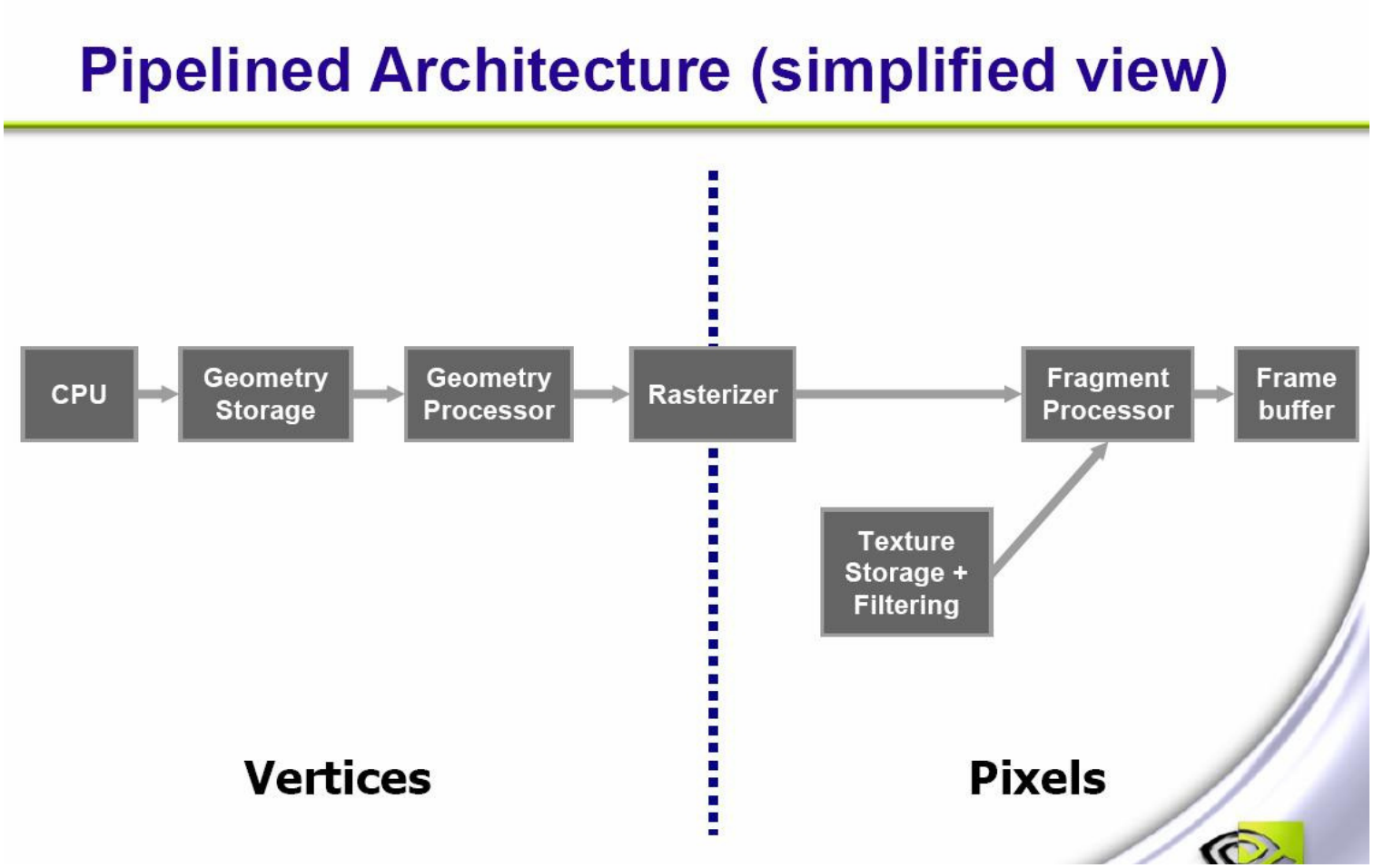
Peter Trebatický
trebaticky@fiit.stuba.sk

Architektúra




Vertex Stream
Vertex Program
Transformed Vertex Stream
Triangle Assembly
Triangle Stream
Clip/Cull/ Viewport
Screen-Space Triangle Stream
Rasterization
Texture
Unprocessed Fragment Stream
Fragment Program
Fragment Stream
Composite
Pixel Stream
Framebuffer
Image

Pipelined Architecture (simplified view)



Obsah

- Moorov zákon CPU vs. GPU
- Architektúra
- Spôsob výpočtu
- Vysvetlenie Kalmana, EKF, UKF
- Čo potrebujeme pre výpočet
  - Násobenie matíc
  - Invertovanie matice
  - Choleského dekompozícia