Porovnání rychlostí násobení matic pomocí OpenMP

Moravec Vojtěch

Zimní semestr 2018

Obsah

1 Typy matic

Součástí zkoumání, bylo porovnání různých typů matic. Jako různé typy rozumíme matice s odlišným způsobem uložení dat. Zvolili jsme 4 typy uložení, do jednorozměrného pole, dvourozměrného pole a obdobně jsme použili *std::vector*.

Pro následné zjednodušení jsme vytvořili třídu nadřazenou všem těmto typům. Tato třídy definuje jednotný způsob přístupu k prvkům matice a to před operátor ().

```
template<typename T>
class BasicMatrix
{
protected:
  long rowCount;
  long colCount;
public:
  virtual T& operator()(long row, long col) = 0;
  virtual const T& operator()(long row, long col) const = 0;
};
```

Příklad přetížení těchto operátoru si můžeme ukázat například na matici, která je uložena v jednorozměrném poli.

```
template <typename T>
class ArrayMatrix : public BasicMatrix<T>
{
  T& operator()(long row, long col)
  {
    return data[((row * this->colCount) + col)];
  }

  const T& operator()(long row,long col) const
  {
    return data[((row * this->colCount) + col)];
  }
};
```

2 Způsoby násobení matic

V tomto dokumentu zkoumáme rychlost násobení matic pomocí 3 vnořených *for* cyklů. Tyto cykly se dají přehazovat, takže dostáváme 6 různých způsobu násobení. Těchto 6 různých způsobu budeme zkoumat jak sériově tak paralelně.

```
for (long resRow = 0; resRow < result.rows(); resRow++)
{
  for (long resCol = 0; resCol < result.cols(); resCol++)
  {
    for (long aCol = 0; aCol < A.cols(); aCol++)
    {
      result(resRow, resCol) +=
        (A(resRow, aCol) * B(aCol, resCol));
    }
  }
}
```

Pokud si označíme první cyklus f_1, druhý cyklus f_2 a poslední f_3, dostáváme těchto 6 "typů" násobení:

1. $m_1 = f_1\ f_2\ f_3$
2. $m_2 = f_1\ f_3\ f_2$
3. $m_3 = f_2\ f_1\ f_3$
4. $m_4 = f_2\ f_3\ f_1$
5. $m_5 = f_3\ f_1\ f_2$
6. $m_6 = f_3\ f_2\ f_1$

Pro vylepšení paralelizace násobení se hodí používat další způsob, a to takový, že výsledek uložíme do výsledné matice až po provedení posledního cyklu. Toto se dá využít u cyklu m_1 a m_3 typu, a nové typy označíme jako m_{1_tmp} a m_{3_tmp}. Tyto cykly poté vypadají následně:

```
int tmp;
for (long resRow = 0; resRow < result.rows(); resRow++)
{
  for (long resCol = 0; resCol < result.cols(); resCol++)
  {
    tmp = 0;
    for (long aCol = 0; aCol < A.cols(); aCol++)
    {
      tmp += (A(resRow, aCol) * B(aCol, resCol));
    }
    result(resRow, resCol) = tmp;
  }
}
```

Pro využití paralelismu použijeme direktivy kompilátoru z OpenMP, přesněji *#pragma omp parallel for*. Tuto direktivu budeme používat před prvním *for* cyklem. Pro násobení typu m_1 také vyzkoušíme přesunout tuto direktivu před druhý cyklus.

3 Způsob měření

Měření probíhalo na maticích 1000×1000. Všechna měření se prováděla 5-krát a výsledný čas je tedy průměrem 5-ti časů. Čas jsme měřili pomocí *std::chrono::high_resolution_clock*.

Matice byly naplněny následující způsobem:

$$A[x, y] = x + y \tag{1}$$

kde A je matice, x je řádek, y sloupec a $A[x, y]$ určuje hodnotu matice A na x-tem řádku a y-novém sloupci.

OpenMP jsme testovali na 1-12 threadech. 12 jsme zvolili jako maximum, neboť počítač, na kterém běžely testy má 6-ti jádrový procesor s celkem 12 thready (Intel Core i7-8750H).

4 Výsledky měření

Všechny časy v této sekci jsou udávány v milisekundách.

4.0.1 Násobení bez OpenMP

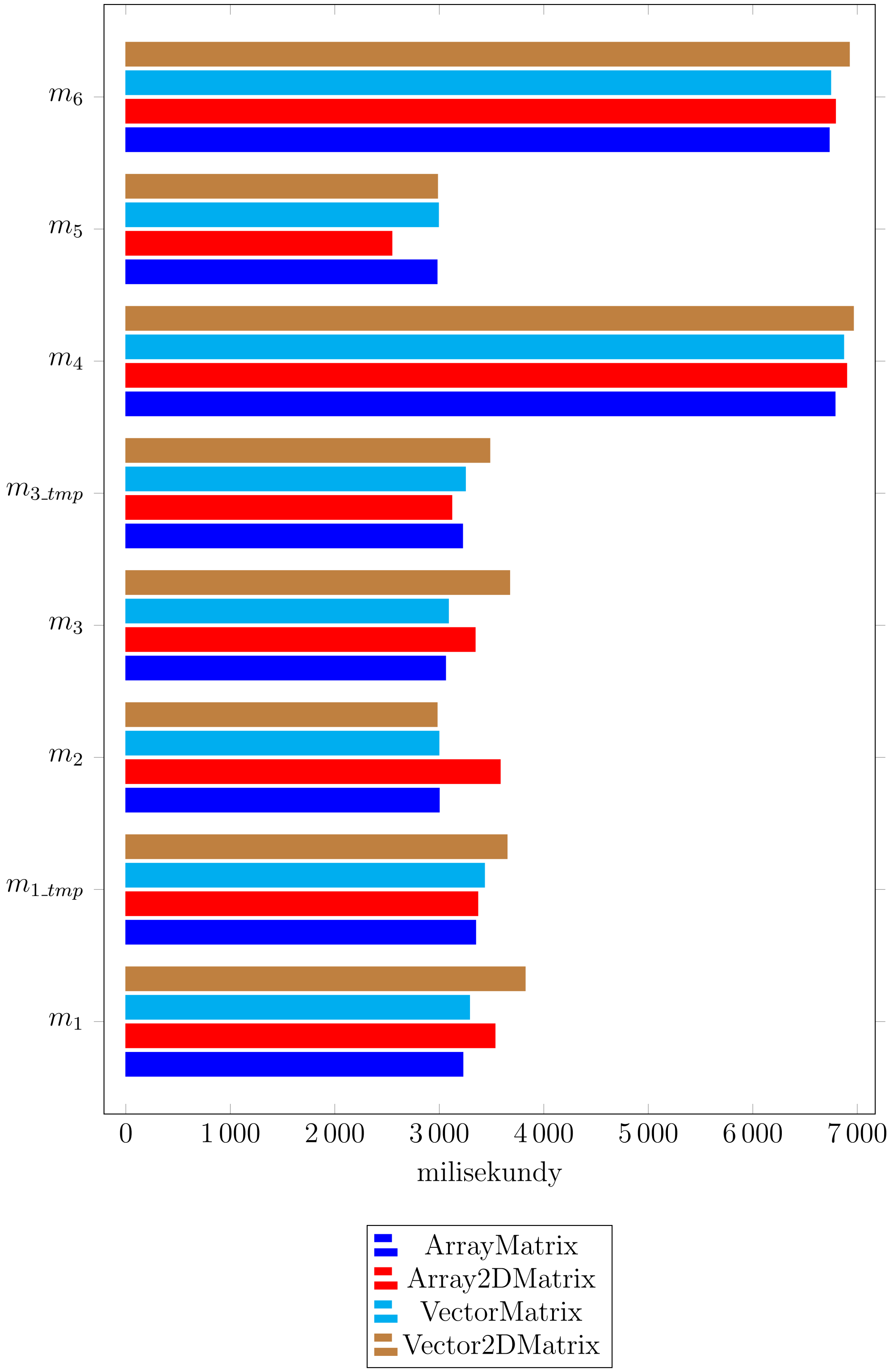
Na Obrázku 1 můžeme vidět porovnání času výpočtu, vzhledem k metodě uložení matice a ke zvolenému typu násobení. Z tohoto Obrázku vyčteme, že uložení do dvoudimenzionálního vektoru je zřejmě nejpomalejší. Nejrychleji se pak jeví uložení v typickém poli. Obecně typ matice nehraje tak velkou roli jako zvolený typ násobení. Očividně pořadí *for* cyklů v m_4 a m_6 dává nejmenší smysl, zatímco ostatní jsou si téměř rovny.

4.0.2 Násobení s OpenMP

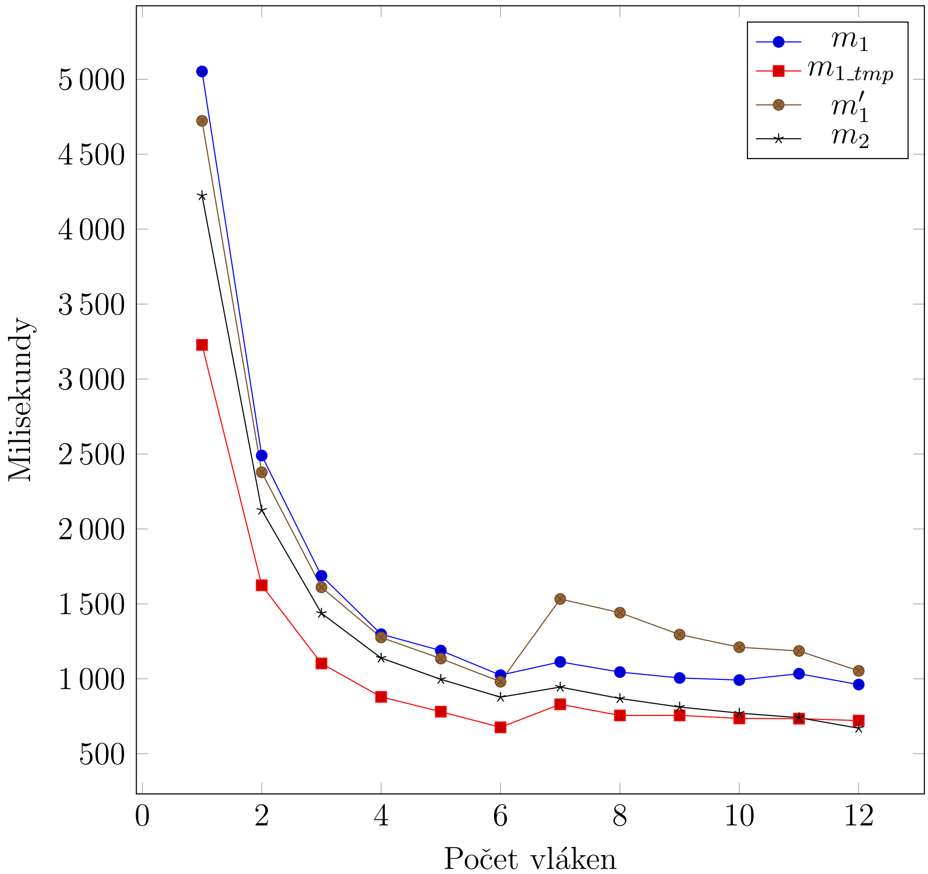
Pro ukázku jsme vybrali násobení m_1, m_{1_tmp} , m'_1 a m_2. Všechny násobení v grafu 2 byly prováděny na matici, typu jednodimenzionálního pole. m'_1 je násobení, kde jsme přesunuli *#pragma omp parallel for* před druhý cyklus. Tato změna nevedla k lepšímu výsledku. Obecně si všimneme, že násobení matic pomocí OpenMP škáluje velmi dobře do 6-tého vlákna, dále se sice čas stále zmenšuje, ale již ne tak razantně. Jak jsme již uvedli uvedli, mezivýsledek pro násobení m_{1_tmp} opravdu zlepšuje rychlost, neboť třetí cyklus může být lépe zparalelizován. Nejsou zde uvedeny všechny typy násobení, ale jako nejlepší vyšlo násobení m_2, které dokázalo vynásobit matici 1000×1000 za $670,4$ milisekundy.

5 Závěr

Pro násobení různě uložených matic, jsme zjistili, že jejich uložení nehraje takovou roli tak jako pořadí tří *for* cyklů, které provádějí vlastní násobení. Dále jsme zjistili, že pomocí OpenMP, umíme tuto operaci velmi dobře zparalelizovat a nejrychlejší čas jsme dostali pro násobení m_2 s 12-ti vlákny.



Obrázek 1: Rychlost násobení vzhledem k typu matice



Obrázek 2: Škálování rychlosti násobení m_1, m_{1_tmp} s počtem threadů