

PA081: Programování numerických výpočtů

8. Lineární algebra rychle

Aleš Křenek

jaro 2011

Motivace

- ▶ proč má smysl optimalizovat právě algoritmy lineární algebry?
- ▶ frekventované problémy
 - ▶ řešení lineárních rovnic, včetně velkých
 - ▶ řešení nelineárních rovnic
 - ▶ optimalizace
 - ▶ metoda konečných prvků
 - ▶ ...

Motivace

Vyrovňovací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ proč má smysl optimalizovat právě algoritmy lineární algebry?
- ▶ frekventované problémy
 - ▶ řešení lineárních rovnic, včetně velkých
 - ▶ řešení nelineárních rovnic
 - ▶ optimalizace
 - ▶ metoda konečných prvků
 - ▶ ...
- ▶ cíle této přednášky
 - ▶ zdánlivě jednoduché algoritmy (násobení matic) lze výrazně vylepšit
 - ▶ problematika se stále vyvíjí
 - ▶ naznačíme směry, kterými se lze vydat
 - ▶ rafinované algoritmy už jednou někdo vymyslel a implementoval
 - ▶ nemá smysl učit se je nazpaměť
 - ▶ stačí o nich vědět a rozumět základní myšlence

Motivace

Vyrovňovací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ hlavní paměť je řádově pomalejší než procesor
 - ▶ rychlou paměť v plné velikosti je technicky/finančně nemožné vyrobit
- ▶ hierarchie vyrovnávacích pamětí (cache)
- ▶ pro Intel Nehalem/Westmere
 - ▶ L1 - plná rychlost, 32 kB instrukce, 32 kB data/jádro
 - ▶ L2 - latence 10 cyklů, 256 kB/jádro
 - ▶ L3 - latence 40 cyklů, 8 MB/procesor (sdílená mezi jádry)

- ▶ řádky (cache-line)
 - ▶ typicky 64 B (16× float, 8× double)
- ▶ přímo mapovaná cache
 - ▶ blok dat z hlavní paměti má dán jeden řádek v cache

(adresa/velikost řádku) % počet řádků

- ▶ snadno dojde ke kolizím, např. pro 32 kB

```
double pole[100][512][8];  
for (i=0; i<100; i++) a += pole[i][0][0];
```

- ▶ řešením je deklarace `pole[100][513][8]`

Vyrovnávací paměti

Organizace přístupu

- ▶ plně asociativní
 - ▶ blok dat z hlavní paměti může být umístěn v kterémkoli řádku
 - ▶ implementačně náročné, ve standardních CPU se nepoužívá
- ▶ n -cestná asociativní
 - ▶ kompromisní řešení
 - ▶ sady po n řádcích
 - ▶ blok z hlavní paměti může skončit v kterémkoli řádku sady
 - ▶ Intel Nehalem/Westmere: $n = 8$

Vyrovnávací paměti

Praktické zásady

- ▶ optimalizovat pro všechny úrovně
- ▶ využít celý řádek
 - ▶ např. může se vyplatit transponovat matici
- ▶ dovolit procesoru/kompilátoru současně přenášet data a počítat
 - ▶ dobře čitelný kód bez možných vedlejších efektů
- ▶ zabránit předčasným kolizím v jedné sadě řádků
- ▶ měřit dosažený výkon (flop/s)
- ▶ atd. ...

- ▶ optimalizovat pro všechny úrovně
- ▶ využít celý řádek
 - ▶ např. může se vyplatit transponovat matici
- ▶ dovolit procesoru/kompilátoru současně přenášet data a počítat
 - ▶ dobře čitelný kód bez možných vedlejších efektů
- ▶ zabránit předčasným kolizím v jedné sadě řádků
- ▶ měřit dosažený výkon (flop/s)
- ▶ atd. ...
- ▶ aplikovat jen pro skutečně kritické sekce kódu
- ▶ používat speciální optimalizované knihovny, kde to jde

- ▶ zjednodušený model jen s L1
- ▶ násobení vektoru maticí $\mathbf{y} := \mathbf{y} + \mathbf{A}\mathbf{x}$

```
načti x do cache
načti y do cache
for i = 1, ..., n
    načti řádek  $A_{i*}$  do cache
    for j = 1, ..., n
         $y_i = y_i + A_{ij}x_j$ 
zapiš y do hlavní paměti
```

- ▶ počet přístupů do pomalé paměti $m = 3n + n^2$
- ▶ počet aritmetických operací $f = 2n^2$
- ▶ efektivita $f/m \approx 2$
- ▶ algoritmus je limitovaný rychlostí paměti
 - ▶ pomůže recyklace řádků cache - vyplatí se ukládat vektory spojitě, ne jako sloupce matice (v C)
 - ▶ jinak se s tím nedá nic moc dělat

Motivace

Vyrovnávací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ násobení matic $\mathbf{C} = \mathbf{C} + \mathbf{A}\mathbf{B}$

```
for  $i = 1, \dots, n$   
  načti řádek  $A_{i*}$  do cache  
  for  $j = 1, \dots, n$   
    načti  $C_{ij}$  do cache  
    načti sloupec  $B_{*j}$  do cache  
    for  $k = 1, \dots, n$   
       $C_{ij} = C_{ij} + A_{ik}B_{kj}$   
    zapiš  $C_{ij}$  do hlavní paměti
```

- ▶ přístupy do paměti $m = n^2 + n^3 + 2n^2$
- ▶ aritmetické operace $f = 2n^3$
- ▶ efektivita opět ≈ 2

Blokové algoritmy

Maticové násobení

- ▶ matice rozdělíme na N^2 bloků velikosti $b \times b$, $b = n/N$

```
for  $i = 1, \dots, N$ 
  for  $j = 1, \dots, N$ 
    načti blok  $C_{ij}$  do cache
    for  $k = 1, \dots, N$ 
      načti blok  $A_{ik}$  do cache
      načti blok  $B_{kj}$  do cache
       $C_{ij} = C_{ij} + A_{ik}B_{kj}$  (maticové násobení)
    zapiš blok  $C_{ij}$  do cache
```

- ▶ přístupy do paměti
 - ▶ čtení bloků **A**: $N^3(n/N)^2 = N * n^2$
 - ▶ dtto **B**: $N * n^2$
 - ▶ čtení a zápis **C**: $2n^2$
- ▶ efektivita $f/m = \frac{2n^3}{(2N+2)n^2} \approx b$

Motivace

Vyrovnávací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ velikost reálné L1 cache je 32 kB,
- ▶ do L1 se vejdou 3 matice velikosti 36×36 (v `double`)
- ▶ L2 cache je $10\times$ pomalejší
- ▶ procesor je dobře využit
 - ▶ i při využití vektorových instrukcí
 - ▶ proto je cache právě tak velká
- ▶ žádoucí aplikovat rekurzivně i pro L2 a L3
- ▶ implementováno v optimalizovaných knihovnách
- ▶ násobení matic je na současných procesorech jeden z nejefektivnějších algoritmů
 - ▶ redukce problému na mat. násobení při řádovém zachování počtu operací téměř vždy vede k výraznému zrychlení

- ▶ *Basic Linear Algebra Subprograms*,
<http://www.netlib.org/blas>
- ▶ základní operace, např.
 - ▶ DOT: $\mathbf{x} \cdot \mathbf{y}$
 - ▶ AXPY: $\mathbf{y} + \mathbf{A}\mathbf{x}$
 - ▶ NRM2: $|\mathbf{x}|^2$
 - ▶ TRMV: $\mathbf{A}\mathbf{x}$
 - ▶ TRSV: $\mathbf{A}^{-1}\mathbf{x}$
 - ▶ GEMM: $\beta\mathbf{C} + \alpha\mathbf{A}\mathbf{B}$
- ▶ verze pro typy `float`, `double`, `complex`
- ▶ specializované varianty pro symetrické, trojúhelníkové, a některé řídké matice
- ▶ de-facto standardizované rozhraní
- ▶ implementace vyladěné pro konkrétní typy CPU
- ▶ využíváné v knihovnách vyšší úrovně (např. LU dekompozice)

Motivace

Vyrovnávací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

Asymptotická složitost

- ▶ časová složitost násobení matic je $O(n^3)$

Motivace

Vyrovňovací
paměti

Blokové
algoritmy

BLAS

**Asymptotická
složitost**

LU
dekompozice

Vektorové
instrukce

Asymptotická složitost

- ▶ časová složitost násobení matic je $O(n^3)$
- ▶ Strassen (1969): $O(n^{2.81})$
- ▶ rozdělení matic na 2×2 bloky, potom

$$\mathbf{M}_1 = (\mathbf{A}_{11} + \mathbf{A}_{22})(\mathbf{B}_{11} + \mathbf{B}_{22})$$

$$\mathbf{M}_2 = (\mathbf{A}_{21} + \mathbf{A}_{22})\mathbf{B}_{11}$$

$$\mathbf{M}_3 = \mathbf{A}_{11}(\mathbf{B}_{12} + \mathbf{B}_{22})$$

$$\mathbf{M}_4 = \mathbf{A}_{22}(\mathbf{B}_{21} - \mathbf{B}_{11})$$

$$\mathbf{M}_5 = (\mathbf{A}_{11} + \mathbf{A}_{12})\mathbf{B}_{22}$$

$$\mathbf{M}_6 = (\mathbf{A}_{21} - \mathbf{A}_{11})(\mathbf{B}_{11} + \mathbf{B}_{12})$$

$$\mathbf{M}_7 = (\mathbf{A}_{12} - \mathbf{A}_{22})(\mathbf{B}_{21} + \mathbf{B}_{22})$$

$$\mathbf{C}_{11} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$

$$\mathbf{C}_{12} = \mathbf{M}_3 + \mathbf{M}_5$$

$$\mathbf{C}_{21} = \mathbf{M}_2 + \mathbf{M}_4$$

$$\mathbf{C}_{22} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

- ▶ rekurzivní aplikace, počet operací $T(n)$

$$T(n) = 7T(n/2) + 18(n/2)^2 = O(n^{\log_2 7}) = O(n^{2.81})$$

Motivace

Vyrovnávací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

Asymptotická složitost

- ▶ časová složitost násobení matic je $O(n^3)$
- ▶ Strassen (1969): $O(n^{2.81})$
- ▶ rozdělení matic na 2×2 bloky, potom

$$\mathbf{M}_1 = (\mathbf{A}_{11} + \mathbf{A}_{22})(\mathbf{B}_{11} + \mathbf{B}_{22})$$

$$\mathbf{M}_2 = (\mathbf{A}_{21} + \mathbf{A}_{22})\mathbf{B}_{11}$$

$$\mathbf{M}_3 = \mathbf{A}_{11}(\mathbf{B}_{12} + \mathbf{B}_{22})$$

$$\mathbf{M}_4 = \mathbf{A}_{22}(\mathbf{B}_{21} - \mathbf{B}_{11})$$

$$\mathbf{M}_5 = (\mathbf{A}_{11} + \mathbf{A}_{12})\mathbf{B}_{22}$$

$$\mathbf{M}_6 = (\mathbf{A}_{21} - \mathbf{A}_{11})(\mathbf{B}_{11} + \mathbf{B}_{12})$$

$$\mathbf{M}_7 = (\mathbf{A}_{12} - \mathbf{A}_{22})(\mathbf{B}_{21} + \mathbf{B}_{22})$$

$$\mathbf{C}_{11} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$

$$\mathbf{C}_{12} = \mathbf{M}_3 + \mathbf{M}_5$$

$$\mathbf{C}_{21} = \mathbf{M}_2 + \mathbf{M}_4$$

$$\mathbf{C}_{22} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

- ▶ rekurzivní aplikace, počet operací $T(n)$

$$T(n) = 7T(n/2) + 18(n/2)^2 = O(n^{\log_2 7}) = O(n^{2.81})$$

- ▶ Coppersmith-Winograd (1987): $O(n^{2.376})$

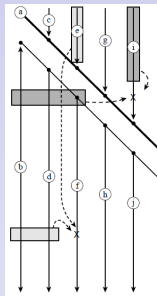
LU dekompozice

- ▶ rekurzivní formulace algoritmu

$$\left(\begin{array}{c|cc} \mathbf{A}_{00} & \mathbf{a}_{01} & \mathbf{A}_{02} \\ \hline \mathbf{a}_{10}^T & \alpha_{11} & \mathbf{a}_{12}^T \\ \mathbf{A}_{20} & \mathbf{a}_{21} & \mathbf{A}_{22} \end{array} \right)$$

- ▶ skalární a bloková verze

$$\begin{aligned} \mathbf{a}_{21} &= \mathbf{a}_{21} / \alpha_{11} & \begin{pmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{pmatrix} &= LU \begin{pmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{pmatrix} \\ \mathbf{a}_{12}^T &\text{ zůstává} & \mathbf{A}_{12} &= \mathbf{L}_{11}^{-1} \mathbf{A}_{12} \\ \mathbf{A}_{22} &= \mathbf{A}_{22} - \mathbf{a}_{21} \mathbf{a}_{12} & \mathbf{A}_{22} &= \mathbf{A}_{22} - \mathbf{L}_{21} \mathbf{A}_{12} \end{aligned}$$



Motivace

Vyrovňovací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ podstatná část operací je násobení matic
- ▶ takto implementuje knihovna LAPACK

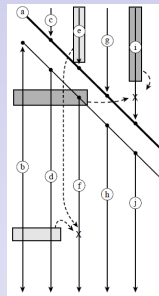
LU dekompozice

- ▶ rekurzivní formulace algoritmu

$$\left(\begin{array}{c|cc} \mathbf{A}_{00} & \mathbf{a}_{01} & \mathbf{A}_{02} \\ \hline \mathbf{a}_{10}^T & \alpha_{11} & \mathbf{a}_{12}^T \\ \mathbf{A}_{20} & \mathbf{a}_{21} & \mathbf{A}_{22} \end{array} \right)$$

- ▶ skalární a bloková verze

$$\begin{aligned} \mathbf{a}_{21} &= \mathbf{a}_{21} / \alpha_{11} & \begin{pmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{pmatrix} &= LU \begin{pmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{pmatrix} \\ \mathbf{a}_{12}^T &\text{ zůstává} & \mathbf{A}_{12} &= \mathbf{L}_{11}^{-1} \mathbf{A}_{12} \\ \mathbf{A}_{22} &= \mathbf{A}_{22} - \mathbf{a}_{21} \mathbf{a}_{12} & \mathbf{A}_{22} &= \mathbf{A}_{22} - \mathbf{L}_{21} \mathbf{A}_{12} \end{aligned}$$



Motivace

Vyrovňovací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ podstatná část operací je násobení matic
- ▶ takto implementuje knihovna LAPACK
- ▶ problém algoritmu - „dlouhé“ matice \mathbf{A}_{21} a \mathbf{A}_{12}
- ▶ Quintana et. al (2008): algoritmus s bloky $2p \times p$

- ▶ paralelismus na úrovni BLAS
 - ▶ existují optimalizované implementace
 - ▶ implicitní synchronizace na konci každého volání
 - ▶ nemusí být dosažitelný optimální výkon
- ▶ blokové operace jsou efektivní provedené vcelku
 - ▶ všechna data se dostanou naráz do cache
 - ▶ na úrovni L1/L2 se vyplatí na jednom jádru CPU
- ▶ vzájemně nezávislé blokové operace na více jádrech

- ▶ Quintana et. al (2008) – prototypová implementace SuperMatrix
 - ▶ konkrétní algoritmus proběhne „abstraktně“
 - ▶ blokové operace s deklaroványými závislostmi se pouze zařadí do fronty
 - ▶ následně se vyhodnotí pořadí zpracování
 - ▶ operace se provedou potenciálně paralelně
- ▶ čitelná formulace algoritmu bez explicitního paralelismu
- ▶ efektivní provedení
 - ▶ matice $n > 5000$
 - ▶ 16 jader CPU
 - ▶ LU dekompozice na více než 50% teoretického výkonu

- ▶ historie – např. Cray v předsálí budovy D
- ▶ současné vektorové systémy (např. NEC SX)
 - ▶ řešení velmi specializovaných problémů
 - ▶ nízký výkon na skalárních operacích
- ▶ vektorová rozšíření v architektuře x86
 - ▶ MMX: pouze celá čísla, kolize s float registry
 - ▶ SSE: 8 (16) registrů po 128 bitech, postupně přidávané instrukce (rsqrt)
 - ▶ AVX: 256 bitové registry, 3-operandové instrukce
- ▶ snažší využití plného výkonu procesoru

Motivace

Vyrovňovací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ triviální příklad

```
float a[4],b[4];
int    i;
for (i=0; i<4; i++) a[i] += b[i];
...
movaps    32(%rsp), %xmm0
addps     (%rsp), %xmm0
movaps    %xmm0, 32(%rsp)
```

- ▶ vektorizaci kódu provede chytřejší kompilátor (icc)
 - ▶ musíme hlídat, abychom tomu nebránili
 - ▶ recyklace proměnných, vedlejší efekty in-line funkcí, ...
 - ▶ vyplatí se kontrola vygenerovaného assembleru

Motivace

Vyrovnávací
paměti

Blokové
algoritmy

BLAS

Asymptotická
složitost

LU
dekompozice

Vektorové
instrukce

- ▶ téměř vše, co platí o cache, platí také o GPU
 - ▶ rychlá paměť omezené velikosti
 - ▶ netriviální režie kopírování z/do hlavní paměti
- ▶ paralelní kód
 - ▶ daleko masivnější (desítky až stovky)
- ▶ viz PV197: GPU Programming