KIV/ZEP - 2011

Cache

Když se řekne „cache“

- Cache = vyrovnávací paměť jejíž úkolem je zvýšení výkonu aplikace při opakovaném přístupu k datům
  - typicky dělena na položky, tj. nejmenší logická jednotka, kterou lze uložit
    - může být to jedna instance třídy nebo blok (tzv. cache line - řádka) o několika Bytech – typicky 32 nebo 64
- při přístupu k datům se nejprve zkontroluje, zda existuje kopie dat v cache, pokud ano (cache hit), jsou poskytnuta z cache, pokud ne (cache miss) jsou načtena ze svého původního, mnohem pomalejšího, zdroje

Když se řekne „cache“

- cache je typicky mnohem menší než původní úložiště
- přístup k datům s využitím a bez využití cache je z programového hlediska transparentní
  - logika je zapouzdřena v nějaké komponentě
    - např. volám metodu nějaké třídy, nevím, co se děje uvnitř („černá krabička“)

Co může být „cache“

- hardwarová cache
  - CPU: L1, L2 a L3
  - pevný disk
  - paměť grafické karty
- softwarová cache
  - ukládání vzdáleného obsahu na disk nebo do paměti prohlížečemi

- algoritmus provádí CPU
- nechť algoritmus chce pracovat s daty uloženými na nějakém serveru

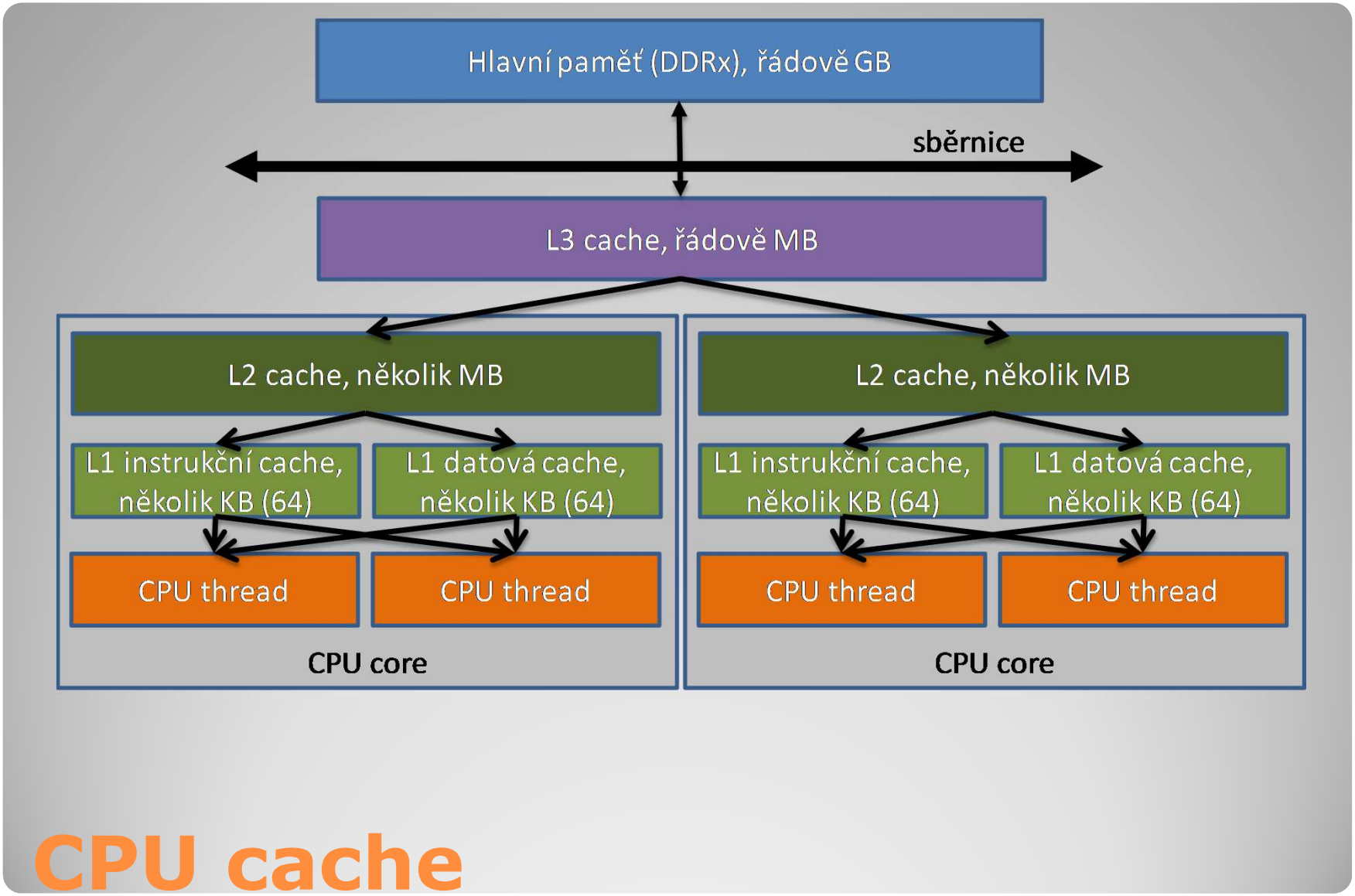
typ zařízení	maximální teoretická přenosová rychlost
síť InfiniBand	96 GBit/s
„obvyklá“ síť	1 GBit/s
pevný disk SATA 3.0 (2009)	6 GBit/s
pevný disk SATA 2.0 (běžnější)	3 GBit/s
paměť DDR3-1600 na 200 MHz	102.4 GBit/s
2 GHz CPU	64 GBit/s

Význam „cache“ při práci s daty

- má-li program běžet rychle, musí efektivně využívat HW cache (případně SW cache)
  - minimalizace výpadků dat z cache
- nejdůležitější je CPU cache
  - L1, L2 a L3 cache
  - dnes obvykle 64B / cache line

umístění dat	počet CPU cyklů
registry	~1
L1 datová cache	~3
L2 cache	~14
hlavní paměť	~240

Co to vše znamená?


Hlavní paměť (DDRx), řádově GB
sběrnice
L3 cache, řádově MB
L2 cache, několik MB
L2 cache, několik MB
L1 instrukční cache, několik KB (64)
L1 datová cache, několik KB (64)
L1 instrukční cache, několik KB (64)
L1 datová cache, několik KB (64)
CPU thread
CPU thread
CPU thread
CPU thread
CPU core
CPU core

CPU cache

Využití cache

- cache předpokládá, že přistupujete k datům v lineárním pořadí
- základní pravidla:
  - strukturovat data tak, aby data algoritmu ležela v paměti vedle sebe
  - minimalizovat otisk struktury / třídy v paměti
  - restrukturalizace cyklů (např. 2 fory v sobě) tak, aby se zpravovávaly položky ležící v paměti vedle sebe

Využití cache

- double[] x;
  double[] y;
  double[] z;

- class / struct Point {
  double x, y, z;
  }

  Point[] xyz;

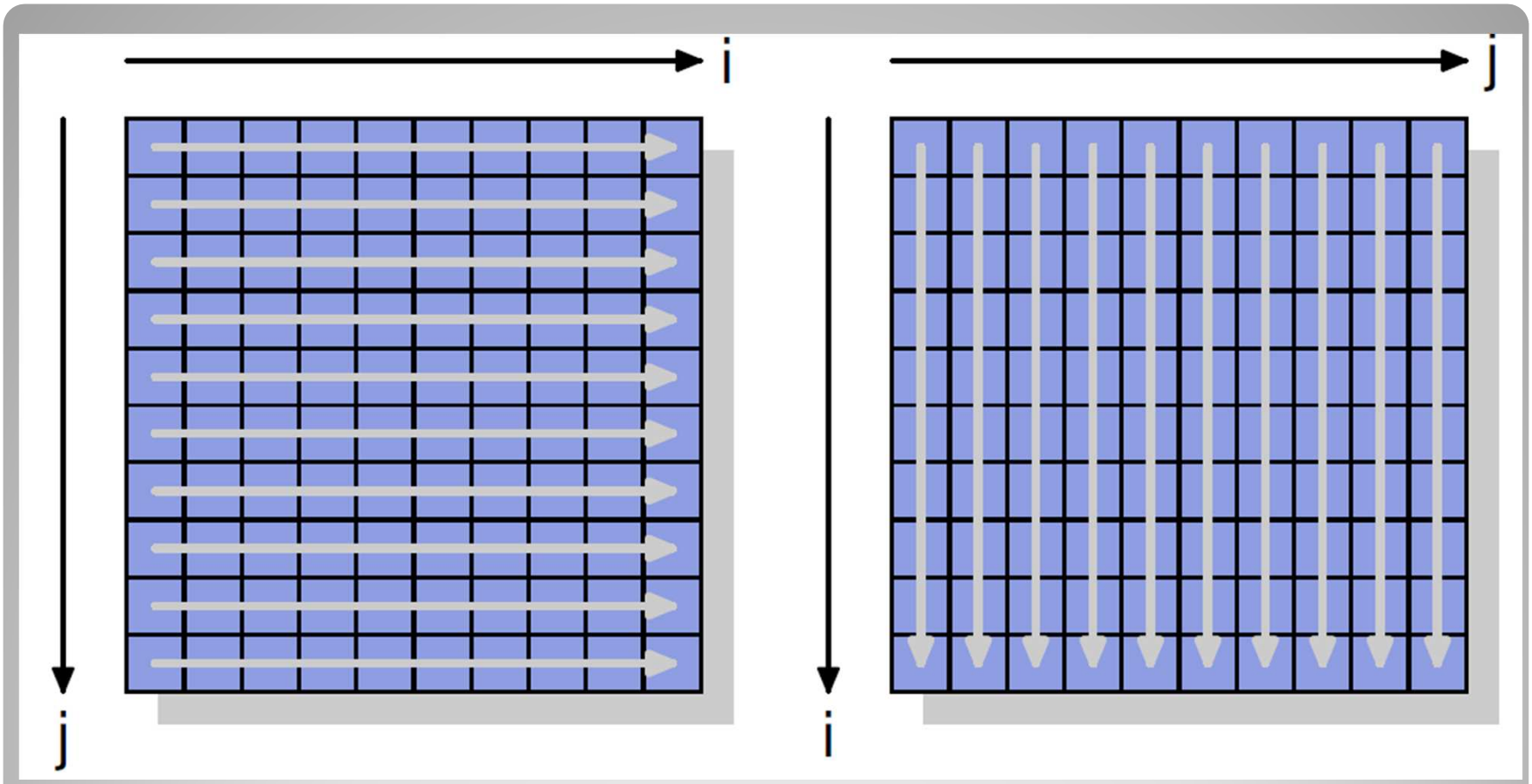
```
class A {
        int a;
        char c;
        int b;
        bool d;
};
```

Využití cache

Využití cache

- násobení matic res = mul1*mul2
  - problém: data uspořádána nevhodně

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    for (k = 0; k < N; ++k)
      res[i][j] += mul1[i][k] * mul2[k][j];
```



Využití cache

Využití cache

- možné řešení: napřed transponovat
  - 77% urychlení na Intel Core 2, 2.6GHz!
  - funguje jen, známe-li organizaci paměti
  - vyžaduje dodatečnou paměť

```
double tmp[N][N];
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    tmp[i][j] = mul2[j][i];
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    for (k = 0; k < N; ++k)
      res[i][j] += mul1[i][k] * tmp[j][k];
```

- *příklad: bricking vs. normal*

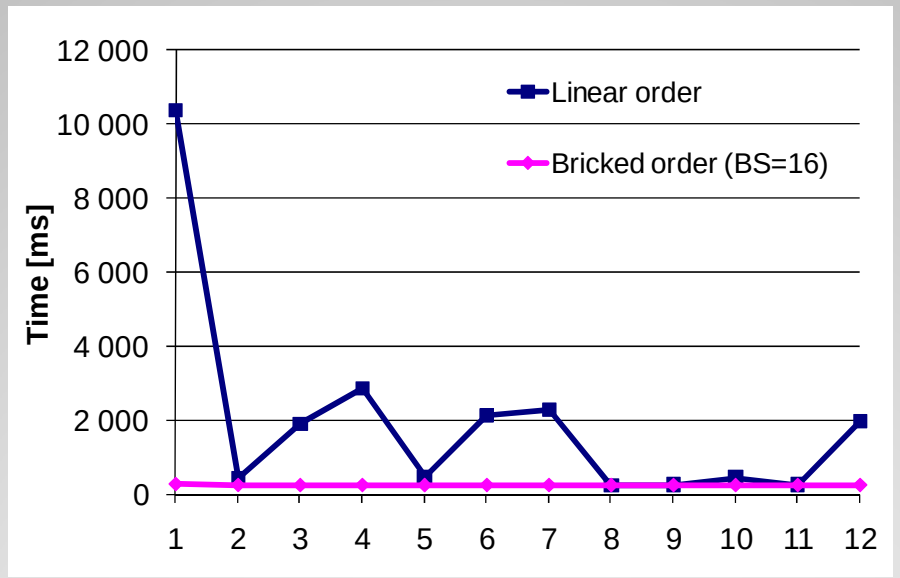
```
for (int i=0; i< MAX; i++) {
      for (int j=0; j< MAX; j++) {
            A[i][j] = A[i][j] + B[j][i];
      }
}


for (int i=0; i< MAX; i+=BLOCK_SIZE) {
      for (int j=0; j< MAX; j+=BLOCK_SIZE) {
            for (int ii=i; ii<i+BLOCK_SIZE; ii++) {
                  for (int jj=j; jj<j+BLOCK_SIZE; jj++) {
                        A[ii][jj] = A[ii][jj] + B[jj][ii];
                  }
            }
      }
}
```

```
D:\...n\ZEP\Programy\CacheTest\Release>CacheTest.exe
Normal: <item name="" value="918.87 ms">
 <subitem name="TotalTime" value="918.87"/>
 <subitem name="LastTime" value="918.87"/>
 <subitem name="Count" value="1"/>
</item>, Bricking: <item name="" value="264.57 ms">
```

Využití cache

Využití cache

- u rozsáhlých dat často zpracovávána jen malá část (region zájmu)
  - lineární uložení dat nevhodné



Načtení12 různých ROI 400×400×400
AMD Athlon 2.8 GHz, 512 MB RAM, HDD 60GB (4200rpm), WinXP

Využití cache

- více průchodů může být výhodnější
  - optimalizuje přístup do L2

```
struct Point
{
  double x,y,z;       //pozice
  double nx, ny, nz;  //normala
  double tu, tv;      //texturovaci souradnice
};

Point points[]; //pole bodu
int N;          //pocet bodu
```

```
for (int i = 0; i < N; i++)
{
  //pracuje s x,y,z a tu, tv
}

for (int i = 0; i < N; i++)
{
  //pracuje s x,y,z a nx, ny, nz
}
```

```
int iStart = 0;
while (iStart < N)
{
  int iEnd = min(iStart + PASS_SIZE, N);

  for (int i = iStart; i < iEnd; i++)
  {
    //pracuje s x,y,z a tu, tv
  }

  for (int i = iStart; i < iEnd; i++)
  {
    //pracuje s x,y,z a nx, ny, nz
  }

  iStart = iEnd  ;
}
```

```
Multiple Pass: <item name="" value="25.30 ms">
One Pass: <item name="" value="48.51 ms">
```

Vlastní SW cache

- typicky, když se data nevejdou do paměti
  - data umístěna ve „swapovacím“ souboru
- plně asociativní cache
  - položka ze souboru může být kdekoliv v cache
  - problém: jak poznat, že položka je v cache
    - možné řešení např. hash tabulky
- „přímé“ mapování
  - soubor i cache stránkovány (stejně)
  - i-tá položka j-té stránky ze souboru musí být na i-té pozici k-té stránky cache
  - počet stránek v cache malý (např. 16), rychlé zjištění, zda data cachována
  - problém: častější výpadky

Vlastní SW cache

- co dělat, když položka v cache neexistuje?
  1. pokud neexistuje volné místo v cache, musí se nějaká položka z cache napřed vyhodit
     - pro „přímé“ mapování je volné místo na fixních pozicích
     - pokud se vyhazovaná položka změnila, je nutné aktualizovat originální obsah (v souboru)
  2. položka se načte na prázdné místo v cache
- **kterou položku vyhodit?**

Vlastní SW cache

- je-li „přímé“ mapování a počet stránek cache roven jedné, není co řešit
- jinak je optimální vyhodit položku, která se nebude v budoucnosti potřebovat
  - většinou toto neznáme → nejčastěji se používá LRU (least recently used) heuristika, tj. vyhodím nejstarší položku

Vlastní SW cache

- implementace SW cache
  - spojový seznam
    - často pro plně asociativní cache
    - nelze pro „přímé“ mapování
    - snadná implementace LRU heuristiky: při přístupu k datům přesunu položku na začátek seznamu
  - pole (statické velikosti)
    - vhodné pro „přímé“ mapování
- volba závisí na aplikaci

- kontrola 1. praktické sady!
- rekurze, dělení prostoru, spojové struktury a grafy

Příště