Semestrální projekt MI-PAP 2015/2016:

Paralelní algoritmus pro FFT

Petr Klejch
29. dubna 2016

1 Definice problému a popis sekvenčního algoritmu

1.1 Obecný popis úlohy

1.1.1 Diskrétní Fourierova transformace

Diskrétní Fourierova transformace (DFT) je transformace převádějící vstupní signál na signál vyjádřený pomocí funkcí sin a cos. Obecně se diskrétní Fourierova transformace vstupního vektoru o velikosti N spočítá jako:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi ikn/N},$$

což odpovídá složitosti $\mathcal{O}(N^2)$. Byl však ale objeven algoritmus, který spočítá diskrétní Fourierovu transformaci v čase $\mathcal{O}(N \log N)$ jménem rychlá Fourierova transformace (FFT).

1.1.2 Rychlá Fourierova transformace

Rychlá Fourierova transformace je algoritmus typu rozděl a panuj, který rekurzivně dělí vstupní vektor na menší částí, na tyto části opět rekurzivně aplikuje algoritmus FFT a poté zkombinuje výsledné části. Nejčastější implementace je varianta, která dělí vstupní vektor na poloviny a tudíž vstupní vektor musí být velikosti N^2.

1.2 Popis sekvenčního algoritmu

Jelikož je dle zadání vstupní vektor o velikosti $N = 2^{k_1} * 3^{k_2}$, nemohla být použita pouze nejčastější implementace FFT - radix-2 FFT, která dělí vektor rekurzivně na poloviny.

Bylo využito obecnější varianty Cooley-Tukey algoritmu FFT, který předpokládá vstupní vektor o velikosti $N = Q*P$. Tento algoritmus převede vstupní vektor do matice o rozměrech $Q * P$, aplikuje $Q\times$ algoritmus DFT na vektor o velikosti P (řádky matice), poté vynásobí všechny prvky matice tzv. twiddle faktory ($e^{-2\pi ik/N}$), následně se matice transponuje a aplikuje se $P\times$ algoritmus DFT na vektor o velikosti Q (opět řádky matice).

Jelikož víme, že vstupní vektor je velikosti $N = 2^{k_1} * 3^{k_2}$, a tedy $Q = 2^{k_1}$ a $P = 3^{k_2}$, je možné použít na řádky matice radix-2, resp. radix-3 FFT variantu, která dělí vektor na poloviny, resp. na třetiny.

Sekvenční algoritmus se tedy skládá z těchto kroků:

1. Faktorizace N na čísla Q a P, kde $Q = 2^{k_1}$ a $P = 3^{k_2}$.
2. Převedení vstupního vektoru do matice.
3. Aplikace $Q\times$ algoritmu radix-3 FFT na řádky matice.
4. Vynásobením všech prvků matice twiddle faktory.
5. Transpozice matice.
6. Aplikace $P\times$ algoritmu radix-2 FFT na řádky matice.
7. Přečtení matice po řádcích, která obsahuje výsledný transformovaný vektor.

2 Popis paralelního algoritmu a jeho implementace v OpenMP

Sekvenční algoritmus byl vhodný pro paralelní zpracování, a tak vyžadoval minimum úprav. Paralelizovány byly zejména cykly, které počítají $Q\times$ (resp. $P\times$) FFT po řádcích. Jelikož se každý řádek zpracovává zvlášť, nebyla nutná žádná synchronizace (krok (3) a (6)). Dále byla paralelizována část, která transponuje matici (krok (5)) a část která násobí každý prvek matice twiddle faktory (krok (4)).

2.1 Vektorizace

Na architektuře x86 drtivá část cyklů nebyla vektorizována, kvůli datovým závislostem. Dále některé cykly obsahují podmínky nebo volání funkcí, které zabraňují vektorizaci.

Na architektuře Xeon Phi byla situace o něco lepší. Kompilátor byl schopen vektorizovat několik cyklů. Bohužel cykly, které jsou prováděny nejčastěji vektorizovány nebyly, proto zrychlení oproti nevektorizované verzi je v nejlepším případě přibližně 6 %, jak lze vidět z tabulek 2 a 3.

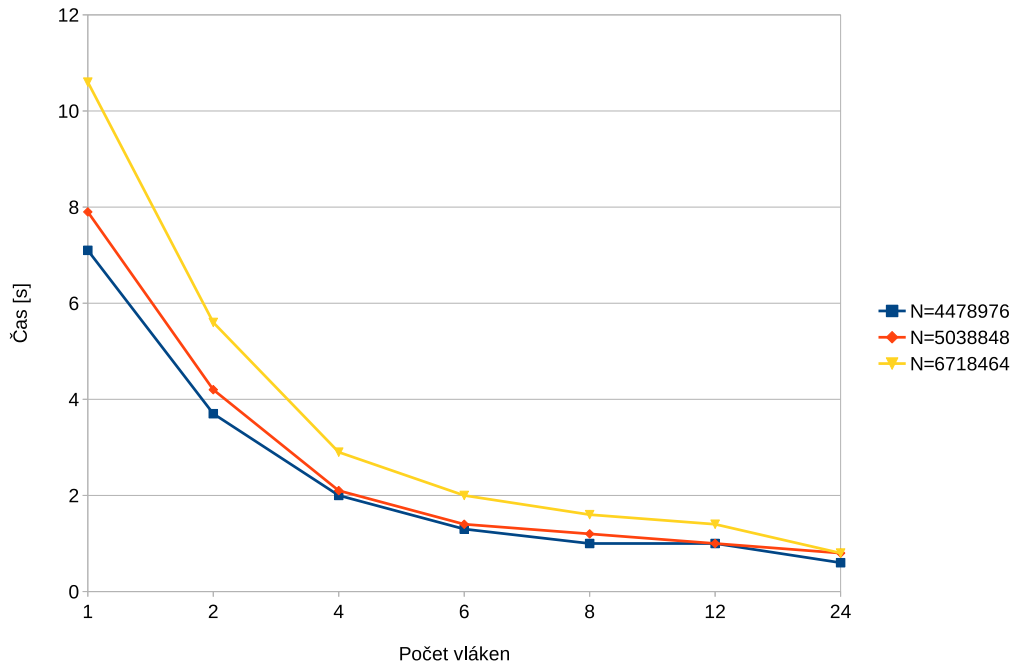
Schopnost vektorizace by se dala zlepšit přepsáním kódu, který by byl v souladu s pravidly pro vektorizaci.

2.2 Naměřené výsledky a grafy

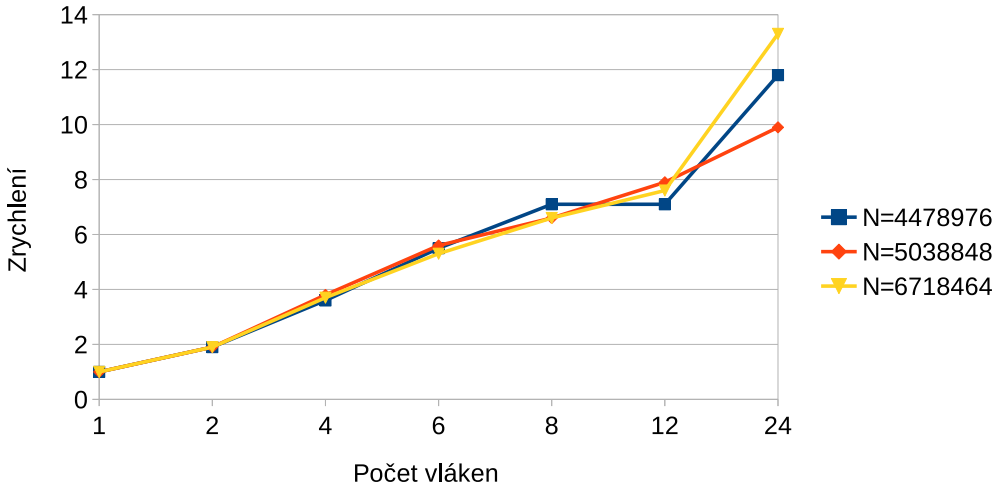
2.2.1 Architektura x86

	Velikost instance		
Počet vláken	N=4 478 976 ($2^{11} * 3^7$)	N=5 038 848 ($2^8 * 3^9$)	N=6 718 464 ($2^{10} * 3^8$)
1	7,1	7,9	10,6
2	3,7	4,2	5,6
4	2,0	2,1	2,9
6	1,3	1,4	2
8	1,0	1,2	1,6
12	1,0	1,0	1,4
24	0,6	0,8	0,8

Tabulka 1: Délka běhu úlohy v sekundách v závislosti na velikosti vstupních dat a počtu vláken.



Obrázek 1: Graf délky běhu úlohy v závislosti na počtu vláken.



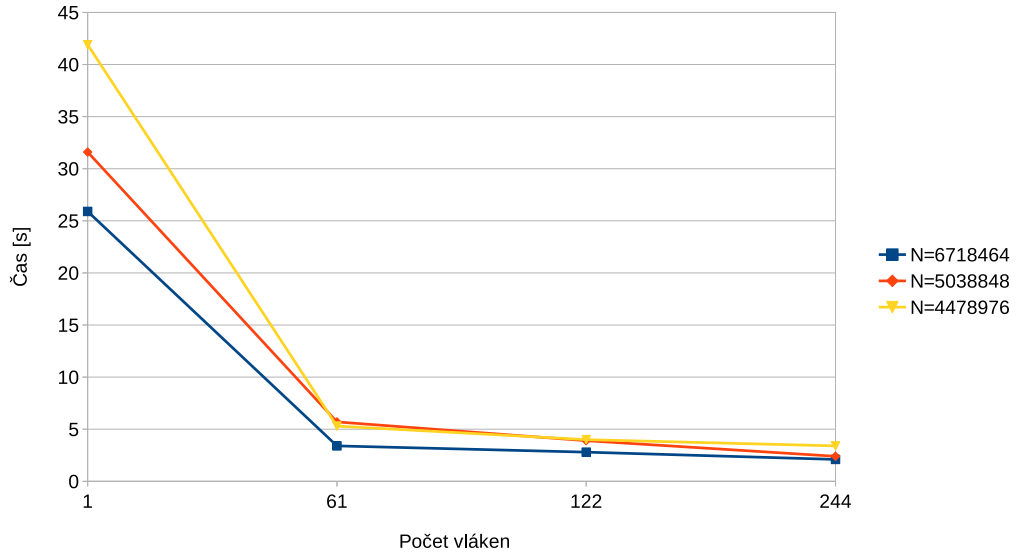
Obrázek 2: Graf zrychlení v závislosti na počtu vláken.

2.2.2 Architektura Xeon Phi

Vypnutá vektorizace

	Velikost instance		
Počet vláken	N=4 478 976 ($2^{11} * 3^7$)	N=5 038 848 ($2^8 * 3^9$)	N=6 718 464 ($2^{10} * 3^8$)
1	25,9	31,6	41,9
61	3,4	5,7	5,3
122	2,8	3,9	4
244	2,1	2,4	3,4

Tabulka 2: Délka běhu úlohy s vypnutou vektorizací v sekundách v závislosti na velikosti vstupních dat a počtu vláken.

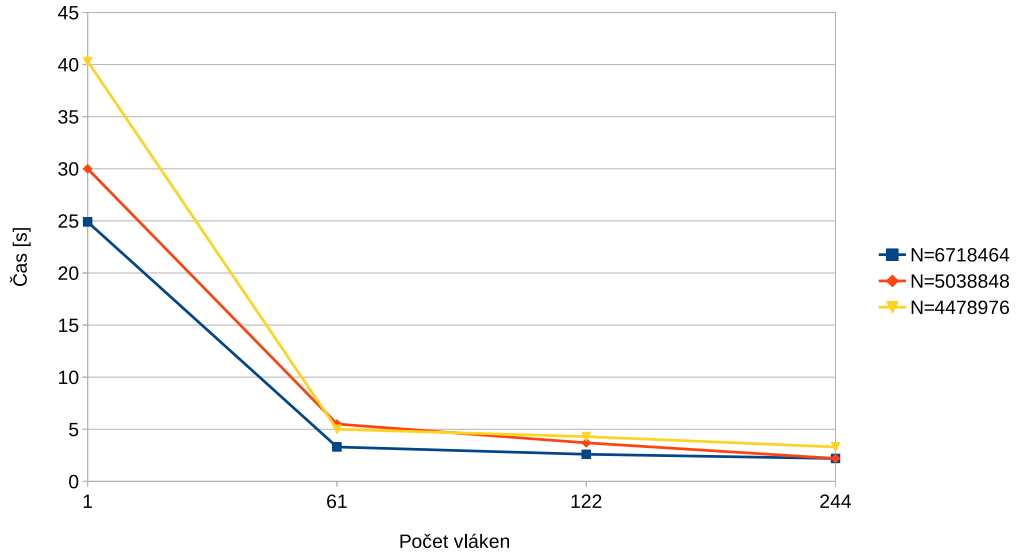


Obrázek 3: Graf délky běhu úlohy v závislosti na počtu vláken na architektuře Xeon Phi s vypnutou vektorizací.

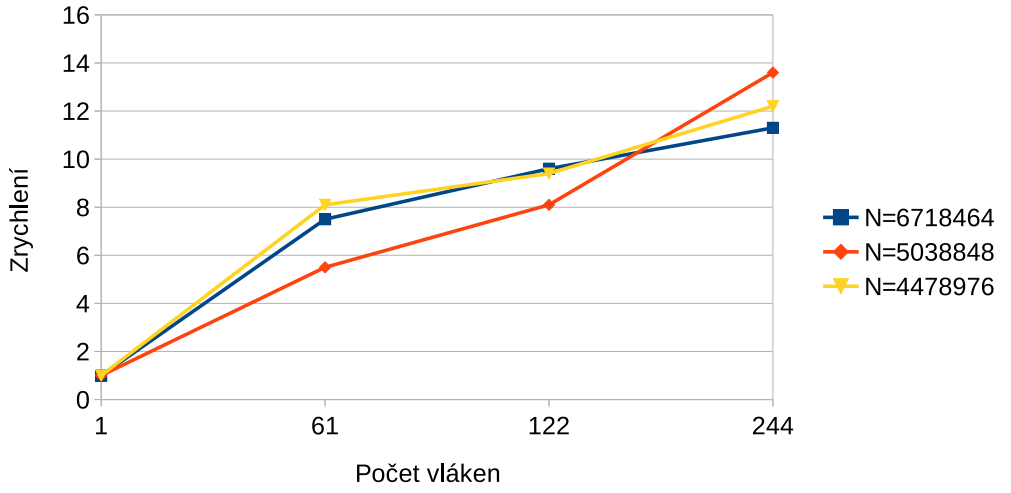
Zapnutá vektorizace

	Velikost instance		
Počet vláken	N=4 478 976 ($2^{11} * 3^7$)	N=5 038 848 ($2^8 * 3^9$)	N=6 718 464 ($2^{10} * 3^8$)
1	24,9	30	40,3
61	3,3	5,5	5
122	2,6	3,7	4,3
244	2,2	2,2	3,3

Tabulka 3: Délka běhu úlohy se zapnutou vektorizací v sekundách v závislosti na velikosti vstupních dat a počtu vláken.



Obrázek 4: Graf délky běhu úlohy v závislosti na počtu vláken na architektuře Xeon Phi se zapnutou vektorizací



Obrázek 5: Graf zrychlení v závislosti na počtu vláken na architektuře Xeon Phi se zapnutou vektorizací.

2.3 Zhodnocení

Na architektuře x86 paralelní algoritmus relativně dobře škáloval a v určitých instancích bylo dosaženo lineárního zrychlení, jak lze vidět na grafu 2.

Bohužel na architektuře Xeon Phi algoritmus škáloval velmi špatně, jak lze vidět na grafu 5. Bylo to pravděpodobně způsobeno špatnou granularitou úlohy a nevyužitou vektorizací. Jelikož vláknům byla přidělována velká část práce, a jelikož výpočetní vlákna na architektuře Xeon Phi jsou výkonnostně slabší než na architektuře x86, práce nebyla efektivně rozvržena, což vedlo ke špatným časovým výsledkům.