Univerzita Pavla Jozefa Šafárika v Košiciach
Prírodovedecká fakulta

ZRÝCHLENIE VÝPOČTU SPLAJN POVRCHOV

ŠVK PRÁCA

Študijný odbor: Informatika
Školiace pracovisko: Ústav informatiky
Vedúci záverečnej práce: doc. RNDr. Csaba Török, CSc.

Košice 2016 Bc. Viliam Kačala

Zadanie práce

Abstrakt

Splajny sú dôležitá súčasť počítačovej grafiky. Jedná sa o matematický model krivky a plochy služiaci na čo „najlepšie spojenie“ konečnej množiny bodov. Termín „najlepšie spojenie“ v našom prípade znamená hladká, matematicky ľahko vyjadriteľná plocha s čo najmenším zakrivením. Využitie splajnov v grafike je veľmi široké od rôznych CAD aplikácií, v štatistike, alebo v analýze dát. Splajny existujú v mnohých formách, či už vo forme krivky v rovine, rôznych trojrozmerných telies, atď.. Táto práca si dáva za cieľ navrhnúť, analyzovať a implementovať nový algoritmus pre bikubickú interpoláciu v trojrozmernom priestore.

Abstract

Splines are important part of computer graphics. It is a mathematical model of curve and surface for the "best connection" of any finite set of points. The term "best connection" in this case means smooth, easily calculable mathematical surface with minimal curvature. Use of splines in graphics varies from large variety of CAD applications, statistics or in data analysing. Splines exist in many forms, whether in the form of curves in the plane, a variety of three-dimensional bodies, etc.. This work aims to desig, analyze and implement a new algorithm for counting and generating splines bicubic clamped interpolation in three-dimensional space.

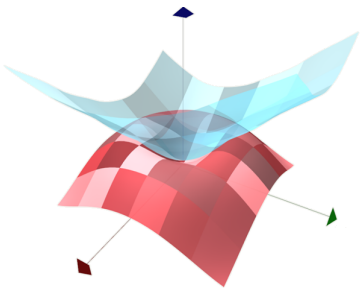
Obsah

Úvod

Témou práce sú priestorové splajn povrchy, pričom naším cieľom je preskúmať nové poznatky o splajnoch, na ktorých istý čas pracuje vedúci tejto práce doc. RNDr. Csaba Török, CSc.

Výsledok tejto práce je návrh novej metódy výpočtu derivácií splajnu v jeho uzloch, jej porovnanie s de Boorovym postupom z ktorého vychádzame, vysvetlenie zrýchlenia implementácia aplikácie na výkonnostne oboch metód.

Prínos zrýchlenia výpočtu splajnov spočíva v lepších možnostiach modelovania ľubovoľných trojrozmerných útvarov v podobe polynomických funkcií. Akékoľvek zrýchlenie totiž znamená možnosť v reálnom čase modelovať zložitejšie objekty alebo fyzikálne dáta, čo je užitočné nielen vzhľadom na vizuálnu reprezentáciu, ale aj na skúmanie fyzikálnych vlastností ako napríklad aerodynamické vlastnosti lietadiel a podobne.



Štruktúra práce sa dá zhrnúť do troch hlavných bodov.

- **De Boorov model a redukovaný algoritmus** Vysvetlenie pojmu splajn, De Boorvho modeu výpočtu derivácií a návrh zrýchlenia tohto modelu pre rovnomerné mriežky uzlov.
- **Inštrukčný paralelizmus**
  Vysvetlenie vplyvu moderných inštrukčných sád procesorov na zrýchlenie nového algoritmu.
- **Zrýchlenie**
  Očakávané zrýchlenie výpočtov novým algoritmom a namerané výsledky.

1 De Boorov model a redukovaný algoritmus

V tejto časti si postupne zadefinujeme pojem splajnu. Následne si popíšeme dve metódy výpočtu derivácií splajnu a konštrukciu celej splajnovej plochy.

Definícia 1.1 Nech $I \geq 0$ a $J \geq 0$ sú prirodzené čísla, $(u_0, \ldots, u_{I-1})$ a $(v_0, \ldots, v_{J-1})$ sú rastúce postupnosti. Nech pre každé i z $\{0, 1, \ldots, I-2\}$ a j z $\{0, 1, \ldots, J-2\}$ funkcie $S_{i,j}$ sú bipolynomické funkcie premenných x a y, ktoré spĺňajú

- $S_{i,j}(u_{i+1}, y) = S_{i+1,j}(u_{i+1}, y)$,
- $S_{i,j}(x, v_{j+1}) = S_{i,j+1}(x, v_{j+1})$,
- $\frac{\partial S_{i,j}}{\partial x}(u_{i+1}, y) = \frac{\partial S_{i+1,j}}{\partial x}(u_{i+1}, y)$
- $\frac{\partial S_{i,j}}{\partial y}(x, v_{j+1}) = \frac{\partial S_{i,j+1}}{\partial y}(x, v_{j+1})$,

Funkciu S z intervalu $[u_0, u_{I-1}] \times [v_0, v_{J-1}]$ do $\mathbb{R}$ pre ktorú platí:

$$S(x,y) = \begin{cases} S_{0,0}(x,y) & \text{pre } (x,y) \in [u_0,u_1] \times [v_0,v_1], \\ S_{0,1}(x,y) & \text{pre } (x,y) \in [u_0,u_1] \times [v_1,v_2], \\ \vdots & \\ S_{0,J-1}(x,y) & \text{pre } (x,y) \in [u_0,u_1] \times [v_{J-2},v_{J-1}], \\ S_{1,0}(x,y) & \text{pre } (x,y) \in [u_1,u_2] \times [v_0,v_1], \\ \vdots & \\ S_{1,J-1}(x,y) & \text{pre } (x,y) \in [u_1,u_2] \times [v_{J21},v_{J-1}], \\ \vdots & \\ S_{I-1,0}(x,y) & \text{pre } (x,y) \in [u_{I-2},u_{I-1}] \times [v_0,v_1], \\ \vdots & \\ S_{I-1,J-1}(x,y) & \text{pre } (x,y) \in [u_{I-2},u_{I-1}] \times [v_{J-2},v_{J-1}], \end{cases} \tag{1.1}$$

nazveme *splajn na uzloch* $(u_0, \ldots, u_{I-1})$ *a* $(v_0, \ldots, v_{J-1})$.

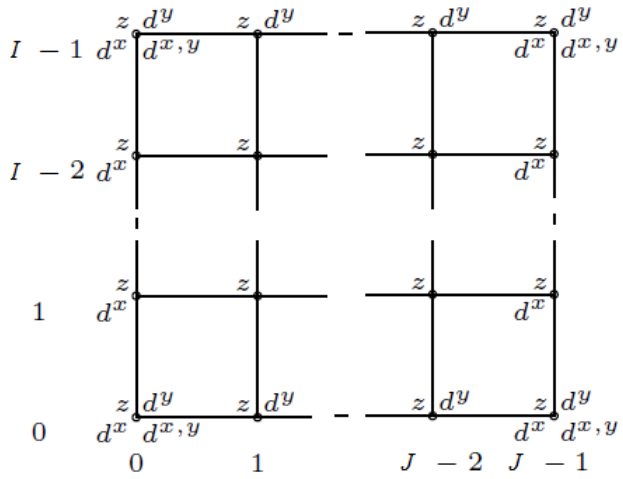
Označenie 1.2 Pri označeniach z predchádzajúcej definície označme:

- Funkcie S_{ij} nazveme *segmenty splajnu.*
- Dvojice $\langle u_i, v_j \rangle$ nazveme *uzly.*
- Uzly $\langle u_0, v_0 \rangle$, $\langle u_{I-1}, v_0 \rangle$, $\langle u_0, v_{J-1} \rangle$ a $\langle u_{I-1}, v_{J-1} \rangle$ nazveme *rohové uzly.*
- $z_{i,j} = S(u_i, v_j)$ nazveme *funkčné hodnoty splajnu v uzloch.*
- $d^x_{i,j} = \frac{\partial S(u_i, v_j)}{\partial x}$ nazveme *(smerové) x-ové derivácie v uzloch.*
- $d^y_{i,j} = \frac{\partial S(u_i, v_j)}{\partial y}$ nazveme *(smerové) y-ové derivácie v uzloch.*

- $d^{xy}_{i,j} = \frac{\partial^2 S(u_i, v_j)}{\partial x \partial y}$ nazveme *zmiešané derivácie v uzloch.*

Aby sme zostrojili splajnové segmenty potrebujeme mať dané všetky uzly aj funkčné hodnoty v uzloch a derivácie v uzloch. Ak nemáme známe všetky derivácie, vieme napriek tomu zvyšné derivácie dopočítať De Boorovov interpoláciou s malou odchýlkou aby splajn bol spojitý a hladký. Na zostrojenie splajnu pomocou De Boorovho modelu interpolácie potrebujeme mať známe

- postupnosti uzlov $(u_0, \ldots, u_{I-1})$ a $(v_0, \ldots, v_{I-1})$,
- funkčné hodnoty $\{z_{0,0}, \ldots, z_{I-1,0}, \ldots, z_{0,J-1}, \ldots, z_{I-1,J-1}\}$,
- smerové derivácie $\{d^x_{0,0}, \ldots, d^x_{I-1,0}, d^x_{0,J-1}, \ldots, d^x_{I-1,J-1}\}$,
- smerové derivácie $\{d^y_{0,0}, \ldots, d^y_{0,J-1}, d^y_{I-1,0}, \ldots, d^y_{I-1,J-1}\}$,
- zmiešané derivácie $\{d^{xy}_{0,0}, d^{xy}_{I-1,0}, d^{xy}_{0,J-1}, d^{xy}_{I-1,J-1}\}$.



Obr. 1.1: De Boorov model vstupných hodnôt pre splajnový povrch.

Kým v prípade splajnových kriviek bolo evidentné hneď od začiatky ich výskumu a aplikácie, že okrem uzlov a funkčných hodnôt $(z_0, \ldots, z_{I-1})$ v uzloch je potrebné zadať ešte dve podmienky, v našom prípade sú to dve hodnoty d_0 a d_{I-1}. V prípade interpolačných splajnových povrchoch nebolo zrejmé ktoré smerové a zmiešané derivácie je potrebné zadať. De Boor navrhol vhodný model, ktorý okrem hodnôt $z_{i,j}$ na mriežke veľkosti $I \times J$ vyžaduje zadanie smerových derivácií na okrajoch mriežky uzlov a štyri zmiešané derivácie v rohoch.

Prirodzene vidieť, že takto definovaný krivkový aj povrchový splajn je triedy C^1, teda derivácie prvého rádu v uzloch sú spojité. Pri vhodne zvolených hodnotách derivácií v uzloch vieme ale zaručiť rovnosť aj druhých derivácií, čim dostaneme splajn z triedy C^2 so spojitými deriváciami aj druhého rádu čo nám zaručí hladkosť *spojenia* jednotlivých segmentov.

Obecne splajny minimalizujú integrál druhej derivácie funkcie (napr. zakrivenie, energie, ...). Poskytujú pružný nástroj na modelovanie reálnych situácií na lokálne požiadavky, pričom ich výpočet je rýchly a stabilný. Navyše sa dá ukázať, že interpolácia funkcie kubickým splajnom je jednoznačná. Témou tejto práce sú práve metódy výpočtu prvých derivácií, ktorými je možné dosiahnuť práve spomenutý cieľ.

Témou našej práce je úloha na základe vstupných uzlov u_0, ..., u_{I-1}, v_0, ..., v_{J-1} a funkčných hodnôt $z_{0,0}$, ..., $z_{I,J}$ nájsť *hladkú*, po častiach definovanú funkciu $S : [u_0, u_{I+1}] \times [v_0, v_{J+1}] \to \mathbb{R}$ so spojitými deriváciami prvého aj druhého rádu takú, že pre každé $i \in 0, \ldots, I-1$ a $j \in 0, \ldots, J-1$ platí $z_{i,j} = S(u_i, v_j)$. Funkciu S nazývame splajn (konkrétne povrchový splajn), pričom jednotlivé časti nazveme *segmenty*.

1.1 Trojdiagonálna LU dekompozícia

Základ De Boorovej interpolácie tkvie v opakovanom počítaní systémov trojdiagonálnych lineárnych rovníc.

Definícia 1.3 Nech $n \geq 3$ je z $\mathbb{N} \cup \{0\}$. Sústavu rovníc tvaru

$$\begin{pmatrix} b_0 & c_0 & 0 & \cdots & 0 & 0 \\ a_0 & b_1 & c_1 & \cdots & 0 & 0 \\ 0 & a_1 & b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & b & c_{n-2} \\ 0 & 0 & 0 & \cdots & a_{n-2} & b_{n-1} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{n-1} \\ r_n \end{pmatrix} \tag{1.2}$$

nazveme *trojdiagonálna sústava lineárnych rovníc*.

Jeden z efektívnych spôsobov riešenia týchto rovníc spočíva v LU dekompozícii $A\mathbf{x} = L \underbrace{U\mathbf{x}}_{\mathbf{y}} = \mathbf{r}$, kde maticu A rozložíme na súčin matíc L a U v

tvare

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ l_1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & l_2 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & l_{n-1} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & u_1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & u_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & v_{n-2} & 0 \\ 0 & 0 & 0 & \cdots & 0 & v_{n-1} \end{pmatrix}. \tag{1.3}$$

Pre k z $\{1, \ldots, n\}$ sú hodnoty v_k a λ_k určené takto:

$$v_i = b, \left\{ \lambda_i = \frac{1}{v_{i-1}}, v = b - \lambda_i \right\}, i \in \{2, \ldots, n\}. \tag{1.4}$$

Pre priamy a spätný chod máme

$$\text{Priamy: } L\mathbf{y} = \mathbf{r},\ y_1 = r_1,\ \{y_i = r_i - \lambda_i\},\ i \in \{2, \ldots, n\}, \tag{1.5}$$

$$\text{Spätný: } U\mathbf{d} = \mathbf{y},\ d_i = \frac{y_i}{u_i},\ \left\{ d_i = \frac{1}{u_i}(y_i - d_{i+1}) \right\}, i \in \{n-1, \ldots, 1\}. \tag{1.6}$$

LU dekompozíciou sa rieši ako de Boorova sústava (1.11), tak aj naša redukovaná (1.22). Teraz prejdeme k cieľu práce a tým je práve De Boorov výpočet derivácií splajnu a z neho odvodený efektívnejší algoritmus.

1.2 De Boorov výpočet derivácií

Skôr než začneme s popisom De Boorvho modelu výpočtu derivácií v uzloch si položme dve označenia s ktorými budeme pracovať.

Označenie 1.4 Postupnosť $(a_0, a_1, \ldots)$ nazveme *rovnomere rastúcou* ak pre ľubovoľné i z $\{0, 1, \ldots\}$ platí $a_{i+1} > a_i$ a pre ľubovoľné i,j z $\{0, 1, \ldots\}$ platí $a_{j+1} - a_j = a_{i+1} - a_i$.

Označenie 1.5 Pre rovnomerne rastúcu postupnosť $(a_0, a_1, \ldots)$ označme hodnotu $h_a = a_1 - a_0$.

Nech sú dané rovnomerne rastúce postupnosti uzlov $(u_0, \ldots, u_{I-1})$ a $(v_0, \ldots, v_{J-1})$, kde I,J sú z $\mathbb{N} \cup \{0\}$, pričom chceme interpolovať funkčné hodnoty $\{z_{0,0}, \ldots z_{0,J-1}, \ldots, z_{I-1,0}, \ldots, z_{I-1,J-1}\}$. Výsledný splajn bude teda tvorený $(I-1) \cdot (J-1)$ segmentami. Ako bolo spomenuté, každý segment splajnu potrebuje na svoje zostrojenie štyri uzly a hodnoty z, d^x, d^y a d^{xy}. Získanie derivácií však môže byť niekedy nákladné. Príkladom môže byť v prípad, keď funkčné hodnoty $z_{i,j}$ sú získané vyhodnotením nejakej funkcie f. To môže

byť v praxi na počítači značne pomalé najmä v prípade, ak pracujeme so symbolicky zapísanou funkciou vo forme textového reťazca, ktorú je potrebné dynamicky interpretovať počas behu programu.

Algoritmus nájdený Carlom de Boorom [3] umožňuje s malou odchýlkou vypočítať hodnoty derivácií v uzloch na základe následujúcich vstupných hodnôt, ktoré máme dané:

- $z_{i,j}$ pre $i \in \{0, \ldots, I-1\}$, $j \in \{0, \ldots, J-1\}$.
- $d^x_{i,j}$ pre $i \in \{0, I-1\}$, $j \in \{0, \ldots, J-1\}$.
- $d^y_{i,j}$ pre $i \in \{0, \ldots, I-1\}$, $j \in \{0, J-1\}$.
- $d^{xy}_{i,j}$ pre $i \in \{0, I-1\}$, $j \in \{0, J-1\}$.

Poznámka 1.6 De Boorova interpolácia vo všeobecnosti nepredpokladá len rovnomerne rastúce postupnosti uzlov $(u_0, \ldots, u_I)$ a $(v_0, \ldots, v_J)$. Náš postup v ďalšej časti článku ale funguje len s takýmito postupnosťami. Preto v tejto časti budeme uvažovať De Boorov postup špeciálne pre rovnomerne rastúce postupnosti uzlov.

Zvyšné derivácie d^x, d^y a d^{xy} vieme jednoznačne vypočítať pomocou $2(I) + J + 5$ lineárnych sústav s celkovo $3IJ + I + J + 2$ rovnicami. Nižšie uvádzame modelové rovnice, pomocou ktorých sú zostrojené tieto sústavy lineárnych rovníc.

Pre $j \in \{0, \ldots, J-1\}$, teda pre každý stĺpec j vypočítame parciálne derivácie d^x

$$d^x_{i+1,j} + 4d^x_{i,j} + d^x_{i-1,j} = \frac{3}{h_u}(z_{i+1,j} - z_{i-1,j}), \quad i \in \{1, \ldots, I-2\} \tag{1.7}$$

Pre $j \in \{0, J-1\}$, teda pre prvý a posledný stĺpec vypočítame parciálne derivácie $d^{x,y}$

$$d^{xy}_{i+1,j} + 4d^{xy}_{i,j} + d^{xy}_{i-1,j} = \frac{3}{h_u}(d^y_{i+1,j} - d^y_{i-1,j}), \quad i \in \{1, \ldots, I-2\} \tag{1.8}$$

Pre $i \in \{0, \ldots, I-1\}$, teda pre každý riadok i vypočítame parciálne derivácie d^y

$$d^y_{i,j+1} + 4d^y_{i,j} + d^y_{i,j-1} = \frac{3}{h_v}(z_{i,j+1} - z_{i,j-1}), \quad j \in \{1, \ldots, J-2\} \tag{1.9}$$

Pre $i \in \{0, \ldots, I-1\}$, teda pre každý riadok j dopočítame parciálne derivácie $d^{x,y}$

$$d^{xy}_{i,j+1} + 4d^{xy}_{i,j} + d^{xy}_{i,j-1} = \frac{3}{h_v}(d^{x}_{i,j+1} - d^{x}_{i,j-1}), \quad j \in \{1, \ldots, J-2\} \tag{1.10}$$

Každá z týchto sústav má takýto maticový tvar:

$$\begin{pmatrix} 4 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 4 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 4 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 4 \end{pmatrix} \cdot \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{N-3} \\ D_{N-2} \end{pmatrix} = \begin{pmatrix} \frac{3}{h}(Y_2 - Y_0) - D_0 \\ \frac{3}{h}(Y_3 - Y_1) \\ \frac{3}{h}(Y_4 - Y_2) \\ \vdots \\ \frac{3}{h}(Y_{N-2} - Y_{N-4}) \\ \frac{3}{h}(Y_{N-1} - Y_{N-3}) - D_{N-1} \end{pmatrix}, \tag{1.11}$$

kde podľa toho o ktorú z modelových rovníc sa jedná, hodnoty N, D a Y zadávame následovne. Nech k z $1, \ldots, K-1$. Potom

- $N = I$, $h = h_u$, $D_k = d^{x}_{k,j}$ a $Y_k = z_{k,j}$, pre rovnicu 1.7,
- $N = I$, $h = h_u$, $D_k = d^{xy}_{k,j}$ a $Y_k = d^{y}_{k,j}$, pre rovnicu 1.8,
- $N = J$, $h = h_v$, $D_k = d^{y}_{i,k}$ a $Y_k = z_{i,k}$, pre rovnicu 1.9,
- $N = J$, $h = h_v$, $D_k = d^{xy}_{i,k}$ a $Y_k = d^{x}_{i,k}$, pre rovnicu 1.9.

Po vypočítaní všetkých derivácií môžeme funkčné hodnoty jednoznačne interpolovať splajnom.

1.3 Počítanie derivácií redukovanou sústavou

Cieľom tejto práce je postup zovšeobecniť pre bikubické splajny, teda pre splajny, kde interpolovaná funkcia f je typu $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Podstatou je výsledok výskumu [1] doc. Töröka a RNDr. Szaba o vzťahu medzi bikvartickými polynómami a bikubickými splajnami a odvodenie sústav na základe tohto výsledku [2].

Označenie 1.7 Teraz popíšeme tento nový algoritmus, ktorý pracovne označíme ako *redukovaný algoritmus*. Ďalej budeme pôvodný de Boorov postup, označovať pojmom *plný algoritmus*.

Vstupné hodnoty sú identické ako pri pôvodnom algoritme. Teda máme dané $(u_0, ..., u_{I-1})$ a $(v_0, ..., v_{J-1})$, kde I,J sú z $\mathbb{N} \cup \{0\}$, pričom chceme zostrojiť splajn S, ktorý interpoluje hodnoty $\{z_{0,0}, \ldots, z_{I-1,0}, \ldots, z_{0,J-1} \ldots z_{I-1,J-1}\}$ aby pre každé $i \in \{0, \ldots, I-1\}$, $j \in \{0, \ldots, J-1\}$ platilo $z_{i,j} = S(u_i, v_j)$. Pre pripomenutie potrebujeme ešte poznať tieto hodnoty.

- $d^x_{i,j}$ pre $i \in \{0, I-1\}$, $j \in \{0, \ldots, J-1\}$.
- $d^y_{i,j}$ pre $i \in \{0, \ldots, I-1\}$, $j \in \{0, J-1\}$.
- $d^{xy}_{i,j}$ pre $i \in \{0, I-1\}$, $j \in \{0, J-1\}$.

Zvyšné derivácie d^x, d^y a d^{xy} vieme jednoznačne vypočítať pomocou $2(I) + J + 5$ lineárnych sústav s celkovo $3IJ + I + J + 2$ rovnicami: Nižšie uvádzame modelové rovnice, pomocou ktorých sú zostrojené tieto sústavy lineárnych rovníc. Označme I_l a J_l indexy po ktoré budeme iterovať. Platí:

$$I_l = \begin{cases} I-2 & \text{ak } I \text{ je nepárne,} \\ I-3 & \text{ak } I \text{ je párne,} \end{cases}$$

$$J_l = \begin{cases} J-2 & \text{ak } J \text{ je nepárne,} \\ J-3 & \text{ak } J \text{ je párne,} \end{cases}$$

Pre $j \in \{0, \ldots, J-1\}$, teda pre každý stĺpec j vypočítame parciálne derivácie d^x

$$d^x_{i+2,j} - 14d^x_{i,j} + d^x_{i-2,j} = \frac{3}{h_u}(z_{i+2,j} - z_{i-2,j}) - \frac{12}{h_u}(z_{i+1,j} - z_{i-1,j}), \quad i \in \{2, 4 \ldots, I_l\} \tag{1.12}$$

Rovnica je podobná ako plnom algoritme 1.7. Všimnime si, že sústavu rovníc teraz budujeme len pre párne indexy i, teda vyriešením tejto sústavy získame iba polovicu žiadaných hodnôt d^x. Pre $i \in \{1, 3 \ldots, I_l\}$ a $j \in \{0, \ldots, J-1\}$ zvyšné derivácie d^x vypočítame ako

$$d^x_{i,j} = \frac{3}{4h_u}(z_{i+1,j} - z_{i-1,j}) - \frac{1}{4}(d^x_{i+1,j} - d^x_{i-1,j}) \tag{1.13}$$

Pre $i \in \{0, \ldots, I-1\}$, teda pre každý riadok i analogicky vypočítame parciálne derivácie d^y

$$d^y_{i,j+2} - 14d^x_{i,j} + d^x_{i,j-2} = \frac{3}{h_v}(z_{i,j+2} - z_{i,j+2}) - \frac{12}{h_v}(z_{i,j+1} - z_{i,j-1}), \quad i \in \{2, 4 \ldots, I_l\} \tag{1.14}$$

Následne analogicky pre $i \in \{1, 2, \ldots, I-1\}$ a $j \in \{1, 3, \ldots, J_l\}$ zvyšné derivácie d^y vypočítame ako

$$d^y_{i,j} = \frac{3}{4h_v}(z_{i,j+1} - z_{i,j-1}) - \frac{1}{4}(d^y_{i,j+1} - d^y_{i,j-1}) \tag{1.15}$$

Pre $j \in \{0, J-1\}$, teda pre prvý a posledný stĺpec vypočítame parciálne derivácie $d^{x,y}$ rovnako ako pri plnom algoritme.

$$\begin{aligned} d^{xy}_{i+1,j} + 4d^{xy}_{i,j} + d^{xy}_{i-1,j} = \frac{3}{h_u}(d^y_{i+1,j} - d^y_{i-1,j}), \\ i \in \{1, \ldots, I-2\} \end{aligned} \tag{1.16}$$

Pre $i \in \{0, I-1\}$, teda pre prvý a posledný riadok analogicky vypočítame parciálne derivácie $d^{x,y}$

$$\begin{aligned} d^{xy}_{i,j+1} + 4d^{xy}_{i,j} + d^{xy}_{i,j-1} = \frac{3}{h_v}(d^x_{i,j-1} - d^x_{i-1,j}), \\ j \in \{1, \ldots, J-2\} \end{aligned} \tag{1.17}$$

Pre $i \in \{2, 4, \ldots, I_l\}$, teda pre každý stĺpec i dopočítame parciálne derivácie $d^{x,y}$

$$\begin{aligned} & d^{xy}_{i,j+2} + 4d^{xy}_{i,j} + d^{xy}_{i,j-2} = \\ & \frac{1}{7}(d^{xy}_{i-2,j+2} - d^{xy}_{i-2,j-2}) - 2d^{xy}_{i-2,j} \\ & + \frac{3}{7h_u}(d^y_{i-2,j+2} - d^y_{i-2,j-2}) + \frac{3}{7h_v}(-d^x_{i-2,j+2} - d^x_{i-2,j-2}) \\ & + \frac{9}{7h_u}(d^y_{i,j+2} - d^y_{i,j-2}) + \frac{9}{7h_uh_v}(-z_{i-2,j+2} + z_{i-2,j-2}) \\ & + \frac{12}{7h_u}(-d^y_{i-1,j+2} - d^y_{i-1,j-2}) + \frac{12}{7h_v}(d^x_{i-2,j+1} - d^x_{i-2,j-1}) \\ & + \frac{3}{7h_v}(d^x_{i,j+2} - d^x_{i,j-2}) + \frac{27}{7h_uh_v}(-z_{i,j+2} + z_{i,j-2}) \\ & + \frac{36}{7h_uh_v}(z_{i-1,j+2} - z_{i-1,j-2} + z_{i-2,j+1} - z_{i-,j-1}) \\ & - \frac{6}{h_u}d^y_{i-2,j} + \frac{144}{7h_uh_v}(-z_{i-1,j+1} + z_{i-1,j-1}) + \frac{24}{h_u}d^y_{i-1,j}, \\ & j \in \{4, 6 \ldots, J_l - 2\} \end{aligned} \tag{1.18}$$

Následne vypočítame zvyšné derivácie $d^{x,y}$. Najprv pre $i \in \{1, 3, \ldots, I_l\}$ a

$j \in \{1, 3, \ldots, J_l\}$ platí

$$\begin{aligned} d^{xy}_{i,j} = &\frac{1}{16}(d^{xy}_{i+1,j+1} + d^{xy}_{i+1,j-1} + d^{xy}_{i-1,j+1} + d^{xy}_{i-1,j-1}) \\ &- \frac{3}{16h_v}(d^{x}_{i+1,j+1} - d^{x}_{i+1,j-1} + d^{x}_{i-1,j+1} - d^{x}_{i-1,j-1}) \\ &- \frac{3}{16h_u}(d^{y}_{i+1,j+1} + d^{y}_{i+1,j-1} - d^{y}_{i-1,j+1} - d^{y}_{i-1,j-1}) \\ &+ \frac{9}{16h_uh_v}(z_{i+1,j+1} - z_{i+1,j-1} - z_{i-1,j+1} + z_{i-1,j-1}). \end{aligned} \tag{1.19}$$

Nakoniec pre $i \in \{1, 3, \ldots, I_l + 1\}$ a $j \in \{2, 4, \ldots, J_l\}$

$$d^{xy}_{i,j} = \frac{3}{4h_v}(d^{x}_{i,j+1} - zd^{x}_{i,j-1}) - \frac{1}{4}(d^{xy}_{i,j+1} - d^{xy}_{i,j-1}) \tag{1.20}$$

a pre $i \in \{2, 4, \ldots, I_l\}$ a $j \in \{1, 3, \ldots, J_l + 1\}$

$$d^{xy}_{i,j} = \frac{3}{4h_v}(d^{x}_{i,j+1} - zd^{x}_{i,j-1}) - \frac{1}{4}(d^{xy}_{i,j+1} - d^{xy}_{i,j-1}) \tag{1.21}$$

Označenie 1.8 Zaveďme dve označenia.

- Rovnice 1.13, 1.15 budeme súhrne označovať pojmom *resty*.
- Rovnice 1.19, 1.20, 1.21 budeme súhrne označovať pojmom *zmiešané resty*.

Modelové sústavy rovníc 1.12 a 1.14 majú takýto maticový tvar

$$\begin{pmatrix} -14 & 1 & 0 & \cdots & 0 & 0 \\ 1 & -14 & 1 & \cdots & 0 & 0 \\ 0 & 1 & -14 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -14 & 1 \\ 0 & 0 & 0 & \cdots & 1 & \mu \end{pmatrix} \cdot \begin{pmatrix} D_2 \\ D_4 \\ D_6 \\ \vdots \\ D_{v-2} \\ D_v \end{pmatrix} = \begin{pmatrix} \frac{3}{h}(Y_4 - Y_0) - \frac{12}{h}(Y_3 - Y_1) - D_0 \\ \frac{3}{h}(Y_6 - Y_2) - \frac{12}{h}(Y_5 - Y_3) \\ \frac{3}{h}(Y_8 - Y_4) - \frac{12}{h}(Y_7 - Y_5) \\ \vdots \\ \frac{3}{h}(Y_\nu - Y_{\nu-4}) - \frac{12}{h}(Y_{\nu-3} - Y_{\nu-5}) \\ \frac{3}{h}(Y_{\nu+\tau} - Y_{\nu-2}) - \frac{12}{h}(Y_{\nu-1} - Y_{\nu-3} - \theta D_{K+1}) \end{pmatrix}, \tag{1.22}$$

kde

$$\begin{aligned} &\mu = -15,\ \tau = 0,\ \theta = -4, \text{ a } \nu = N, && \text{ak } K \text{ je párne}, \\ &\mu = -14,\ \tau = 2,\ \theta = 1, \text{ a } \nu = N - 1, && \text{ak } N \text{ je nepárne}, \end{aligned} \tag{1.23}$$

a podľa toho o ktorú z modelových rovníc sa jedná, hodnoty N, D a Y zadávame následovne. Nech k z $1, \ldots, K-1$. Potom

- $N = I$, $h = h_u$, $D_k = d^x_{k,j}$ a $Y_k = z_{k,j}$, pre rovnicu 1.12..
- $N = J$, $h = h_v$, $D_k = d^y_{i,k}$ a $Y_k = z_{i,k}$, pre rovnicu 1.14.

Analogicky vieme zostrojiť maticový tvar aj pre modelovú sústavu pre derivácie d^{xy} podľa 1.18.

Označenie 1.9 Špeciálne zaveďme označenia pre tieto hodnoty:

- Parciálne derivácie d^x a d^y počítané rovnicami 1.13 a 1.15 budeme nazývať *zostatkové derivácie.*
- Zmiešané parciálne derivácie $d^x y$ počítané rovnicami 1.19, 1.20 a 1.21 budeme nazývať *zmiešané zostatkové derivácie.*

2 Inštrukčný paralelizmus

Predtým ako začneme s počítaním operácií je nutné objasniť si ako funguje aplikovanie výpočtov na moderných procesoroch. Architektúry CPU prešli za posledné desaťročia značným vývojom. V dnešnej dobe už nemožno zvyšovať výpočtovú rýchlosť hrubou silou zvyšovaním frekvencie. Moderné procesorové mikroarchitektúry často používajú rôzne optimalizačné „triky a finty“ ako dosiahnuť zlepšenie výkonu a ktoré majú značný vplyv na reálne rýchlosti programov a algoritmov.

V posledných rokoch sa v počítačovej vede stále viac spomína pojem paralelizmu. Dnešné procesory architektúry x86 využívajú až štyri úrovne paralelizácie výpočtov. Na najvyššej úrovni hovoríme o návrhu kedy je procesor zložený z niekoľkých autonómne pracujúcich „podprocesorov“ nazývaných *jadrá.* Tie zdieľajú systémové zbernice a pamäť pričom majú väčšinou vlastnú L1 alebo L2 cache. Každé jadro môže spracovávať na sebe nezávislé procesy prípadne jeden proces môže byť rozdelený do takzvaných vlákien, kde každé môže byť spracovávané iným jadrom. V tomto prípade hovoríme o takzvanom *vláknovom paralelizme* s ktorým sa stretávame najmä pri programovaní vo vyšších jazykoch ako sú C, Java, Haskell a iné.

Poznámka 2.10 V našej implementácii testujeme vláknovo paralelizované aj klasické sériové verzie algoritmov. V rámci tohto článku budeme uvažovať len sériovú implementáciu.

Ďalšie úrovne paralelizácie majú spoločné označenie *inštrukčný paralelizmus*. Pod týmto pojmom rozumieme *superskalárnosť*, *pipelining* a *vektorizáciu*. Pri tejto úrovni paralelizmu má programátor len obmedzené možnosti jeho ovplyvnenia, všetku „ťažkú“ prácu obstará prekladač a pri behu aplikácie zasa inštrukčný plánovač v samotnom procesorovom jadre.

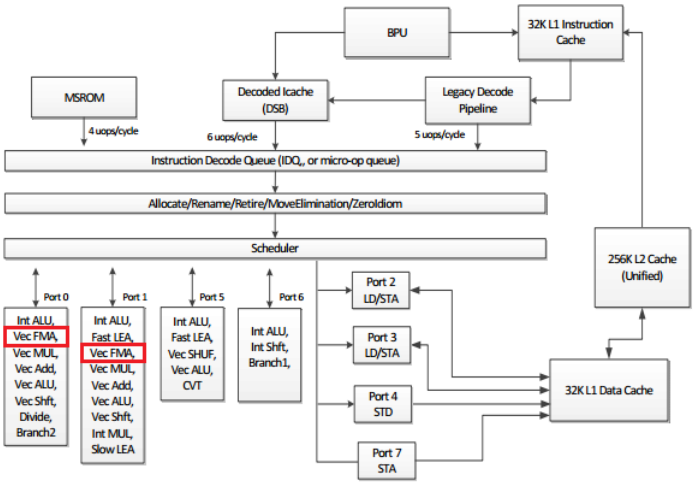
Hlavný vplyv na výkon plného a najmä redukovaného algoritmu má superskalárnosť. Vysvetlime čo tento pojem znamená. Jadro je tvorené niekoľkými až desiatkami špecializovanými jednotkami ako sú aritmeticko logické jednotky (ALU), bitové posuvníky (Shift), numerické koprocesory (FPU) a podobne. Narozdiel od celých jadier, tieto jednotky vo všeobecnosti nedokážu fungovať samostatne a simultánne spracovávať viacero vlákien alebo procesov[1]. Vedia iba v rámci jedného vlákna, za splnenia určitých podmienok, spracovávať niekoľko inštrukcií naraz. Nutnou podmienkou je napríklad vzájomná nezávislosť niekoľkých po sebe idúcich inštrukcií, teda keď výsledok jednej inštrukcie nezávisí na výsledku predchádzajúcej.

Tu ale paralelizácia nekončí. Aj samotné jednotky totiž dokážu simultánne spracovávať viac než jeden dátový vstup. Na tejto úrovni rozlišujeme medzi vektorizáciou, a pipeliningom. Prvá menovaná technika umožňuje jednu inštrukciu aplikovať na celé vektory, resp. polia. Vektorizáciu vieme použiť ak vykonávame operácie na vektore tak, že výpočet i-teho prvku nezávisí na výsledku výpočtu (napr.) $(i-1)$-teho prvku. Pri LU dekompozícii môžeme vidieť, že toto neplatí. Vektorizácia nás teda nemusí zaujímať.

Pipelining je založený na myšlienke princípu fungovania výpočtových jednotiek podobne ako výrobná linka vo fabrike, kde nový výrobok na linku vstúpi skôr ako je predchádzajúci dokončený [8]. Teda aj väčšina výpočtových jednotiek dokáže s novým výpočtom začať ešte pred dokončením práve prebiehajúcej operácie. Táto technika sa prejavuje najviac ak máme veľký počet operácií rovnakého typu, ktoré sú navyše na sebe nezávislé podobne ako pri vektorizácii. To opäť nie je prípad našich algoritmov takže ho zanedbáme.

Poznámka 2.11 Aby som neuviedol čitateľa v omyl, svoje závery ohľadom vektorizácie a pipeliningu upresním. Procesor pri vykonávaní algoritmu robí mnoho operácií nevýpočtového charakteru, ktoré sú programátorovi skryté a kde sa môžu (nielen) tieto techniky prejaviť. Ale aj po zanedbaní dostávame korešpondujúce výsledky meraní a vypočítaného zrýchlenia, ktoré sa líšia najviac o jednu desatinu. Uvažovaním vplyvu týchto techník pri počítaní zrýchlenia by sa dosiahnutý cieľ a síce vysvetlenie príčin urýchlenia redukovaného algoritmu nezmenil. Teda vzhľadom na množstvo potenciálne investovaného času by nezanedbanie týchto techník bolo kontraproduktívne.

[1] Existujú technológie ako napríklad Hyper-Threading umožňujúce za určitých podmienok uplatniť vláknový paralelizmus aj v rámci jediného superskalárneho jadra.



Obr. 2.2: Schéma výpočtového jadra mikroarchitektúry Intel Skylake [6].

Spomenuli sme majoritný vplyv superskalárnosti. Na obrázku 2.2 vidíme schému jedného jadra procesorovej mikroarchitektúry Intel Skylake, ktorá je ku dňu písania práce najmodernejšou bežne dostupnou procesorovou generáciou a na ktorej testujeme zrýchlenie redukovaného algoritmu. Jadrá (nielen) tejto architektúry sú vybavené dvomi sčítačko-násobičkami čísiel s plávajúcou desatinnou čiarkou, ktoré sú na obrázku zvýraznené červenou farbou. Pointa superskalárnosti v tomto prípade spočíva v možnosti výpočty výrazov typu $a_0 \circ a_1 \circ \cdots \circ a_n$, kde $\circ \in \{+, \cdot\}$ rozložiť medzi dve jednotky a dosiahnuť dvojnásobného zrýchlenia. Toto má značný vplyv na rýchlosť redukovaného algoritmu najmä pri počítaní pravých strán trojdiagonálnych rovníc, ktoré sú už na prvý pohľad zložitejšie ako v prípade plného algoritmu. Keďže procesor má ale iba jednu deličku, pre výrazy typu $a_0/a_1/\ldots/a_n$ takýto trik fungovať nebude.

2.1 Rýchlosť aritmetických operácií

Procesory majú mnoho aritmetických jednotiek špecializovaných na určitý typ operácie. Je prirodzené predpokladať, že tieto jednotky budú pracovať navzájom rozličnými rýchlosťami. V prípade výpočtových algoritmov má vplyv na rýchlosť, pochopiteľne okrem výberu vhodných dátových štruktúr, najmä doba vykonania základných matematických operácií a síce sčítania, odčítania, násobenia a delenia.

Najlepším zdrojom ako zistiť rýchlosť týchto operácií je dokumentácia

inštrukčných sád procesorov. Inštrukčná sada x86 sa od svojho prvotného uvedenia v roku 1978 dočkala mnohých rozšírení. Moderné procesory obsahujú niekoľko spôsobov ako napríklad vynásobiť dve čísla. Keďže nie je v silách otestovať všetky možné rozšírenia sady, zvolili sme si jedno konkrétne rozšírenie a síce Streaming SIMD Extensions (skrátene SSE), konkrétne vo verzii 4 (SSE4). Sady z rodiny SSE sú v čase písania práce najpoužívanejšími sadami (najmä verzia SSE2) ktoré sú podporované prakticky všetkými procesormi od roku 2003.

V súčasnosti už existuje modernejšia náhrada tejto sady zvaná Advanced Vector Extensions (skratka AVX), ktorej hlavný prínos spočíva vo vylepšených vektorových operáciach. Na oba algoritmy ale vektorizácia nemá vplyv. Praktické zrýchlenie sme testovali na štyroch procesoroch architektúry x86, pričom sme pokryli väčšinu mikroarchitektúr v rozmedzí rokoch 2007 až 2015.

V nasledujúcej tabuľke 2.1 uvidíme rýchlosti štyroch základných matematických operácií v rámci šiestich testovaných mikroarchitektúr. Tabuľka obashuje tieto stĺpce.

- **Architektúra (rok)**
  Testovaná mikroarchitektúra a rok jej uvedenia na trh. Architektúry sú zoradené abecedne podľa výrobcu a následne podľa roku vydania.

- **Odozva**
  Počet strojových cyklov potrebných na vykonanie inštrukcie.

- **Inverzný prietok**
  Počet strojových cyklov ktoré je nutné čakať kým je daná výpočtová jednotka schopná zopakovať inštrukciu. V prípade operácií sčítania, odčítania a násobenia je tento počet menší ako odozva. To znamená, že aritmetické sčítačky a násobičky sú schopné, vďaka technike pipeliningu, začať nový výpočet ešte pred dokončením aktuálnehu výpočtu.

Podľa tabuľky vidno, že operácie sčítania a odčítania sú rovnako rýchle čo sa pochopiteľne dalo očakávať. Tieto dve operácie preto budeme spoločne označovať symbolom $\pm$. Od tejto chvíle ak spomenieme operáciu sčítania tak tým budeme súčasne myslieť aj operáciu odčítania. Ako vidieť zďaleka najpomalšie je práve delenie.

Pre zaujímavosť si v tabuľke 2.2 nižšie ukážme praktické výsledky z testovacej aplikácie. Operácie sme merali o na 512 prvkovom poli, pričom aby sme dostali „rozumne“ dlhé časy výpočty boli opakované 500000 krát. Operácie boli v tvare $a[i] = a[i] \circ a[i-1]$, kde $\circ \in \{+, \cdot, \div\}$. V tabuľke 2.2 sú namiesto

[1] Odozva je nadobúda menšie hodnoty ak je deliteľ celý.

Architektúra (rok)	Odozva +	Odozva −	Odozva ×	Odozva ÷	Inverzný prietok +	Inverzný prietok −	Inverzný prietok ×	Inverzný prietok ÷
AMD Piledriver (2012)	5-6	5-6	5-6	9-27	0,5	0,5	0,5	5-10
Intel Penryn (2007)	3	3	5	6-21[1]	1	1	1	5-20[1]
Intel Sandy Bridge (2011)	3	3	5	10-22	1	1	1	10-22
Intel Skylake (2015)	4	4	4	13-14	0,5	0,5	0,5	4

Tabuľka 2.1: Tabuľka aritmetických operácií na rôznych mikroarchitektúrach podľa [7]. Údaje predstavujú počet strojových cyklov.

Procesor	±	×	÷
AMD FX-6300	247	236	445
Intel Core 2 Duo E8200	204	261	417
Intel Core i3 2350M	261	362	894
Intel Core i7 6700K	77	80	147

Tabuľka 2.2: Rýchlosť aritmetických operácií na konkrétnych CPU. Údaje sú v milisekundách.

mikroarchitektúr uvedené konkrétne modely procesorov, pričom ich poradie zodpovedá poradiu z minulej tabuľky.

V praxi vidno, že sčítavanie a násobenie môžeme považovať za podobne rýchle operácie. Je nutné podotknúť, že pomery rýchlosti operácií sú závislé od povahy testovania. My sme doby trvania aritmetických operandov $\circ \in \{+, \cdot, \div\}$ merali na jednom vektore, pričom testy boli v tvare $a[i] = a[i] \circ a[i-1]$, teda výpočet i-teho prvku závisel od výsledku výpočtu predchádzajúceho $(i-1)$-teho prvku. Týmto sme sa snažili povahu testu čo najviac napodobniť tvaru výpočtov v interpolačných algoritmoch.

Poznámka 2.12 V prípade, ak by test bol v tvare $a[i] = b[i] \circ c[i]$, tak výpočty pre jednotlivé i by boli na sebe nezávislé. To by procesoru umožnilo použiť techniku pipeliningu, pričom pomery časov násobenia a sčítania by vyšli, vďaka väčšiemu prietoku týchto dvoch operácií oproti deleniu, násobne väčšie. Jednak by takýto test nesúhlasil s tvarom výpočtov v testovaných algoritmoch a taktiež by nekorešpondovalo teoretické zrýchlenie s meraným zrýchlením.

3 Teoretické zrýchlenie

V tejto časti si spočítame počty operácií procedúr tvoriacich plný a redukovaný algoritmus, ktorých časová zložitosť je rovnaká a síce $O(I \cdot J)$. Pri určení asymptotickej časovej zložitosti zvyčajne zanedbávame rýchlosti jednotlivých krokov algoritmu ako sú napríklad aritmetické operácie, porovnávanie veľkosti čísel, kopírovanie a podobne. Keď porovnávame rýchlosť asymptoticky rovnako rýchlych postupoch musíme brať do úvahy vplyvy jednotlivých týchto elementárnych krokov. Na základe poznatkov z časti o procesorovej architektúre si formálne zadefinujme pojmy pre počty a ceny aritmetických operácií ktoré budeme v nasledujúcich častiach.

Označenie 2.13 Množinu všetkých matematických výrazov budeme označíme symbolom $\mathbb{V}$.

Zadefinujme si funkciu o ktorá vráti počet aritmetických operácií sčítania.

Definícia 2.14 Nech $\mathcal{V}$ je matematický výraz obsahujúci $p^{\pm}$ sčítaní a odčítaní, $p^{\times}$ násobení a $p^{\div}$ delení. Definujme funkciu $o : \mathbb{V} \to \mathbb{N}^3$ vzťahom

$$o(\mathcal{V}) = \langle p^{\pm}, p^{\times}, p^{\div} \rangle.$$

Funkciu o budeme nazývať *počet operácií výrazu* $\mathcal{V}$.

Označenie 2.15 Podľa časti 2 o inštrukčnom paralelizme položme dve premenné, ktoré nám pomôžu definovať ceny aritmetických operácií.

- Hodnota β značí *faktor inštrukčného paralelizmu* operácií s plávajúcou desatinnou čiarkou. Inými slovami hodnota β značí počet jednotiek jadra procesora schopných počítať desatinné čísla.
- Hodnota γ značí *pomer odozvy delenia a sčítania* pri operáciách s plávajúcou desatinnou čiarkou. Inými slovami hodnota γ značí koľkokrát je delenie pomalšie oproti sčítaniu.

Poznámka 2.16 Pre moderné procesory architektúry x86 budeme uvažovať $\beta = 2$ a $\gamma = 3$.

Poznámka 2.17 Faktor inštrukčného paralelizmu β budeme uvažovať iba pre operácie sčítania a násobenia. Žiadny bežne dostupný procesor totiž nedokáže inštrukčne paralelizovať delenie.

Teraz si definujme funkciu c ktorá vráti cenu aritmetických operácií pre nejakú matematickú operáciu berúc do úvahy hodnoty β a γ.

Definícia 2.18 Nech $p^{\pm}$ je počet sčítaní a odčítaní, $p^{\times}$ počet násobení a $p^{\div}$ počet delení. Definujme funkcie cien operácií

- $c : \mathbb{N}^3 \to \mathbb{N}^3$ vzťahom

$$c(\langle p^{\pm}, p^{\times}, p^{\div} \rangle) = \left\langle \left\lceil \frac{p^{\pm}}{\beta} \right\rceil, \left\lceil \frac{p^{\times}}{\beta} \right\rceil, \lceil \gamma p^{\div} \rceil \right\rangle,$$

  ktorú nazveme *cena operácií*.

- Nech V je matematický výraz a $o(V) = \langle p^{\pm}, p^{\times}, p^{\div} \rangle$. Potom *cena operácií výrazu* V je funkcia $c : \mathbb{V} \to \mathbb{N}^5 \times \mathbb{N}$ v tvare

$$c(\mathcal{V}) = c(o(\mathcal{V})).$$

V kontexte počítania operácií a cien budeme z dôvodu zjednodušenia procedúry považovať za množiny matematických výrazov, pričom algoritmy budeme považovať za množiny obsahujúce procedúry a výrazy. Stručne doplňme počty aj pre procedúry, resp. algoritmy.

Označenie 2.19 Nech $\mathcal{P}$ je procedúra. Funkciu

$$o(\mathcal{P}) = \sum_{v \in \mathcal{P}} o(v)$$

nazveme *počet operácií procedúry* $\mathcal{P}$.

Analogicky označme aj ceny procedúr, resp. algoritmov.

Označenie 2.20 Nech $\mathcal{P}$ je procedúra. Funkciu

$$c(\mathcal{P}) = \sum_{v \in \mathcal{P}} c(v)$$

nazveme *cena operácií procedúry* $\mathcal{P}$.

Dôvod, prečo je náš redukovaný algoritmus rýchlejší je práve fakt, že pri ňom dochádza k značne menšiemu počtu delení, ktoré je oproti ostatným trom operáciám výrazne pomalšie. Navyše v prípade redukovaného algoritmu sa prejavuje superskalárnosť procesora (najmä) pri príprave pravých strán rovníc pre LU dekompozíciu ako podľa 1.12, 1.14, 1.16, 1.17, 1.18 v časti 1.3 o počítaní derivácií redukovaným spôsobom. V ďalšej sekcii sy spočítame jednotlivé operácie a vypočítame teoretické zrýchlenie redukovaného algoritmu. To budeme počítať tak, že si zadefinujeme „procedúry“ ktoré predstavujú jednotlivé časti algoritmu podľa častí 1.1 o trojdiagonálnej LU dekompozícii, 1.2 o plnom algoritme a 1.3 o redukovanom algoritme.

Najprv položme procedúry predstavujúce LU dekompozíciu spoločné pre oba postupy.

- Procedúra *InicalizujLU* inicializuje hodnoty pravej strany sústavy rovníc $r_0, \ldots, r_{K-1}$ a hodnotu b z LU dekompozície podľa rovnice 1.11 v prípade plného algoritmu, respektíve podľa rovnice 1.22 v prípade redukovaného algoritmu.
- Procedúra *VyriešLU* vypočíta sústavu rovníc na základe hodnôt pravej strany poskytnutými *InicalizujLU*.

Každý algoritmus inicializuje pravé strany inak, takže pre oba si ceny uvedieme osobitne.

3.1 Cena plného algoritmu

Ceny základných aritmetických a pamäťových operácií pre vyššie uvedené procedúry môžeme zhrnúť do tabuľky 2.3, pričom uvažujeme počty v našej ukážkovej implementácii[2]. Riadky tabuľky predstavujú jednotlivé procedúry a stĺpce udávajú počet vykonaných operácii, kde K je počet neznámych. Podotýkam, že operáciu odčítania budeme uvažovať ako sčítanie. V tabuľkách budeme pre sčítanie, násobenie a delenie uvádzať ich ceny podľa definície 2.18.

[2]Repozitár so zdrojovými kódmi k aplikáciám možno nájsť na adrese https://github.com/vildibald/VKDiplom-master

Procedúra	±	×	÷
InicalizujLU	K	K	0
VyriešLU	$3K$	$2K$	γK

Tabuľka 2.3: Ceny operácií LU dekompozície pre plný algoritmus vzhľadom na počet uzlov $I \cdot J$.

Procedúry *VyriešLU* a *InicalizujLU* neobsahujú matematické výrazy s viac ako jedným sčítaním prípadne násobením. Superskalárnosť procesora sa teda v prípade plného algoritmu neprejaví. Uvažujme procedúry predstavujúce implementáciu plného algoritmu podľa časti 1.2:

- Procedúra *VypočítajDx* vypočíta parciálne derivácie d^x pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.7. Jedno volanie *VyriešLU* vypočíta derivácie d^x pre jeden stĺpec. Teda procedúra musí počítať LU pre každý stĺpec, ktorých je J.

- Procedúra *VypočítajDxy* vypočíta parciálne derivácie d^{xy} pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.8. Jedno volanie *VyriešLU* vypočíta derivácie d^{xy} pre jeden stĺpec. Procedúra musí počítať LU pre prvý a posledný stĺpec.

- Procedúra *VypočítajDy* vypočíta parciálne derivácie d^y pomocou procedúry procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.9. Jedno volanie *VyriešLU* vypočíta derivácie d^y pre jeden riadok. Teda procedúra musí počítať LU pre každý riadok, ktorých je I.

- Procedúra *VypočítajDyx* vypočíta parciálne derivácie d^{xy} pomocou procedúry procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.10. Jedno volanie *VyriešLU* vypočíta derivácie d^{xy} pre jeden riadok. Teda procedúra musí počítať LU pre každý riadok, ktorých je I.

- Procedúra *VypočítajPlný* vypočíta na základe vstupných hodnôt pre de Boorovu interpoláciu 1.2 zavolaním procedúr *VypočítajDx*, *VypočítajDxy*, *VypočítajDy* a *VypočítajDyx*.

Nech I značí počet uzlov na osi x a J značí počet uzlov na osi y. Všetky ceny sú v tvare $a \cdot IJ + b \cdot I + c \cdot J + d$. Pre zjednodušenie budeme ceny, tam kde je koeficient a nenulový, uvádzať v tvare $a \cdot IJ$. Keďže ceny algoritmov rastú kvadraticky tak pre veľké I a J bude odchýlka zanedbateľná. Pre plný algoritmus teda dostaneme tieto počty

Procedúra	±	×	÷
VypočítajDx	$4IJ$	$3IJ$	γIJ
VypočítajDxy	$8I$	$6I$	$2\gamma I$
VypočítajDy	$4IJ$	$3IJ$	γIJ
VypočítajDyx	$4IJ$	$3IJ$	γIJ

Tabuľka 2.4: Ceny operácií plného algoritmu.

Sčítaním všetkých cien operácií v tabuľke dostaneme následujúci výsledok.

Lemma 2.21 *Nech I a J označujú počty uzlov na osiach x a y. Potom sumárna cena operácií plného algoritmu je*

- $12IJ$ *sčítaní,*
- $9IJ$ *násobení,*
- $3\gamma IJ$ *delení.*

Cena plného algoritmu teda je

$$21IJ + 3\gamma IJ.$$

3.2 Cena redukovaného algoritmu

Redukovaný spôsob na počítanie derivácií využíva iný tvar trojdiagonálej sústavy rovníc. Zmeníme teda počty operácií procedúr *VyriešLU* a *InicalizujLU*. Ďalej položme dve nové pomocné procedúry *InicalizujZmiešLU*, ktorá inicializuje hodnoty $r_0, \ldots, r_{K-1}$ a b z LU dekompozície 1.4 pre zmiešané derivácie d^{xy} a k nej príslušnú *VyriešZmiešLU*.

Procedúra	±	×	÷
InicalizujLU	${}^{3}/_{\beta}K$	${}^{2}/_{\beta}K$	0
VyriešLU	$3K$	$2K$	γK

Tabuľka 2.5: Ceny operácií LU dekompozície pre redukovaný algoritmus.

Procedúra	±	×	÷	$\vdots\cdot$
InicalizujZmiešLU	${}^{33}/_{2}K$	${}^{17}/_{2}K$	0	0
VyriešZmiešLU	$3K$	$2K$	γK	$\gamma^{\vdots\cdot}K$

Tabuľka 2.6: Ceny operácií LU dekompozície na spojitých dátach pre zmiešané zostatkové derivácie.

V prípade inicializovania pravej strany pre LU dekompozíciu v procedúre *InicalizujLU* dochádza k viacerým sčítaniam a/alebo násobeniam v rámci jedného výrazu. Tu sa následne prejaví superskalárnosť výpočtových jednotiek procesorového jadra podľa časti 2. Moderné procesory architektúry x86 obsahujú práve dve jednotky pre výpočty s pohyblivou desatinnou čiarkou čo znamená faktor inštrukčného paralelizmu $\beta = 2$. To implikuje zaujímavý dôsledok, kedy algoritmus s väčším počtom matematických operácií je v praxi rýchlejší ako algoritmus s menším celkovým počtom operácií, ale s viacerými na sebe závislými výrazmi (t.j. prípad, keď výraz b musí byť vyhodnotený až po výraze a).

V prípade zmiešaných derivácií používame odlišné pravé strany rovníc pre LU dekompozíciu. Osobitne si v tabuľke 2.6 spočítajme ceny aj pre tieto procedúry. Následne analogicky ako v predchádzajúcej sekcii položme procedúry predstavujúce implementáciu redukovaného algoritmu podľa časti 1.3::

- Procedúra *VypočítajDxResty* vypočíta zvyšné parciálne derivácie d^x podľa 1.13.

- Procedúra *VypočítajDx* vypočíta parciálne derivácie d^x pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.12. Jedno volanie *VyriešLU* vypočíta derivácie na párnych riadkoch d^x pre jeden stĺpec. Teda procedúra musí počítať LU pre každý stĺpec, ktorých je J. Zvyšné derivácie dopočítame procedúrou *VypočítajResty*.

- Procedúra *VypočítajDyResty* vypočíta zvyšné parciálne derivácie d^y podľa 1.15.

Procedúra	±	×	÷	∶·
VypočítajDx	$^3/_2IJ + {}^3/_{2\beta}IJ$	$1IJ + {}^1/_{\beta}IJ$	$^1/_2\gamma IJ$	$^1/_2\gamma^{\vdots\cdot}IJ$
VypočítajDy	$^3/_2IJ + {}^3/_{2\beta}IJ$	$1IJ + {}^1/_{\beta}IJ$	$^1/_2\gamma IJ$	$^1/_2\gamma^{\vdots\cdot}IJ$
VypočítajDxy	$8I + 8J$	$6I + 6J$	$2I + 2J$	$\gamma^{\succ}I + \gamma^{\succ}J$
VypočítajDyx	$^3/_4IJ + {}^{33}/_{4\beta}IJ$	$^1/_2IJ + {}^{17}/_{4\beta}IJ$	$^1/_4\gamma IJ$	$^1/_2\gamma^{\vdots\cdot}$

Tabuľka 2.7: Ceny operácií na spojitých dátach redukovaného algoritmu.

- Procedúra *VypočítajDy* vypočíta parciálne derivácie d^y pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.14. Jedno volanie *VyriešLU* vypočíta derivácie na párnych stĺpcoch d^y pre jeden riadok. Teda procedúra musí počítať LU pre každý riadok, ktorých je I. Zvyšné derivácie dopočítame procedúrou *VypočítajResty*.

- Procedúra *VypočítajDxy* vypočíta parciálne derivácie d^{xy} pomocou procedúr *InicalizujLU* a *VyriešLU*. Vstupné hodnoty vezmeme z 1.16 a 1.17. Jedno volanie *VyriešLU* vypočíta na párnych riadkoch(stĺpcoch) derivácie d^{xy} pre jeden stĺpec(riadok). Procedúra musí počítať LU pre prvý a posledný stĺpec a pre prvý a posledný riadok.

- Procedúra *VypočítajZmiešResty* vypočíta zvyšné parciálne derivácie d^{xy} podľa 1.19, 1.20 a 1.21.

- Procedúra *VypočítajDyx* vypočíta parciálne derivácie d^{xy} pomocou procedúr *InicalizujZmiešLU* a *VyriešZmiešLU*. Vstupné hodnoty vezmeme z 1.18, pričom na ich inicializovanie do LU dekompozície použijeme práve procedúru *InicalizujZmiešLU*. Jedno volanie *VyriešLU* vypočíta na párnych stĺpcoch d^{xy} pre jeden riadok. Procedúra musí počítať LU pre každý párny riadok, ktorých je I. Zvyšné derivácie dopočítame procedúrou *VypočítajZmiešResty*.

- Procedúra *VypočítajRedukovaný* vypočíta na základe vstupných hodnôt pre de Boorovu interpoláciu 1.2 postupným vykonaním predchádzajúcich siedmich procedúr.

Opäť pre zjednodušenie budeme počty v tvare $a \cdot IJ + b \cdot I + c \cdot J + d$ zanedbávať na $a \cdot IJ$. Ak hodnota I značí počet uzlov na osi x a hodnota J značí počet uzlov na osi x tak pre redukovaný algoritmus dostaneme počty v tabuľkách 2.7 a 2.8. Sčítaním operácií v predchádzajúcich dvoch tabuľkách dostaneme následujúci výsledok.

Procedúra	±	×	÷
VypočítajDxResty	${}^{3}/_{2\beta}IJ$	${}^{1}/_{\beta}IJ$	0
VypočítajDyResty	${}^{3}/_{2\beta}IJ$	${}^{1}/_{\beta}IJ$	0
VypočítajZmiešResty	${}^{17}/_{4\beta}IJ$	${}^{7}/_{4\beta}IJ$	0

Tabuľka 2.8: Ceny operácií pre zmiešané zostatkové derivácie.

Lemma 2.22 *Nech I a J označujú počty uzlov na osiach x a y. Potom sumárna cena redukovaného algoritmu je*

- ${}^{15}/_{4}IJ + {}^{37}/_{2\beta}IJ$ *sčítaní,*
- ${}^{5}/_{2}IJ + {}^{10}/_{\beta}IJ$ *násobení,*
- ${}^{5}/_{4}\gamma IJ$ *delení.*

Cena redukovaného algoritmu teda je

$$ {}^{25}/_{4}IJ + {}^{57}/_{2\beta}IJ + {}^{5}/_{4} \cdot \gamma IJ. $$

Redukovaný algoritmus De Boorovej interpolácie obsahuje väčší počet aritmetických operácií ako sú sčítanie a násobenie, pričom obsahuje menší počet delení. V časti 3.3 si ukážeme, že redukovaný algoritmus je po aplikovaní vplyvu inštrukčného paralelizmu a ceny za operácie delenia skutočne rýchlejší. Redukovaný algoritmus by nemal byť lepší len v počte aritmetických operácií, ale taktiež priniesť menšie pamäťové nároky vyplývajúce z polovičnej veľkosti sústav rovníc riešených LU dekompozíciou.

3.3 Zhrnutie

Po spočítaní jednotlivých matematických a pamäťových operácií v častiach 3.1 a 3.2 pristúpime k formulácii zrýchlenia.

Veta 2.23 *Nech I a J označujú počty uzlov na osiach x a y. Potom očakávané zrýchlenie redukovaného algoritmu je*

$$ \frac{21IJ + 3\gamma IJ}{{}^{25}/_{4}IJ + {}^{57}/_{2\beta}IJ + {}^{5}/_{4} \cdot \gamma IJ}. $$

Ukážme si dva príklady zrýchlenia aby sme názorne videli vplyv inštrukčného paralelizmu na rýchlosť redukovaného algoritmu. V prvom príklade predpokladajme porovnanie algoritmov na modernom procesore, pričom v druhom uvažujme primitívny procesor.

Príklad 2.24 Nech $\beta = 2$, teda procesorové jadro má práve dve jednotky pre výpočty s pohyblivou desatinnou čiarkou a $\gamma = 3$. Analogicky ako v predchádzajúcom príklade dosadením do vzťahu dostaneme

$$\frac{30}{24{,}25} \approx 1{,}17.$$

V prípade inštrukčného paralelizmu s faktorom 2 je redukovaný algoritmus o 23% rýchlejší ako plný.

Príklad 2.25 Nech $\beta = 1$, teda procesorové jadro má iba jednu jednotku pre výpočty s pohyblivou desatinnou čiarkou a $\gamma = 3$, to jest operácia delenia je trojnásobne pomalšia ako operácia sčítania podľa tabuliek 2.1 a 2.2 operácií v časti 2.1. Dosadením do vzťahu z predchádzajúcej vety dostaneme

$$\frac{30}{38{,}5} \approx 0{,}78.$$

Pomer je menší ako 1 čo znamená, že bez inštrukčného paralelizmu ($\beta = 1$) je redukovaný algoritmus pomalší ako plný.

Podľa príkladov vidíme, že rýchlosť redukovaného algoritmu je závislá na schopnosti hardvérovej architektúry procesora paralelizovať vyhodnotenia aritmetických výrazov s viacerými operandami. Je otázne nakoľko by s rastúcim faktorom inštrukčného paralelizmu β rástlo zrýchlenie. Keďže nemáme k dispozícii procesor s vhodnou hardvérovou výbavou, ktorý by mal viac ako dve jednotky pre výpočty s pohyblivou desatinnou čiarkou (ak vôbec taký existuje) je náročné predpokladať ako by si takýto stroj s redukovaným postupom poradil.

V nasledujúcej časti si ukážeme reálne výsledky a uvidíme či s nimi vypočítané zrýchlenie skutočne súhlasí.

4 Merané zrýchlenie

V predchádzajúcej časti sme určili teoretické zrýchlenie počítania uzlov, dosiahnuteľné zredukovaním veľkosti trojdiagonálnych sústav. V tejto časti si ukážeme reálne výsledky dosiahnuté v ukážkovej implementácii.

Softvér obsahuje testy pre sekvenčné aj paralelné počítanie derivácií pre oba predmetné algoritmy. Použitý prekladač bol Intel C++ Compiler v 64 bitovej verzii nastavený na generovanie agresívne optimalizovaného binárneho kódu (-O2).

Testy boli vykonané na štyroch rôznych počítačových zostavách, všetky so systémom Windows 7 a 10. Testovacie stroje obsahujú rozličné multivláknové

procesory od starého Penryn z roku 2007 až po najmodernejší Skylake z roku 2015 so vzájomne odlišnými architektúrami a hlavne spôsobmi vykonávania paralelizovaných procesov.

Stĺpec 1 obsahuje modely procesorov zoradených podľa mikroarchitektúry ako v tabuľke 2.1. Stĺpce 2 a 3 predstavujú časy behov sériovej verzie pre povrchové splajny.

Procesor	Plný	Redukovaný
FX-6300	81	72
C2D E8200	99	87
Ci3 2350M	93	75
Ci7 6700K	36	30

Tabuľka 2.9: Reálne merania plného a redukovaného algoritmu na mriežke 1000×1000 uzlov. Údaje sú v milisekundách.

V tabuľke 2.10 budeme v stĺpci 2 uvažovať pomery nameraných rýchlosti delenia a sčítania operácií z tabuľky 2.2 v časti 2.1. Stĺpec 3 predstavuje teoretické zrýchlenie redukovaného algoritmu vypočítané podľa vety 2.23 v predchádzajúcej časti. Posledný stĺpec 4 predstavuje merané zrýchlenie sériovej verzie redukovaného algoritmu podľa predchádzajúcej tabuľky. Ako vidieť, namerané zrýchlenie algoritmu korešponduje s teoretickým zrýchlením v rámci malej odchýlky.

		Zrýchlenie	
Procesor	γ	teoretické	merané
FX-6300	1,89	1,17	1,13
C2D E8200	1,08	1,11	1,14
Ci3 2350M	3,43	1,26	1,24
Ci7 6700K	2,98	1,24	1,2

Tabuľka 2.10: Porovnanie teoretického a meraného zrýchlenia na mriežke 1000×1000 uzlov.

Záver

Podarilo sa nám v prípade rovnomerne rozložených uzlov urýchliť sériový výpočet derivácií splajnov v uzloch. Napriek tomu, že výsledný redukovaný

algoritmus obsahuje viac aritmetických operácií je vďaka povahe mikroarchitektúr moderných procesosov a tiež vďaka povahy samotných operácií rýchlejší o približne 20%. Prioritou teraz ostáva jednak zovšeobecniť redukovaný algoritmus aj pre splajny s nerovnomernými uzlami a upraviť zmiešané zbytkové derivácie aby sme mohli docieliť ďalšie zrýchlenie a najmä zefektívniť vláknovú paralelizáciu. Ďalšou výhodou sú polovičné pamäťové nároky čo umožňuje riešiť väčšie úlohy.

Zoznam použitej literatúry

[1] I. Szabó, L. Miňo, C. Török, Biquartic polynomials in bicubic spline construction, PF UPJŠ, 2014

[2] Lukáš Miňo, Csaba Török, Fast algorith for spline surfaces, PF UPJŠ, 2015

[3] C. de Boor, Bicubic spline interpolation, Journal of Mathematics and Physics, 41(3),1962, 212-218.

[4] https://github.com/vildibald/VKDiplom-master, repozitár so zdrojovými kódmi k aplikáciám.

[5] https://en.wikibooks.org/wiki/Algorithm_Implementation/Linear_Algebra/Tridiagonal_matrix_algorithm

[6] http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[7] http://www.agner.org/optimize/instruction_tables.pdf

[8] http://www.lighterra.com/papers/modernmicroprocessors/