Organizace předmětu a seznámení se s paralelizací

Cvičení 1

B4B36PDV – Paralelní a distribuované výpočty
FEL ČVUT

Osnova

- Čím se budeme zabývat?
- Hodnocení předmětu
- Úvod do paralelního hardwaru a softwaru

Organizace předmětu

Důležité informace

Přednášející

- Matěj Kafka (paralelní část)
- Michal Jakob (distribuovaná část)

Cvičící

- Petr Macejko
- Jakub Dupák
- Max Hollmann
- Jáchym Herynek
- David Milec
- Adéla Kubíková

Stránky cvičení:

`https://pdv.pages.fel.cvut.cz/`

Paralelní a Distribuované výpočty

Paralelní a Distribuované výpočty

Paralelní výpočty

- **Jeden** výpočet provádí současně **více** vláken
- Vlákna typicky sdílí paměť a výpočetní prostředky
- Cíl:
  - Zrychlit výpočet úlohy

7 týdnů

Paralelní a Distribuované výpočty

Paralelní výpočty

- **Jeden** výpočet provádí současně **více** vláken
- Vlákna typicky sdílí paměť a výpočetní prostředky
- Cíl:
  - Zrychlit výpočet úlohy

7 týdnů

Distribuované výpočty

- Výpočet provádí současně více oddělených výpočetních uzlů (často i geograficky)
- Cíle:
  - Zrychlit výpočet
  - Robustnost výpočtu

6 týdnů

Paralelní a Distribuované výpočty

Paralelní výpočty

- **Jeden** výpočet provádí současně **více** vláken
- Vlákna typicky sdílí paměť a výpočetní prostředky
- Cíl:
  - Zrychlit výpočet úlohy

7 týdnů

Distribuované výpočty

- Výpočet provádí současně více oddělených výpočetních uzlů (často i geograficky)
- Cíle:
  - Zrychlit výpočet
  - Robustnost výpočtu

6 týdnů

Docházka

Docházka na cvičení není povinná.

To ale neznamená, že byste na cvičení neměli chodit...

- Budeme probírat látku, která se Vám bude hodit u úkolů a u zkoušky.
- Dostanete prostor pro práci na semestrálních pracích.
- Konzultace budou probíhat **primárně** na cvičeních.
- Ušetříme Vám čas a nervy (nebo v to alespoň doufáme ;-)

⚠ Pokud se na cvičení rozhodnete nechodit, budeme předpokládat, že probírané látce dokonale rozumíte. Případné konzultace v žádném případě nenahrazují cvičení!

Vyžadujeme samostatnou práci na všech úlohách.

⚠ **Plagiáty jsou zakázané.**

Nepřidělávejte prosím starosti nám, ani sobě.

Na čem budeme stavět?

- Programování v jazyce C/C++ (B0B36PRP)
  - Základy programování v jazyce C/C++
  - Kompilace programů v jazyce C/C++
  - Základy objektového programování (znalost C++11 výhodou)

Na čem budeme stavět?

- Programování v jazyce C/C++ (B0B36PRP)
  - Základy programování v jazyce C/C++
  - Kompilace programů v jazyce C/C++
  - Základy objektového programování (znalost C++11 výhodou)
- Technologické předpoklady paralelizace (B4B36OSY)
  - Vlákna a jejich princip
  - Metody synchronizace a komunikace vláken

Na čem budeme stavět?

- Programování v jazyce C/C++ (B0B36PRP)
  - Základy programování v jazyce C/C++
  - Kompilace programů v jazyce C/C++
  - Základy objektového programování (znalost C++11 výhodou)
- Technologické předpoklady paralelizace (B4B36OSY)
  - Vlákna a jejich princip
  - Metody synchronizace a komunikace vláken
- Základní znalost fungování počítače a procesoru (B4B35APO)

Na čem budeme stavět?

- Programování v jazyce C/C++ (B0B36PRP)
  - Základy programování v jazyce C/C++
  - Kompilace programů v jazyce C/C++
  - Základy objektového programování (znalost C++11 výhodou)
- Technologické předpoklady paralelizace (B4B36OSY)
  - Vlákna a jejich princip
  - Metody synchronizace a komunikace vláken
- Základní znalost fungování počítače a procesoru (B4B35APO)
- Znalost základních algoritmů (B4B33ALG)

Opakování

Vygenerování build scriptů

```
cmake <src dir> -B <build dir> -DCMAKE_BUILD_TYPE=Release
```

Zde `<src dir>` je složka se souborem `CMakeLists.txt`.

Kompilace

```
cmake --build <build dir>
```

Zde `<build dir>` je složka s vygenerovanými soubory pro sestavení programu.

Nebo použijte IDE s dobrou podporou C++, například CLion (multiplatformní).

Cvičení

Vyzkoušejte na souboru `0hello.cpp`.

Podpora C++20

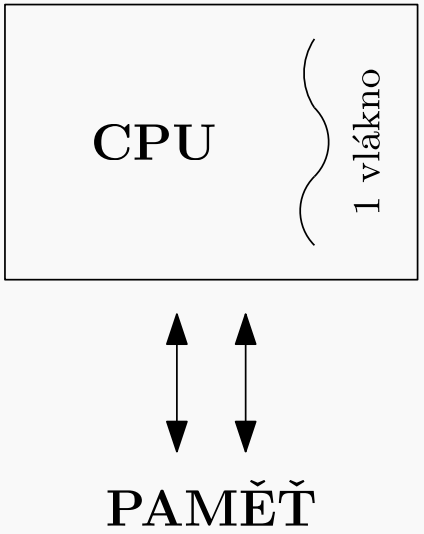
Vyzkoušejte, že vám funguje kompilace C++20 na souboru `0cpp20.cpp`.

Pokud se vám nepodaří sestavit program, zkuste si nainstalovat novější verzi kompilátoru do **7. cvičení**, kde budeme potřebovat některé nové vlastnosti C++20.

Dnes smažte `0cpp20.cpp` z `CMakeLists.txt` a pokračujte dále.

Pro připomenutí

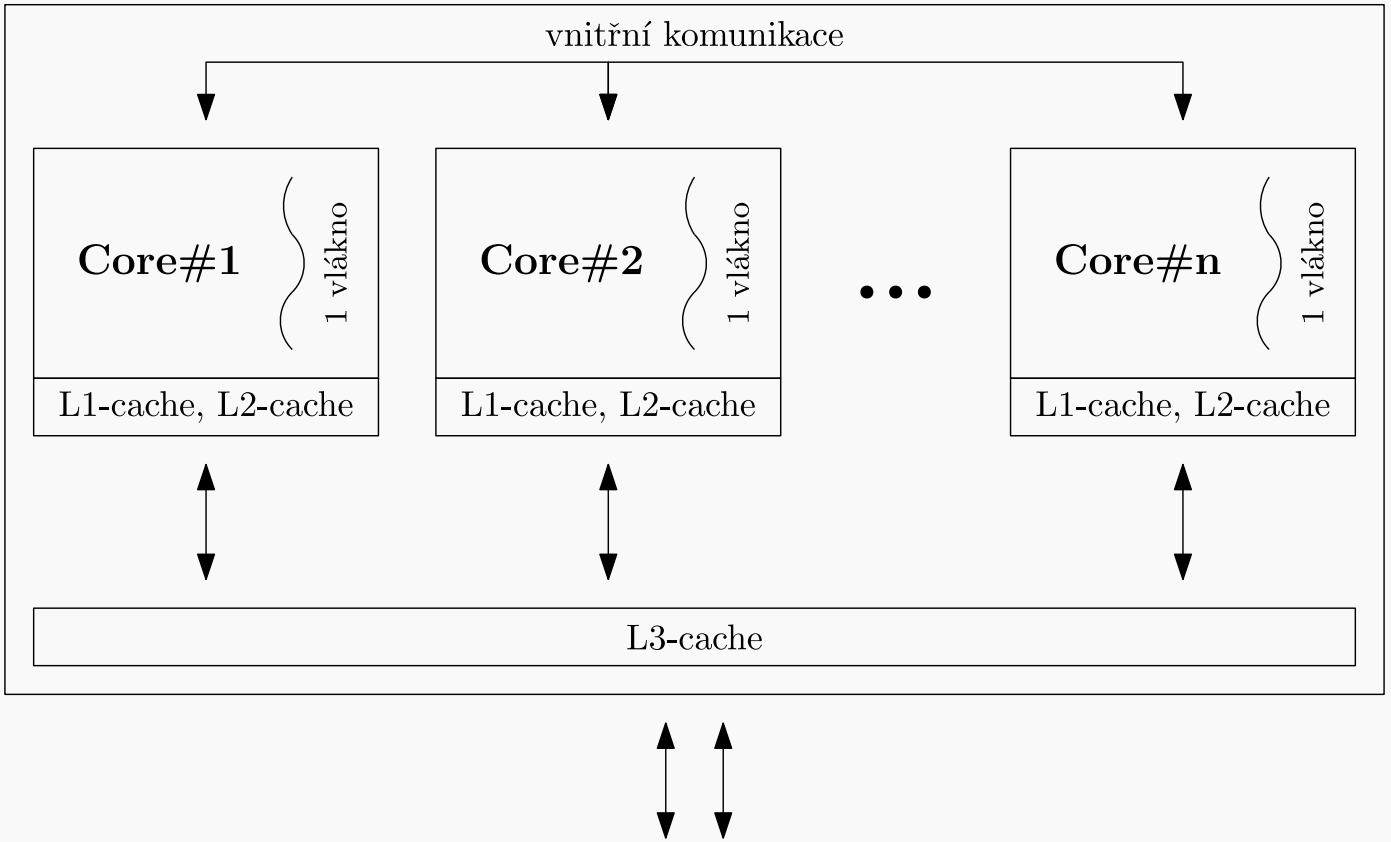
Cílem paralelních výpočtů je dosáhnout zvýšení výkonu

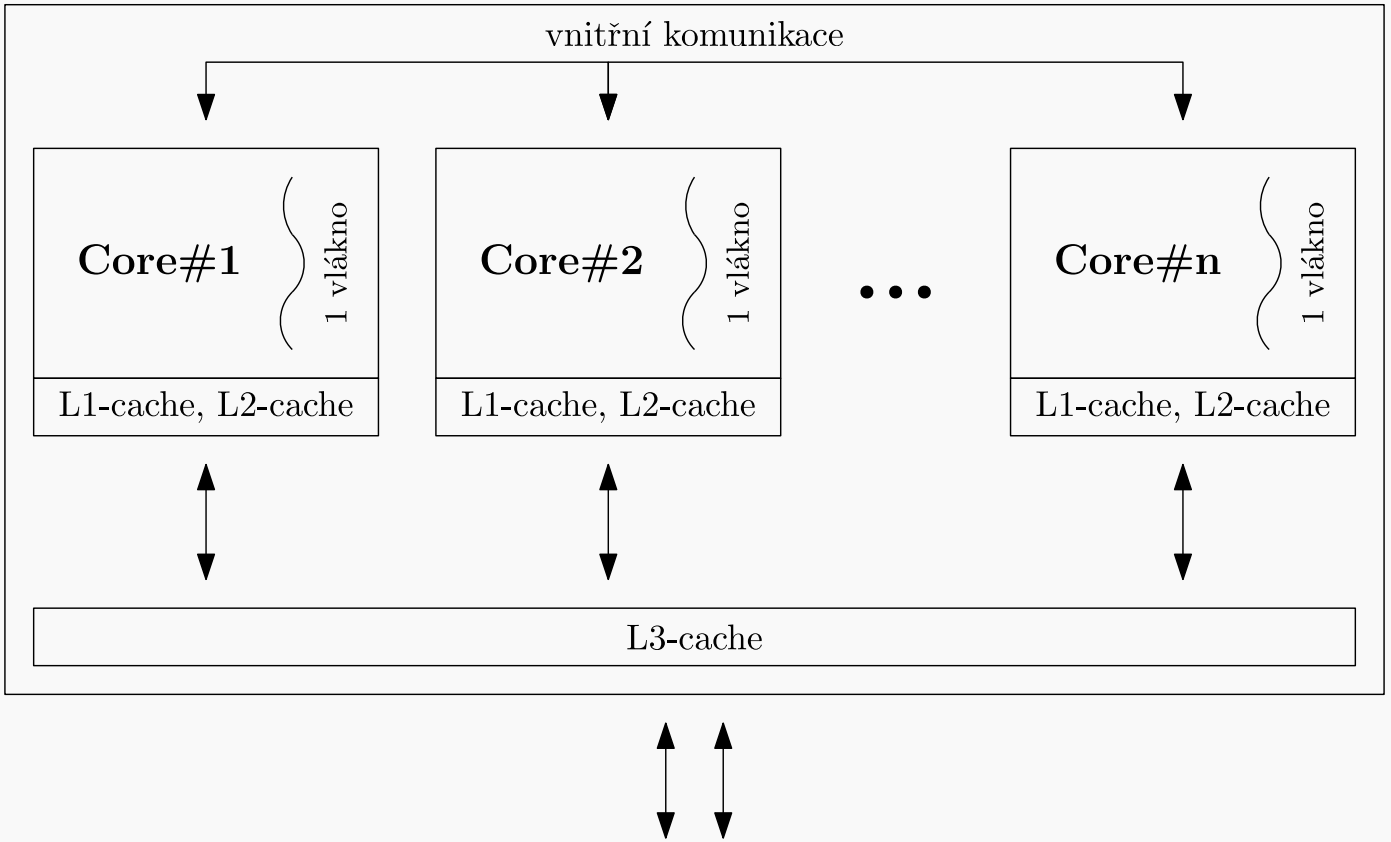


von Neumannova architektura

- Jaké má nevýhody?
- Jak bychom je mohli opravit?
- A jak bychom dále mohli navýšit výkon procesoru?

Moderní procesor





Memory bandwidth

Otestujte vaši paměť pomocí kódu v `1memory.cpp`.

Paralelizace:

Paralelizace:

Pipelining (procesor)

Paralelizace:

Pipelining (procesor)

Vektorizace (kompilátor)

Paralelizace:

Pipelining (procesor)

Vektorizace (kompilátor)

Vlákna (Vy 😀)

Moderní procesor

Možné “nástrahy” použití moderního procesoru s více jádry a cache:

- Komunikace s pamětí je stále pomalá (problém **cache-miss**)
- Přístup ke sdíleným datům více vlákny (**true sharing**)
- Udržování koherence cache může být drahé (**false sharing**)
- … a jiné

Cache-miss

Pozorujete důsledky cache-missu na výkon programů:

- `1memory.cpp`
- `2matrix.cpp`.

True sharing

```
void multiply(int* number, int multiplyBy) {
 *number = (*number) * multiplyBy;
}
```

Předpokládejme `int number = 1;` a mějme dvě vlákna:

- Vlákno 1: `multiply(&number,2)`
- Vlákno 2: `multiply(&number,3)`

Co bude v proměnné `number` po skončení obou vláken?

True sharing

```
void multiply(int * number, int multiplyBy) {
 *number = (*number) * multiplyBy;
}
```

↓

```
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

---

https://godbolt.org

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Výsledek:

`number = 6`

True sharing

Vlákno 1

```
mov esi, 2



imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

True sharing

Vlákno 1

```
mov esi, 2



imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3
imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Výsledek:

`number = 6`

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]

mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3

imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]

mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3

imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Výsledek:

`number = 3`

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]


mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3


imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]


mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

```
mov esi, 3


imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Výsledek:

`number = 2`

True sharing

Vlákno 1

```
mov esi, 2
imul esi, DWORD PTR [rdi]


mov DWORD PTR [rdi], esi
ret
```

Vlákno 2

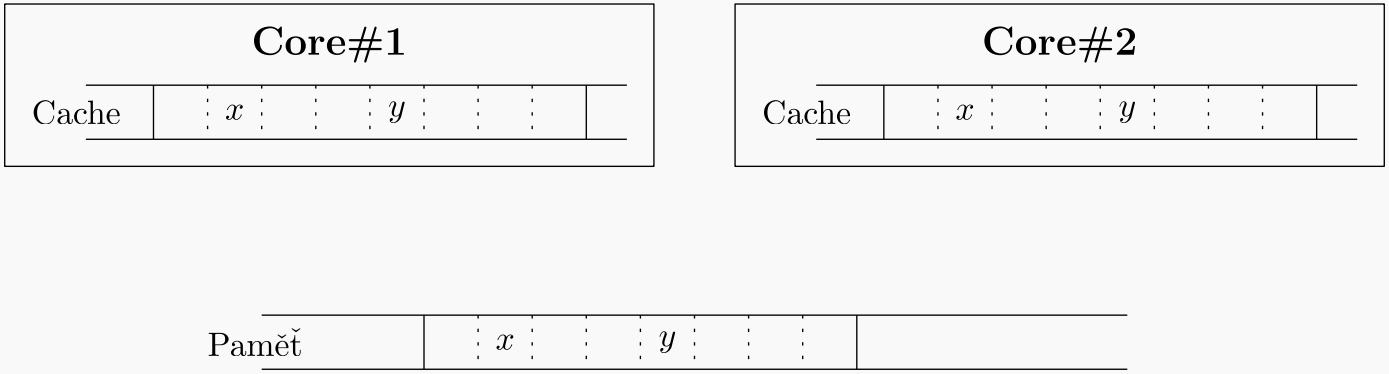
```
mov esi, 3


imul esi, DWORD PTR [rdi]
mov DWORD PTR [rdi], esi
ret
```

Jaké máme možnosti, abychom dosáhli deterministického výsledku (*který pravděpodobně chceme*)?

False-sharing

Moderní procesor pracuje s pamětí **po blocích**, které se mapují do cache.

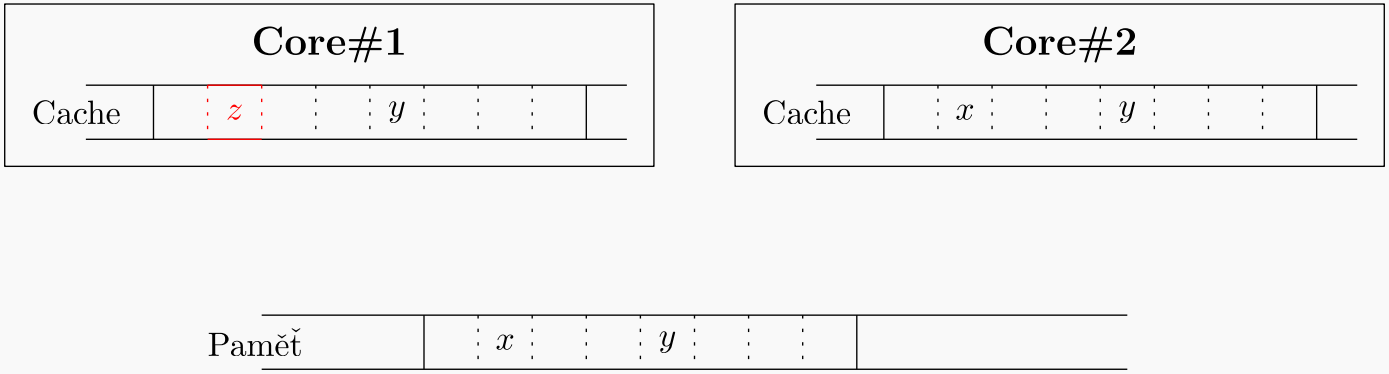
- I když vlákna nepracují se stejnými proměnnými, mohou chtít pracovat se stejným **blokem**.
- Jeden blok se pak nutně musí nacházet v cachích různých jader – a ve více kopiích



False-sharing

Moderní procesor pracuje s pamětí **po blocích**, které se mapují do cache.

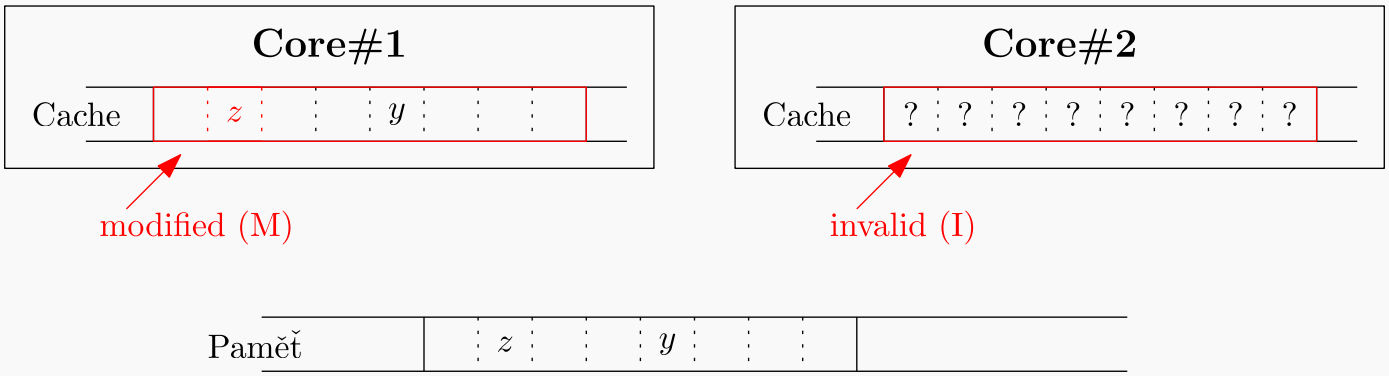
- I když vlákna nepracují se stejnými proměnnými, mohou chtít pracovat se stejným **blokem**.
- Jeden blok se pak nutně musí nacházet v cachích různých jader – a ve více kopiích



False-sharing

Moderní procesor pracuje s pamětí **po blocích**, které se mapují do cache.

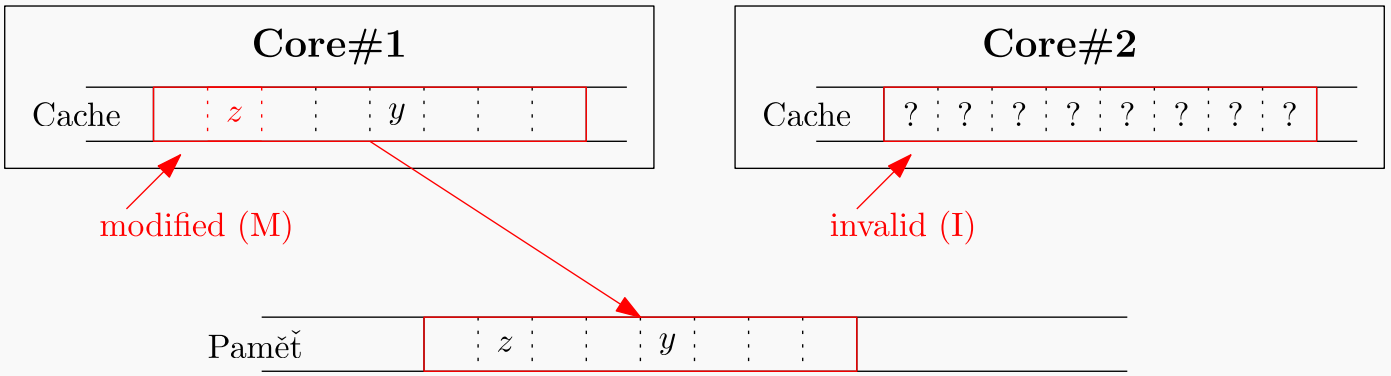
- I když vlákna nepracují se stejnými proměnnými, mohou chtít pracovat se stejným **blokem**.
- Jeden blok se pak nutně musí nacházet v cachích různých jader – a ve více kopiích



False-sharing

Moderní procesor pracuje s pamětí **po blocích**, které se mapují do cache.

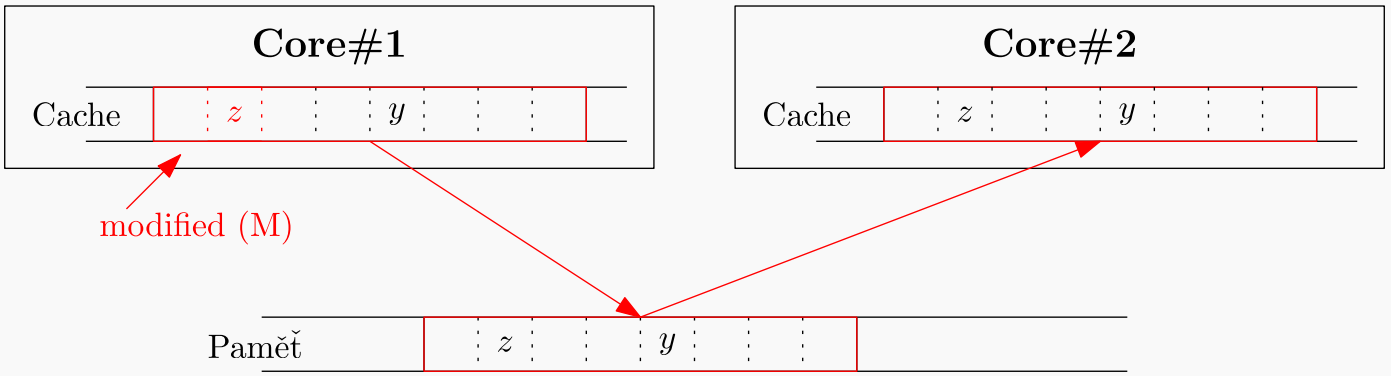
- I když vlákna nepracují se stejnými proměnnými, mohou chtít pracovat se stejným **blokem**.
- Jeden blok se pak nutně musí nacházet v cachích různých jader – a ve více kopiích



False-sharing

Moderní procesor pracuje s pamětí **po blocích**, které se mapují do cache.

- I když vlákna nepracují se stejnými proměnnými, mohou chtít pracovat se stejným **blokem**.
- Jeden blok se pak nutně musí nacházet v cachích různých jader – a ve více kopiích



- Ale právě té komunikaci s pamětí jsme se chtěli použitím cache vyhnout!

False-sharing

$\Rightarrow$ `3false_sharing.cpp`

Paralelizace v praxi

Je následující tvrzení pravdivé?

Mějme procesor s P jádry a úlohu, která při využití jednoho jádra trvá t milisekund. Využijeme-li všech P jader pro vyřešení úlohy, vyřešení úlohy zvládneme za $\frac{t}{P}$ milisekund.

Je následující tvrzení pravdivé?

Mějme procesor s P jádry a úlohu, která při využití jednoho jádra trvá t milisekund. Využijeme-li všech P jader pro vyřešení úlohy, vyřešení úlohy zvládneme za $\frac{t}{P}$ milisekund.

Tvrzení není pravdivé. Proč? Zkuste vymyslet co možná nejvíce důvodů, proč tomu tak není.

Je následující tvrzení pravdivé?

Mějme procesor s P jádry a úlohu, která při využití jednoho jádra trvá t milisekund. Využijeme-li všech P jader pro vyřešení úlohy, vyřešení úlohy zvládneme za $\frac{t}{P}$ milisekund.

Tvrzení není pravdivé. Proč? Zkuste vymyslet co možná nejvíce důvodů, proč tomu tak není.

O úlohách, kde toto tvrzení platí říkáme, že jsou tzv. *lineární* nebo také *embarassingly parallel*. Takových úloh ale v praxi potkáme velmi málo.

Je následující tvrzení pravdivé?

Mějme pole o 1,000,000 prvků. S každým prvkem pole máme za úkol 500× provést "magickou operaci" $x \leftarrow e^{\ln(x)}$. Tuto úlohu lze dobře paralelizovat.

```
void magic_operation(std::vector<double>& array) {
  for (size_t i = 0; i < array.size(); i++) {
    for (size_t k = 0; k < 500; k++) {
      array[i] = exp(log(array[i]));
    }
  }
}
```

Je následující tvrzení pravdivé?

Mějme pole o 1,000,000 prvků. S každým prvkem pole máme za úkol 500× provést "magickou operaci" $x \leftarrow e^{\ln(x)}$. Tuto úlohu lze dobře paralelizovat.

```
void magic_operation(std::vector<double>& array) {
  for (size_t i = 0; i < array.size(); i++) {
    for (size_t k = 0; k < 500; k++) {
      array[i] = exp(log(array[i]));
    }
  }
}
```

Tvrzení je pravdivé. Jednotlivé výpočty hodnot `array[i]` na sobě nezávisí a můžeme je rozložit mezi různá vlákna a dosáhnout téměř lineárního nárůstu výkonu.

Je následující tvrzení pravdivé?

Mějme pole o 1,000,000 prvků. S každým prvkem pole máme za úkol 500× provést "magickou operaci" $x \leftarrow e^{\ln(x)}$. Tuto úlohu lze dobře paralelizovat.

```
void magic_operation(std::vector<double>& array) {
  for (size_t i = 0; i < array.size(); i++) {
    for (size_t k = 0; k < 500; k++) {
      array[i] = exp(log(array[i]));
    }
  }
}
```

Tvrzení je pravdivé. Jednotlivé výpočty hodnot `array[i]` na sobě nezávisí a můžeme je rozložit mezi různá vlákna a dosáhnout téměř lineárního nárůstu výkonu.

A nebo bychom si mohli vzpomenout, že $\ln x$ a e^x jsou inverzní funkce. Ale to bychom neměli co paralelizovat ;-)

Je následující tvrzení pravdivé?

Mějme pole o 1,000,000 prvků. S každým prvkem pole máme za úkol 100× provést "magickou operaci" $x \leftarrow e^{\ln(x)}$. Tuto úlohu lze dobře paralelizovat.

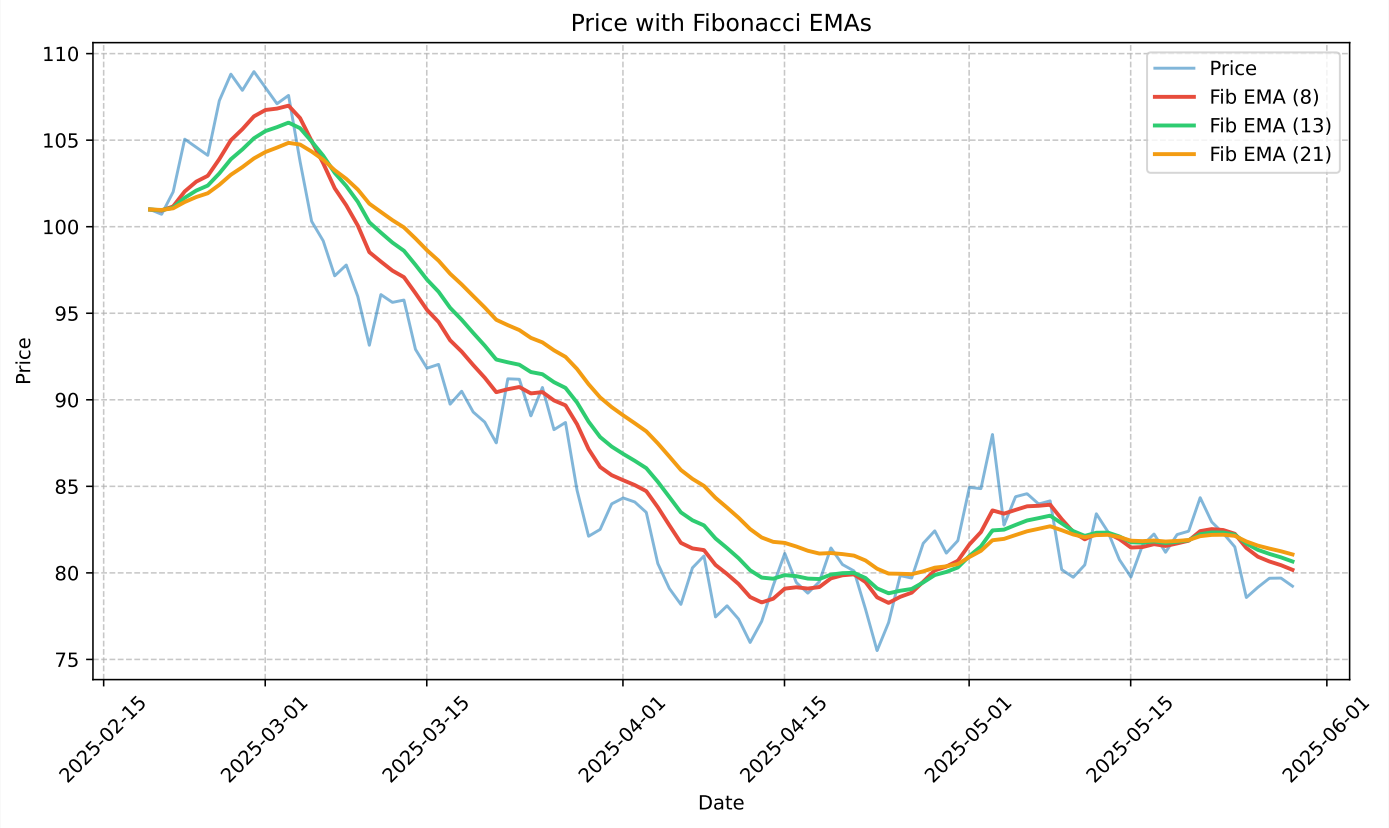
```
void magic_operation(std::vector<double>& array) {
  for (size_t i = 0; i < array.size(); i++) {
    for (size_t k = 0; k < 500; k++) {
      array[i] = exp(log(array[i]));
    }
  }
}
```

Proč jsme ale nedosáhli P-násobného zrychlení (kde P je počet jader procesoru?). Vzpomeňte si na Amdahlův zákon.

$$S = \frac{1}{s + \frac{1-s}{P}}$$

Dokážete říct, co tvoří neparalelizovatelnou část programu?
(vyžadující $s\%$ času)

Exponenciální kluzavý proměr



$$\mathrm{EMA}_0 = \mathrm{price}_0$$
$$\mathrm{EMA}_i = \mathrm{price}_i * \mathrm{k} + \mathrm{EMA}_{i-1} * (1 - \mathrm{k})$$

---

https://en.wikipedia.org/wiki/Exponential_smoothing

Exponenciální kluzavý průměr

Je následující tvrzení pravdivé?

Proces výpočtu exponenciálního kluzavého průměru je možné snadno paralelizovat.

Exponenciální kluzavý průměr

Je následující tvrzení pravdivé?

Proces výpočtu exponenciálního kluzavého průměru je možné snadno paralelizovat.

Tvrzení není pravdivé. Co nám dělá s paralelizací problémy?

Exponenciální kluzavý průměr

Je následující tvrzení pravdivé?

Proces výpočtu exponenciálního kluzavého průměru je možné snadno paralelizovat.

Tvrzení není pravdivé. Co nám dělá s paralelizací problémy?

Uvažujte množinu datových sad. Mohli bychom využít více jader v tomto případě?