VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

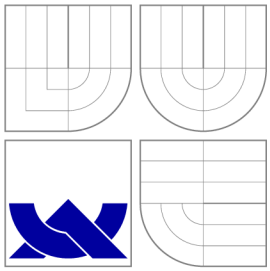
HYPERBOLICKÁ PARCIÁLNÍ DIFERENCIÁLNÍ ROVNICE HOMOGENNÍHO A NEHOMOGENNÍHO VEDENÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

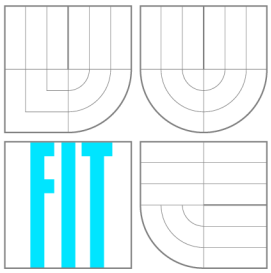
AUTOR PRÁCE
AUTHOR

Bc. ALEXANDR SZÖLLÖS

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

HYPERBOLICKÁ PARCIÁLNÍ DIFERENCIÁLNÍ ROVNICE HOMOGENNÍHO A NEHOMOGENNÍHO VEDENÍ

WAVE PARTIAL DIFFERENTIAL EQUATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ALEXANDR SZÖLLÖS

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. JIŘÍ KUNOVSKÝ, CSc.

BRNO 2009

Zadání práce

1. Seznamte se s jednotlivými typy parciálních diferenciálních rovnic.
2. Zaměřte se na hyperbolickou parciální diferenciální rovnici popisující chování homogenních a nehomogenních elektrických vedení.
3. Seznamte se s programovým prostředím CUDA grafických procesorů (GPU).
4. Vytvořte program pro řešení konkrétního vedení s respektováním různého řádu integrační metody.
5. Navržený program implementujte.
6. Srovnejte se světovými standardy (Matlab, Maple).

Licenční smlouva

Abstrakt

Práce se zabývá diferenciálními rovnicemi, jejich využitím při analýze vedení, experimenty s vedením a možnou akcelerací výpočtu v GPU s využitím prostředí nVidia CUDA.

Klíčová slova

diferenciální rovnice, parciální diferenciální rovnice, homogenní vedení, CUDA, TKSL

Abstract

This work deals with diffrential equations, with the possibility of using them for analysis of the line and the possibility of accelerating the computations in GPU using nVidia CUDA.

Keywords

differential equations, partial differential equations homogenous line, CUDA, TKSL

Citace

Alexandr Szöllös: Hyperbolická parciální diferenciální rovnice homogenního a nehomogenního vedení, diplomová práce, Brno, FIT VUT v Brně, 2009

Hyperbolická parciální diferenciální rovnice homogenního a nehomogenního vedení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Jiřího KUNOVSKÉHO, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Alexandr Szöllös
25. května 2009

Poděkování

Tímto bych chtěl poděkovat panu doc. Ing. Jiřímu Kunovskému, CSc. za vedení této práce, poskytnutí kvalitních materiálů a vytvoření velmi dobré pracovní atmosféry po celou dobu vytváření této práce.

Obsah

Kapitola 1

Úvod

V dnešní době pronikají diferenciální rovnice snad do všech vědních disciplín. Umožňují popis změn ve sledovaných systémech. Jedná se tedy o rovnice, kde se vyskytuje neznámá hledaná funkce a také její derivace. Ukázalo se, že někdy je vhodné derivovat funkce podle více proměnných - vznikly tak parciální diferenciální rovnice. Diferenciálními rovnicemi a jejích aplikací při analýze přechodných dějů na homogenním a nehomogenním vedení, se zabývá tato diplomová práce.

Pro počítače platí Moorův zákon, který říká, že výkon počítače a velikost paměti se každých 18 měsíců zdvojnásobí. Tento zákon jistě platí pro běžné procesory stolních počítačů. Čipy grafických karet však prodělávají o poznání razantnější vývoj směrem k rapidnímu zvyšování výkonu. V současné době se uvádějí do praxe myšlenky paralelizace výpočtů, protože tato oblast slibuje další navýšení výkonnosti. Při zkoumání historického vývoje grafických karet je vidět, že jde vývoj těchto zprvu specializovaných čipů velmi razantně kupředu a v současné době již výpočetní výkon grafické nemusí být nutně využit jen pro grafické výpočty, ale z grafických čipů se staly masivně paralelní zařízení vhodná pro náročné výpočty. Uvážíme-li, že první takovýto grafický čip spatřil světlo světa v roce 2001, je oprávněné tvrdit, že tento vývoj je opravdu pozoruhodný.

Uvážíme-li, že analýza přechodných dějů na homogenním, či nehomogenním vedení představuje relativně složitý proces, je jistě vhodné hledat způsob, jak tento problém dekomponovat na více částí. Ukázalo se tedy, že tato dekompozice je možná a dokonce velmi vhodná. Vedení, které důkladně popisuje kapitola 3 lze rozdělit na určitý počet částí a ty poté analyzovat samostatně a získávat tak nové poznatky. Protože se jedná o analýzu především přechodných dějů, je nezbytné pro tuto analýzu použít diferenciální rovnice, dokonce parciální. Touto problematikou se zabývá kapitola 2.

Analytické řešení diferenciálních rovnic představuje složitý proces, někdy dokonce analytické řešení ani není známo, proto se pro řešení diferenciálních rovnic používají různé numerické metody. Numerické metody jsou však zatíženy určitou chybou výpočtu, která lze v určitých případech snížit při zkrácení kroku výpočtu. Samotný výpočet derivací potřebných pro konkrétní numerickou metodu však rovněž nemusí být triviální, a proto je vhodné použití počítače, avšak i zde může výpočet trvat velmi dlouho. Vzhledem k tomu, že čipy grafických karet dnes představují výkonnostní špičku, je logická snaha směřovat náročné výpočty právě do grafických karet. Historií a vývojem grafických karet, specializovanými prostředími pro realizaci paralelních výpočtů v grafických kartách a zejména prostředím CUDA, se zabývá kapitola 4, návrhem programu implementací kapitola 5. Nedílnou součástí této práce je rovněž srovnání způsobů řešení diferenciálních rovnic se světovými standardy, touto problematikou se zabývá kapitola 6.

Kapitola 2

Diferenciální rovnice

Celá práce staví na teorii diferenciálních rovnic. Proto je vhodné definovat a seznámit se se základními pojmy z oblasti diferenciálních rovnic.

2.1 Obyčejná diferenciální rovnice

Obyčejnou diferenciální rovnici lze definovat jako takovou matematickou rovnici, v níž se vyskytuje neznámá funkce jedné proměnné a její derivace. Úkolem pak zpravidla bývá nalézt všechna řešení takovéto rovnice (pokud existují) nebo najít řešení splňující určité doplňující podmínky. Formálněji lze obyčejnou diferenciální rovnici a související pojmy definovat takto:

Obyčejnou diferenciální rovnicí nazveme takovou diferenciální rovnici, v níž se vyskytuje (či vyskytují) derivace hledané funkce jedné proměnné, obecně lze zapsat obyčejnou diferenciální rovnici následujícím způsobem:

$$F\left(x, y, y', y'', ..., y^{(n)}\right) = 0, \tag{2.1}$$

kde $y = \varphi(x)$ je hledaná funkce (viz [16]).

Příkladem obyčejné diferenciální rovnice může být např. tato rovnice:

$$\frac{\mathrm{d}u(t)}{\mathrm{d}(t)} = u(t) \tag{2.2}$$

s počáteční podmínkou typicky $u(t_0) = u_0$, kde t_0 volíme často $t_0 = 0$.

Dalším důležitým pojmem v teorii (obyčejných, parciálních) diferenciálních rovnic je pojem **Řád diferenciální rovnice**. Řádem diferenciální rovnice nazýváme nejvyšší řád derivace hledané funkce v uvažované diferenciální rovnici.

Diferenciální rovnici nazveme **lineární** tehdy, pokud je tato rovnice lineární vzhledem ke hledané funkci i k její derivaci (případně derivacím). Obecně lze tedy obyčejnou lineární diferenciální rovnici zapsat takto:

$$y^{(n)} + a_{n-1}(x)y^{(n-1)} + \cdots + a_1(x)y' + a_0(x)y = f(x), \tag{2.3}$$

kde n představuje řád diferenciální rovnice, x je nezávislá proměnná, $y^{(k)}$ je k-tá derivace hledané funkce $y(x)$, $a_k(x)$ jsou koeficienty obecně závislé na x a $f(x)$ představuje pravou stranu diferenciální rovnice.

V případě, kdy jsou koeficienty a_k konstanty, pak se jedná o diferenciální rovnici s konstantními koeficienty. Pokud $f(x) = 0$, jedná se o tzv. **homogenní diferenciální rovnici**.

V případě, že je rovnice jiného než výše uvedeného tvaru, pak obecně hovoříme o **nelineární diferenciální rovnici**.

Dále je potřeba si definovat pojem **Řešení diferenciální rovnice**. Řešením diferenciální rovnice nazýváme každou $n-$krát spojitě derivovatelnou funkci na nějakém intervalu I, která vyhovuje dané rovnici, takže po dosazení této funkce do dané rovnice dostaneme na intervalu I identickou rovnost. Dle definice existují tyto druhy řešení diferenciálních rovnic:

1. **Obecným řešením obyčejné diferenciální rovnice** budeme rozumět každou funkci závisející na n obecných parametrech $C_1, \cdots, C_n$ takových, že speciální (přípustnou) volbou $C_1, \cdots, C_n$ lze získat řešení každého počátečního problému.

2. **Partikulární řešení obyčejné diferenciální rovnice** je takové řešení, které obdržíme z obecného řešení pevnou volbou konstant $C_1 \cdots, C_n$.

3. **Výjimečné (singulární) řešení** je řešení obyčejné diferenciální rovnice, které nelze získat z obecného řešení žádnou volbou hodnot $C_1, \cdots, C_b$.

Řešit diferenciální rovnici tedy znamená nalézt všechna její řešení. Pokud nalezneme všechna řešení diferenciální rovnice, považujeme ji za vyřešenou.

Jak již bylo uvedeno dříve u diferenciální rovnice bylo potřeba zvolit určitou **počáteční podmínku** (obecně počáteční podmínky). Počáteční podmínkou rozumíme libovolný, ale pevně daný bod (v případě rovnice 2.2 bod t_0). Pak nalezení řešení diferenciální rovnice vyhovující počáteční podmínce (počátečním podmínkám) nazýváme **počáteční problém**. Název pojmů počáteční podmínka a počáteční problém plyne hlavně z toho, že se nejčastěji volí v bodě, který reprezentuje určitý počátek.

2.2 Parciální diferenciální rovnice

Stejně jako v předchozí kapitole, i zde začnu nejprve definicí parciální diferenciální rovnice.

Parciální diferenciální rovnice je taková diferenciální rovnice, v níž se vyskytují parciální derivace hledané funkce dvou nebo více proměnných. Obecně lze parciální diferenciální rovnice zapsat ve tvaru

$$F\left(x_1, x_2, \cdots, x_n, z, \frac{\partial z}{\partial x_1}, \cdots, \frac{\partial z}{\partial x_n}, \frac{\partial^2 z}{\partial x_1^2}, \frac{\partial^2 z}{\partial x_1 \partial x_2}, \cdots, \frac{\partial^2 z}{\partial x_1 \partial x_n}, \frac{\partial^2 z}{\partial x_2^2}, \cdots, \frac{\partial^k z}{\partial x_n^k}, \cdots\right) = 0, \tag{2.4}$$

kde $z(x_1, x_2, \cdots, x_n)$ je neznámá funkce n proměnných (viz [17]). V tomto případě se jedná o nejobecnější vztah, kterým lze popsat jak lineární, tak nelineární parciální diferenciální rovnice.

Pro řád parciální diferenciální rovnice platí totéž, co platilo v předchozí kapitole, tedy že řádem parciální diferenciální rovnice rozumíme nejvyšší řád derivace hledané funkce v dané parciální diferenciální rovnici.

I v případě parciálních diferenciálních rovnic má smysl uvažovat o jejich linearitě resp. nelinearitě. Parciální diferenciální rovnice je lineární právě tehdy, když je tato rovnice lineární vzhledem ke hledané funkci a jejím derivacím. Speciálně lze tedy vyjádřit lineární parciální diferenciální rovnici prvního řádu, kde neznámou je funkce $u = u(x, y)$ v obecném tvaru

$$a(x, y)\frac{\partial u}{\partial x} + b(x, y)\frac{\partial u}{\partial y} + c(x, y)u = d(x, y), \tag{2.5}$$

kde a, b, c, d jsou funkce dvou proměnných.

Analogicky jako v předchozí kapitole lze rovnici 2.5 nazvat **homogenní** v případě, že pravá strana rovnice, tedy $d(x, y) = 0$ na zvoleném intervalu. V opačném případě se jedná o rovnici **nehomogenní**. Linearitu, resp. nelinearitu a homogennost resp. nehomogennost dané parciální diferenciální rovnice lze zobecnit i na rovnice vyšších řádů.

Řešením parciální diferenciální rovnice v nějaké oblasti $\Omega \in \Re^N$ lze nazvat každou funkci, která má v Ω spojité všechny potřebné parciální derivace a která dosazena zároveň s těmito derivacemi do původní parciální diferenciální rovnice vyhovuje pro všechna $x_1, \cdots, x_n$ této rovnici.

Analogicky k obyčejným diferenciálním rovnicím se zavádějí i v případě parciálních diferenciálních rovnic pojmy počáteční problém a počáteční podmínka.

Pro tuto práci nemá smysl se zabývat parciálními diferenciálními rovnicemi vyššího než druhého řádu. V dalším textu této práce se proto již zaměřím hlavně na tyto parciální diferenciální rovnice.

2.3 Hyperbolická parciální diferenciální rovnice

Nejprve je třeba zdůraznit, že v případě hyperbolické parciální diferenciální rovnice se jedná o parciální diferenciální rovnici druhého řádu. Obecně lze tuto rovnici popsat vzorcem

$$A\frac{\partial^2 z(x,y)}{\partial x^2} + B\frac{\partial^2 z(x,y)}{\partial x \partial y} + C\frac{\partial^2 z(x,y)}{\partial y^2} + D\frac{\partial z(x,y)}{\partial x} + E\frac{\partial z(x,y)}{\partial y} + Fz(x,y) + G = 0, \quad (2.6)$$

kde $A \cdots G \in \Re$ lze chápat jako určité koeficienty, případně jako spojité funkce proměnných x, y, tedy $A = A(x, y) \cdots G = G(x, y)$, kdy tyto funkce musí být spojité na určit oblasti Ω, tedy oblasti, v níž danou diferenciální rovnici řešíme. Analogicky $z = z(x, y)$ je neznámou funkcí.

Pro jistý druh klasifikace lze sestavit determinant

$$\delta_{det} = \begin{vmatrix} A(x,y) & B(x,y) \\ B(x,y) & C(x,y) \end{vmatrix}, \quad (2.7)$$

případně lze použít vztah pro výpočet diskriminantu

$$\delta_{disk} = B^2 - 4AC \quad (2.8)$$

a poté lze na základě formální podobnosti s rovnicemi kuželoseček klasifikovat tyto diferenciální rovnice následujícím způsobem (viz [21], [9]):

- **Parabolická parciální diferenciální rovnice** je taková parciální diferenciální rovnice, kde $\delta_{det} = \delta_{disk} = 0$.
- **Eliptická parciální diferenciální rovnice** je taková parciální diferenciální rovnice, kde $\delta_{det} > 0$ případně $\delta_{disk} < 0$.
- **Hyperbolická parciální diferenciální rovnice** je taková parciální diferenciální rovnice, kde $\delta_{det} < 0$ případně $\delta_{disk} > 0$.

Podmínkou pro dříve uvedenou klasifikaci je, že δ_{det} si musí zachovávat své znaménko na celé oblasti Ω.

Významnou hyperbolickou parciální diferenciální rovnicí je **vlnová rovnice**, pomocí které lze modelovat celou řadu fyzikálních jevů, např. různá vlnění, či kmitání struny. Zpravidla se pod pojmem vlnové rovnice rozumí rovnice homogenní.

2.4 Řešení diferenciálních rovnic

V této části naznačím některé postupy řešení diferenciálních rovnic.

2.4.1 Analytické postupy

Za řešení diferenciální rovnice lze považovat každou funkci, která obsahuje příslušné derivace a vyhovuje tak dané diferenciálníc rovnici. V případě hledání řešení soustavy diferenciálních rovnic, je daným řešením soustava takových funkcí, které obsahuje patřičné derivace potřebného řádu, které vyhovují všem rovnicím řešené soustavy.

Řešení diferenciálních rovnic lze rozdělit takto ([18], [11]):

- Obecné — Za obecné řešení diferenciální rovnice považujeme takové řešení dfiferenciální rovnice, které obsahuje libovolnou integrační konstantu. Přiřadíme-li každé konstantě obecného řešení číselnou hodnotu, pak dostaneme řešení partikulární.
- Partikulární — Partikulární (částečné) řešení diferenciální rovnice je řešení diferenciální rovnice, které získáme přiřazením určité číselné hodnotě každé integrační konstantě obecného řešení.
- Singulární — Některá řešení diferenciální rovnice nelze získat z obecného řešení. Taková řešení, která se vyskytují pouze u některých rovnic, označujeme jako singulární (výjimečné)..

V případě řešení jednoduchých diferenciálních rovnic lze získat partikulární řešení diferenciálních rovnic analyticky. V případě složitějších diferenciálních rovnic je zpravidla analytické řešení příliš obtížhné, proto se používá numerické řešení diferenciálních rovnic.

2.4.2 Numerické metody

Numerické řešení diferenciálních rovnic se používá tehdy, pokud by nalezení analytického řešení diferenciální rovnice (resp. soustavy diferenciálních rovnic) bylo obtížné, nebo v případech, kdy nalezení analytického řešení diferenciální rovnice není možné. Numerické metody lze rozdělit na **jednokrokové** a **vícekrokové**[1]. Mezi běžně používané jednokrokové metody patří např. metoda Eulerova, metody Runge-Kutta a metody s využitím Taylorova polynomu.

Eulerova metoda

Eulerova metoda je nejjednodušší numerickou metodou pro řešení diferenciálních rovnic ([20], [13]). Lze ji považovat za metodu prvního řádu.

Chceme řešit diferenciální rovnici s počátečními podmínkami

$$y' = f(t, y(t)), \qquad y(t_0) = y_0.$$

Použijí se první dva členy Taylorova rozvoje, které reprezentují lineární aproximaci hledaného řešení okolo bodu $(t_0, y(t_0))$. Pro jeden krok výpočtu platí vztah

$$y_{n+1} = y_n + hf\left(t_n, y_n\right),$$

kde konstanta h reprezentuje krok výpočtu. Eulerova metoda je metodou explicitní.

[1] Další možné dělení je na implicitní a explicitní.

Metody Runge-Kutta

Metody Runge-Kutta tvoří celou rodinu numerických integračních metod. Běžně používaná metoda je označována **RK4**, tedy metoda Runge-Kutta 4. řádu (RK4, **??**). Řešíme deferenciální rovnici s počátečními podmínkami a krokem h.

$$y' = f(t, y(t)), \qquad\qquad y(t_0) = y_0$$

Pak je metoda RK4 pro tento problém dána rovnicemi

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ t_{n+1} &= t_n + h, \end{aligned}$$

kde y_{n+1} je aproximace hledanáho řešení a konstanty k_1 až k_4 jsou dány vztahy

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \\ k_3 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right) \\ k_4 &= f\left(t_n + h, y_n + hk_3\right) \end{aligned}$$

Taylorův polynom

Taylorův polynom aproximuje hodnoty funkce $f(x)$, která má v daném bodě a derivaci, pomocí polynomu, jehož koeficienty závisí na derivacích funkce v tomto bodě ([23]). Je definován vztahem

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Pokud má aproximovaná funkce f derivace až do řádu n, pak funkce f v bodě a je polynom

$$T_n = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k,$$

kde nultou derivací je myšlena samotná funkce, tedy $f^{(0)} = f$.

Vícekrokové metody

Vícekrokové metody získávají hodnotu y_{n+1} z předchozích hodnot y_{n-i} proložených nějakých interpolačním polynomem. Řád metody v tomto případě odpovídá řádu interpolačního polynomu. Obecný vzorec vícekrokové metody lze zapsat takto:

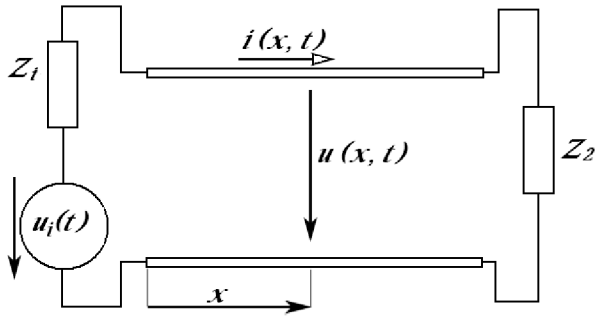
$$y_{n+1} = \sum_{i=0}^{r} \alpha_i y_{n-i} + h \sum_{j=-1}^{s} \beta_j f_{n-j}.$$

Kapitola 3

Vedení

Vedení jsou obecně vzato určité přenosové prvky sloužící k přenosu energie (tepelná či elektrická, atd.) nebo informace na nějakou delší vzdálenost. Z toho plyne i praktická realizace takovéhoto vedení. Zpravidla je vedení tvořeno soustavou dvou či více nějakých vodičů, a to rovnoběžných, pro které platí, že délka vodičů je mnohonásobně větší než příčná vzdálenost mezi nimi.

V této práci se omezím pouze na analýzu dvouvodičového vedení. Schema takového dvouvodičového vedení je znázorněno na následujícím obrázku.



Obrázek 3.1: Schéma dvouvodičového vedení

Po připojení takového vedení ke zdroji proměnného elektrického napětí protéká vedením proměnný elektrický proud a v okolí takovýchto vodičů se vytvoří elektrické pole úměrné protékajícímu napětí a magnetické pole úměrné protékajícímu proudu. Vzhledem k tomu, že se jedná o proměnné elektrické napětí, je třeba uvážit, že protékající napětí a proud mění svou velikost v závislosti na vzdálenosti od zdroje elektrického napětí, ukazuje se, že šíření energie podél vedení je v podstatě **vlnový proces**.

3.1 Primární parametry vedení

Na obrázku 3.1 je znázorněno dvouvodičové vedení na jedné straně zakončené zdrojem napětí, na straně druhé pak pasivním prvkem. Vodiče mohou být libovolného příčného průřezu, či jakkoli uspořádány, nicméně předpokládá se, že příčné rozměry vodičů a jejich vzdálenost jsou mnohonásobně menší, než délka vedení. Vedení je definováno **primárními**

parametry R_0, G_0, L_0, C_0, které budou diskutovány dále.

3.1.1 Měrný elektrický odpor R_0

Měrný elektrický odpor $R_0[\Omega m^{-1}]$ je celkový činný odpor obou vodičů na jednotku délky. Protéká-li vedením jednotkové délky proud i, který vyvolává na vodičích podélný úbytek napětí $\Delta u_0 = \Delta u_{10} + \Delta u_{20}$, je měrný odpor

$$R_0 = \frac{\Delta u_0}{i}. \tag{3.1}$$

Dále se definuje **odpor elementárního úseku vedení** délky dx $R_0 dx$, **podélný úbytek napětí** $R_0 dx i$ a **výkon přeměněný v teplo** $R_0 dx i^2$.

3.1.2 Měrná příčná vodivost G_0

Měrná příčná vodivost $G_0[Sm^{-1}]$ je vodivost mezi oběma vodiči vedení na jednotku délky. Vyjadřují se jí ztráty způsobené svodem dielektrika. Pokud Δi_0 reprezentuje příčný svodový proud na jednotku délky vedení při napětí $u = konst.$, pak je měrná příčná vodivost

$$G_0 = \frac{\Delta i_0}{u}. \tag{3.2}$$

Dále se definuje **vodivost elementu** délky dx $G_0 dx$, **příčný proud** $G_0 dx u$ a **ztrátový výkon** $G_0 dx u^2$.

3.1.3 Měrná indukčnost L_0

Měrná indukčnost $L_0[Hm^{-1}]$ je indukčnost jednotkové délky vedení. Protéká-li vedením proud i, prochází plochou mezi vodiči jednotkové délky vlastní magnetický tok Φ_0 a měrná indukčnost je

$$L_0 = \frac{\Phi_0}{i}. \tag{3.3}$$

Dále se definuje **indukčnost elementu** délky dx $L_0 dx$, **indukované napětí** $L_0 dx \frac{\partial i}{\partial t}$ a **energie v magnetickém poli** $\frac{1}{2} L_0 dx i^2$.

3.1.4 Měrná kapacita C_0

Měrná kapacita $C_0[Fm^{-1}]$ je kapacita mezi vodiči vedení na jednotku délky. Pokud τ je náboj akumulovaný na jednotkovou délku vedení při napětí u, pak měrná kapacita je

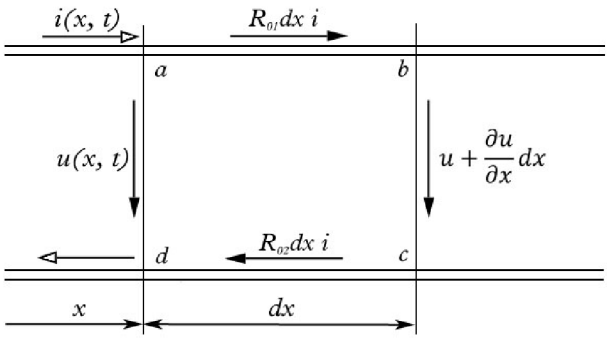
$$C_0 = \frac{\tau}{u}. \tag{3.4}$$

Dále se definuje **kapacita elementu** délky dx $C_0 dx$, **příčný kapacitní proud** $C_0 dx \frac{\partial u}{\partial t}$ a **energie v elektrickém poli** $\frac{1}{2} C_0 dx u^2$.

V případě, že se tyto primární parametry vedení nemění v celé délce vedení, pak hovoříme o **homogenním vedení**, v opačném případě o **nehomogenním vedení**.

3.2 Základní rovnice vedení

Předpokládejme, že časově proměnné napětí u i proud i se podél vedení mění spojitě a ve vzdálenosti x od počátku vedení tedy mají hodnoty $u(x,t)$ a $i(x,t)$. Vztah mezi napětím a proudem na elementu vedení délky dx pak lze vyjádřit pomocí primárních parametrů vedení. Uvažujme, že zkoumaný element vedení délky dx lze popsat jako běžnou smyčku $abcd$, jak ukazuje obrázek 3.2.



Obrázek 3.2: Napětí a proud na elementu vedení

Pak pro tuto smyčku platí věta o obvodovém napětí

$$\oint E.dl = emn = -\frac{\partial \Phi}{\partial t}. \tag{3.5}$$

Pak lze levou stranu rovnice vyjádřit pomocí dílčích napětí dle obrázku 3.2 a elektromotorické napětí na pravé straně rovnice jako napětí vlastní indukce ($L_0 dx \frac{\partial i}{\partial t}$).

$$R_{01}dxi + \left(u + \frac{\partial u}{\partial x}dx\right) + R_{02}dxi - u = -L_0 dx \frac{\partial i}{\partial t}. \tag{3.6}$$

Uvedenou rovnici lze zjednodušit aplikací úvahy $R_0 = R_{01} + R_{02}$, tedy že součet R_{01} a R_{02} je měrný odpor vedení. Rovnice pak nabude tvaru

$$-\frac{\partial u}{\partial x} = R_0 i + L_0 \frac{\partial i}{\partial t}. \tag{3.7}$$

Dále z rovnosti přitékající a odtékajících proudů v elementu vedení plyne rovnice

$$i = G_0 dx u + C_0 dx \frac{\partial u}{\partial t} + \left(i + \frac{\partial i}{\partial x}dx\right), \tag{3.8}$$

jejíž úpravou dostaneme rovnici velmi podobnou rovnici 3.7

$$-\frac{\partial i}{\partial x} = G_0 u + C_0 \frac{\partial u}{\partial t}. \tag{3.9}$$

Tyto odvozené rovnice 3.7 a 3.9 se nazývají **základní rovnice vedení**.

3.3 Telegrafní rovnice vedení

Telegrafní rovnice vedení jsou další rovnice, které popisují dvouvodičové vedení. Základní rovnice vedení se získaly odvozením z věty o obvodovém napětí a následným vyjadřováním pomocí primárních parametrů vedení. Analogicky k základním rovnicím vedení, i telegrafní rovnice budou existovat pro napětí a proud. Telegrafní rovnice se získají vzájemným řešením základních rovnic, přičemž se vždy vyloučí jedna z proměnných. V dalším textu bude odvozena diferenciální rovnice pro napětí, analogicky k ní pak existuje diferenciální rovnice pro proud.

Nejprve se bude derivovat rovnice pro napětí 3.7 podle x.

$$-\frac{\partial^2 u}{\partial x^2} = R_0 \frac{\partial i}{\partial x} + L_0 \frac{\partial^2 i}{\partial x \partial t} \tag{3.10}$$

Pak se bude derivovat rovnice pro pro proud 3.9 podle t.

$$-\frac{\partial^2 i}{\partial t \partial x} = G_0 \frac{\partial u}{\partial t} + C_0 \frac{\partial^2 u}{\partial t^2} \tag{3.11}$$

Nyní se do pravé strany rovnice 3.10 dosadí rovnice 3.11 a 3.9 a po drobných úpravách dostaneme rovnici pro napětí

$$\frac{\partial^2 u}{\partial x^2} = R_0 G_0 u + (R_0 C_0 + L_0 G_0) \frac{\partial u}{\partial t} + L_0 C_0 \frac{\partial^2 u}{\partial t^2} \tag{3.12}$$

a obráceným postupem pro proud

$$\frac{\partial^2 i}{\partial x^2} = R_0 G_0 i + (R_0 C_0 + L_0 G_0) \frac{\partial i}{\partial t} + L_0 C_0 \frac{\partial^2 i}{\partial t^2} \tag{3.13}$$

Rovnice 3.12 a 3.13 jsou již dříve zmiňované **telegrafní rovnice vedení**.

Při bližším pohledu na tyto rovnice je zřejmé, že se jedná o parciální diferenciální rovnice druhého řádu takové, o kterých pojednávala kapitola 2.3. Pro úplnost uvádím, jak by vypadaly jednotlivé koeficienty v základní rovnici parciální diferenciální rovnice druhého řádu (2.6).

$$\begin{aligned} A &= 1 \\ B &= 0 \\ C &= L_0 C_0 \\ D &= 0 \\ E &= R_0 C_0 + G_0 L_0 \\ F &= R_0 G_0 \\ G &= 0 \end{aligned}$$

Dále lze telegrafní rovnice (jakožto parciální diferenciální rovnice) klasifikovat na parabolické, eliptické a hyperbolické. Záleží pouze na volbě koeficientů. Vzhledem k zápisu telegrafních rovnic je zřejmé, že v klasifikačních vztazích 2.7, 2.8 bude vždy hodnota $B = 0$, záleží tedy jen a pouze na parametrech A a C, přičemž hodnota parametru A vždy bude buď $A = 1$ nebo $A = -1$ podle toho, jak bude daná telegrafní rovnice zapsána.

V předchozím textu jsem uvažoval i reálnou možnost, že na vedení vznikají ztráty. Pokud by se ale tyto ztráty na vedení zanedbaly, pak se zjednoduší telegrafní rovnice. Znamená to

tedy, že položíme $R_0 = G_0 = 0$, pak pro toto **bezztrátové vedení** budou platit **vlnové** rovnice ve tvaru:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} &= \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2} \\ \frac{\partial^2 i}{\partial x^2} &= \frac{1}{v^2}\frac{\partial^2 i}{\partial t^2}, \end{aligned}$$

ve kterých je $\frac{1}{v^2} = L_0 C_0$.

Je potřeba si uvědomit, že na skutečném elementu vedení vznikají ještě další jevy a přeměny energie než ty, které byly popsány primárními parametry vedení. Na vedení dále vznikají ztráty v dielektriku vyvolané polarizací a magnetizací. Polarizace dielektrika jsou úměrné časové změně intenzity elektrického pole nebo též napětí mezi vodiči, dají se zahrnout do příčné vodivosti. Magnetizace dielektrika jsou úměrný časové změně intenzity magnetického pole, tj. proudu a zahrnutí se do podélného odporu R_0. V definici parametru L_0 není zahrnuto magnetické pole uvnitř vodičů, to lze ale respektovat zvětšením hodnoty L_0 o vnitřní indukčnost. Dále nebyla uvažována magnetická pole příčných posuvných proudů a podélná kapacita. Tato pole jsou však většinou zanedbatelná, neboť změny $u(x)$, $i(x)$ podél vedení probíhají velmi pozvolna ve srovnání s příčnou vzdáleností vodičů.

3.4 Homogenní vedení s harmonickými proudy

Nyní uvažujme, že se napětí a proud procházející vedením mění s časem harmonicky. Zápis rovnice pro napětí a proud budou pak vypadat následovně:

$$u(x,t) = U_m sin\left[\omega t + \varphi_u(x)\right] \tag{3.14}$$

$$i(x,t) = I_m sin\left[\omega t + \varphi_i(x)\right] \tag{3.15}$$

Pro další řešení je vhodné použít symbolické metody, tedy komplexních proudů a napětí, které budou v tomto případě i funkcí prostorové souřadnice x. Komplexní okamžité hodnoty pak budou

$$\hat{u}(x,t) = \sqrt{2}\hat{U}(x)e^{j\omega t} \tag{3.16}$$

$$\hat{i}(x,t) = \sqrt{2}\hat{I}(x)e^{j\omega t} \tag{3.17}$$

a komplexní efektivní hodnoty

$$\hat{U}(x) = \frac{U_m(x)}{\sqrt{2}}e^{j\omega_u(x)} \tag{3.18}$$

$$\hat{I}(x) = \frac{I_m(x)}{\sqrt{2}}e^{j\omega_i(x)}. \tag{3.19}$$

Dále lze dosadit rovnice 3.16 a 3.17 do základních rovnic vedení 3.7 a 3.9, dostaneme

$$-\sqrt{2}e^{j\omega t}\frac{\partial\hat{(U)}(x)}{\partial x} = R_0\hat{i} + j\omega L_0\hat{i} = \sqrt{2}e^{j\omega t}\left(R_0 + j\omega L_0\right)\hat{I}(x), \tag{3.20}$$

$$-\sqrt{2}e^{j\omega t}\frac{\partial\hat{(I)}(x)}{\partial x} = G_0\hat{i} + j\omega C_0\hat{i} = \sqrt{2}e^{j\omega t}\left(G_0 + j\omega C_0\right)\hat{U}(x). \tag{3.21}$$

V těchto rovnicích lze krátit výrazem $\sqrt{2}e^{j\omega t}$, takže základní rovnice vedení pak nabudou tvaru

$$-\frac{d\hat{U}(x)}{dx} = \hat{Z}_0\hat{I}(x) \tag{3.22}$$

$$-\frac{d\hat{I}(x)}{dx} = \hat{Y}_0\hat{U}(x). \tag{3.23}$$

Jsou to obyčejné diferenciální rovnice pro komplexní efektivní hodnoty v počáteční poloze. Derivace podle času jsou nahrazeny operátorem $j\omega$ a jsou zahrnuty do parametrů **podélné měrné impedance** $\hat{Z}_0 = R_0 + j\omega L_0$ $[\Omega/m]$ a **podélné měrné admitance** $\hat{Y}_0 = G_0 + j\omega C_0$ $[S/m]$.

V předchozím textu byly ze základních rovnic získány jejich vzájemným řešením telegrafní rovnice vedení. Stejně tak nyní lze stejným postupem získat tyto telegrafní rovnice vedení.

$$\frac{d^2\hat{U}}{dx^2} = \hat{Z}_0\hat{Y}_0\hat{U} = \hat{\gamma}^2\hat{U} \tag{3.24}$$

$$\frac{d^2\hat{I}}{dx^2} = \hat{Z}_0\hat{Y}_0\hat{I} = \hat{\gamma}^2\hat{I} \tag{3.25}$$

Konstanta $\hat{\gamma}$ je tzv. **měrný činitel přenosu** (činitel šíření, konstanta šíření).

$$\hat{\gamma} = \beta + j\alpha = \sqrt{\hat{Z}_0\hat{Y}_0} = \sqrt{(R_0 + j\omega L_0)\,(G_0 + j\omega C_0)} \tag{3.26}$$

Reálná část β je **měrný útlum**, imaginární část α je **měrný posuv**.

Kořeny charakteristické rovnice k 3.24 jsou $\pm\gamma$ a obecné řešení pro napětí je dáno součtem dvou složek

$$\hat{U}(x) = \hat{U}_{p1}e^{-\hat{\gamma}x} + \hat{U}_{z1}e^{\hat{\gamma}x} = \hat{U}_{p1}(x) + \hat{U}_{z1}(x), \tag{3.27}$$

kde $\hat{U_{p1}}$ a $\hat{U_{z1}}$ jsou integrační konstanty. Dílčí řešení

$$\hat{U}_p(x) = \hat{U}_{p1}e^{-\hat{\gamma}x} \tag{3.28}$$

$$\hat{U}_z(x) = \hat{U}_{z1}e^{\hat{\gamma}x} \tag{3.29}$$

mají charakter postupné a zpětné harmonické vlny napětí. Proud lze získat dosazením řešení 3.27 do základní rovnice 3.22.

$$\hat{I}(x) = \frac{1}{\hat{Z}_0}\frac{d\hat{U}(x)}{dx} = \frac{\hat{\gamma}}{\hat{Z}_0}(\hat{U}_{p1}e^{-\hat{\gamma}x} - \hat{U}_{z1}e^{\hat{\gamma}x}). \tag{3.30}$$

Pak výraz

$$\frac{\hat{Z}_0}{\hat{\gamma}} = \frac{\hat{Z}_0}{\sqrt{\hat{Z}_0\hat{Y}_0}} = \sqrt{\frac{\hat{Z}_0}{\hat{Y}_0}} = \hat{Z}_v = z_v e^{j\varphi_v} \tag{3.31}$$

je tzv. **vlnová impedance homogenního vedení**. Proud je pak dán rovnicí

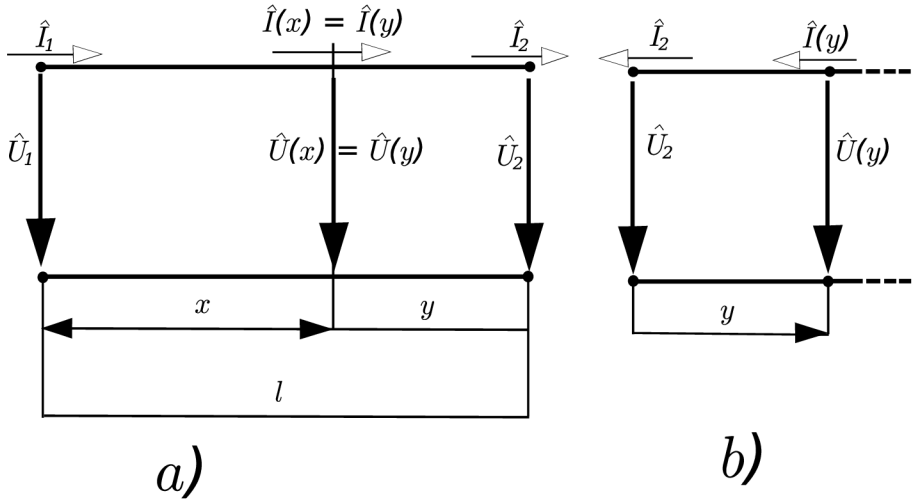
$$\hat{I}(x) = \frac{\hat{U}_{p1}}{\hat{Z}_v}e^{-\hat{\gamma}x} - \frac{\hat{U}_{z1}}{\hat{Z}_v}e^{\hat{\gamma}x} = \hat{I}_p(x) + \hat{I}_z(x). \tag{3.32}$$

Složky

$$\hat{I}_p(x) = \frac{\hat{U}_p(x)}{\hat{Z}_v} = \frac{\hat{U}_{p1}}{\hat{Z}_v} e^{-\hat{\gamma}x} \tag{3.33}$$

$$\hat{I}_z(x) = \frac{-\hat{U}_z(x)}{\hat{Z}_v} = \frac{-\hat{U}_{z1}}{\hat{Z}_v} e^{\hat{\gamma}x} \tag{3.34}$$

mají opět výraz postupné a zpětné vlny proudu.



Obrázek 3.3: Volba souřadnic a počátečních podmínek

Pro stanovení integračních konstant $\hat{U}_{p1}$, $\hat{U}_{z1}$ zavedeme souřadnice podle obrázku 3.3 a). Počátek vedení bude ztotožněn s počátkem souřadnic, tedy $x = 0$, konec pak je v místě $x = 1$. Napětí a proud na vedení určíme z hodnot $\hat{U}_1$, $\hat{I}_1$ na počátku vedení. Dosazením $x = 0$, $\hat{U}(x = 0) = \hat{U}_1$ do 3.27 a 3.32 dostaneme

$$\hat{U}_1 = \hat{U}_{p1} + \hat{U}_{z1} \tag{3.35}$$

$$\hat{I}_1 = \frac{1}{\hat{Z}_v}\left(\hat{U}_{p1} - \hat{U}_{z1}\right) \tag{3.36}$$

Řešením obou rovnic dostaneme pro integrační konstanty výrazy

$$\hat{U}_{p1} = \hat{U}_{p1} e^{j\varphi_{p1}} = \frac{\hat{U}_1 + \hat{Z}_v \hat{I}_1}{2}, \tag{3.37}$$

$$\hat{U}_{z1} = \hat{U}_{z1} e^{j\varphi_{z1}} = \frac{\hat{U}_1 - \hat{Z}_v \hat{I}_1}{2}. \tag{3.38}$$

Hledané řešení pak je

$$\hat{U}(x) = \frac{\hat{U}_1 + \hat{Z}_v \hat{I}_1}{2} e^{-\hat{\gamma}x} + \frac{\hat{U}_1 - \hat{Z}_v \hat{I}_1}{2} e^{\hat{\gamma}x}, \tag{3.39}$$

$$\hat{I}(x) = \frac{\hat{U}_1 + \hat{Z}_v \hat{I}_1}{2\hat{Z}_v} e^{-\hat{\gamma}x} + \frac{\hat{U}_1 - \hat{Z}_v \hat{I}_1}{2\hat{Z}_v} e^{\hat{\gamma}x}. \tag{3.40}$$

$$\tag{3.41}$$

Úpravou a zavedením hyperbolických funkcí vzniknou rovnice

$$\begin{aligned}\hat{U}(x) &= \hat{U}_1 cosh\gamma x - \hat{Z}_v \hat{I}_1 sinh\hat{\gamma} x,\\ \hat{I}(x) &= -\frac{\hat{U}_1}{\hat{Z}_v} sinh\hat{\gamma} x + \hat{I}_1 cosh\hat{\gamma} x.\end{aligned} \tag{3.42}$$

Často lze s výhodou použít vyjádření napětí a proudu na vedení z hodnot na konci vedení $\hat{U}_2$, $\hat{I}_2$ a v závislosti na vzdálenosti y, měřené od konce vedení podle obrázku 3.3. V obrázku 3.3 b) je znázorněn právě konec vedení s orientací y souhlasně s x. Pokud porovnáme pravou a levou část obrázku je zřejmé, že řešení bylo získáno záměnou $x \to y$, $\hat{U}_1 \to \hat{U}_2$, $\hat{I}_1 \to \hat{I}_2$, $\hat{I}(x) = -\hat{I}(y)$. Rovnicím 3.39 a 3.40 odpovídá řešení

$$\begin{aligned}\hat{U}(y) &= \frac{\hat{U}_2 + \hat{Z}_v \hat{I}_2}{2} e^{\hat{\gamma} y} + \frac{\hat{U}_2 - \hat{Z}_v \hat{I}_2}{2} e^{-\hat{\gamma} y} = \hat{U}_{p2} e^{\hat{\gamma} y} + \hat{U}_{z2} e^{-\hat{\gamma} y},\\ \hat{I}(y) &= \frac{\hat{U}_2 + \hat{Z}_v \hat{I}_2}{2\hat{Z}_v} e^{\hat{\gamma} y} + \frac{\hat{U}_2 - \hat{Z}_v \hat{I}_2}{2\hat{Z}_v} e^{-\hat{\gamma} y} = \frac{\hat{U}_{p2}}{\hat{Z}_v} e^{\hat{\gamma} y} + \frac{\hat{U}_{z2}}{\hat{Z}_v} e^{-\hat{\gamma} y},\end{aligned} \tag{3.43}$$

kde

$$\begin{aligned}\hat{U}_{p2} &= \frac{\hat{U}_2 + \hat{Z}_v \hat{I}_2}{2},\\ \hat{U}_{z2} &= \frac{\hat{U}_2 - \hat{Z}_v \hat{I}_1}{2} -\end{aligned} \tag{3.44}$$

Rovnicím 3.42 odpovídá řešení

$$\begin{aligned}\hat{U}(y) &= \hat{U}_2 cosh\gamma y - \hat{Z}_v \hat{I}_2 sinh\hat{\gamma} y,\\ \hat{I}(y) &= -\frac{\hat{U}_2}{\hat{Z}_v} sinh\hat{\gamma} y + \hat{I}_1 cosh\hat{\gamma} y.\end{aligned} \tag{3.45}$$

Střední hodnota výkonu postupujícího vedením je dána vztahem

$$P = Re\left\{\hat{U}(x)\hat{I}(x)\right\} = Re\left\{\hat{U}(y)\hat{I}(y)\right\} \tag{3.46}$$

Rozdíl střední hodnoty výkonů $P(x_1) - P(x_2)$ je roven ztrátám v úseku vedení délky $(x_1 - x_2)$.

Výsledky ukazují, že napětí, proud i výkon v libovolném místě vedení jsou určeny počátečními podmínkami $\hat{U}_1$, $\hat{I}_1$, resp. $\hat{U}_2$, $\hat{I}_2$, souřadnicí x nebo y a konstantami $\hat{Z}_v$ a $\hat{\gamma}$, které se nazývají **sekundární parametry vedení**.

Sekundární parametry definované rovnicemi 3.26 a 3.31 jsou komplexními funkcemi reálných parametrů R_0, G_0, C_0, L_0 a ω.

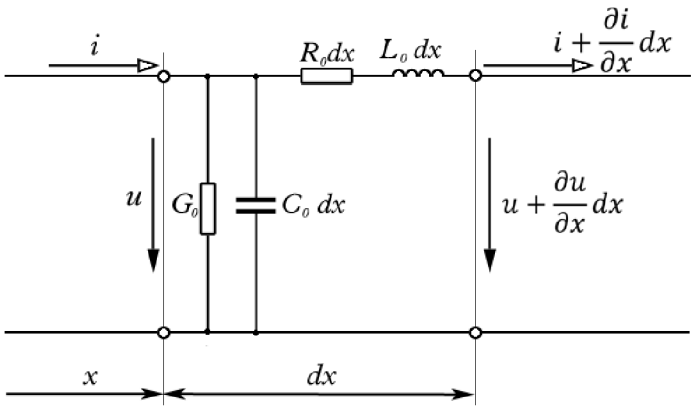
Frekvenční závislost sekundárních parametrů je patrná z definičních vztahů. U **reálných vedení** však vykazují frekvenční závislost i primární parametry vedení. Nejvýrazněji se mění podélný odpor v důsledku tzv. povrchového jevu ($R_0 \sim \sqrt{\omega}$). Také příčná vodivost se u méně kvalitních dielektrik zvyšuje s frekvencí v důsledku ztrát polarizací. Parametry L_0 a C_0 lze považovat v širokém rozsahu frekvencí za konstantní.

3.5 Pohled na vedení jako elektrický obvod

V předchozích kapitolách byly definovány primární parametry vedení. Tyto parametry do značné míry korespondují s běžnými elektrickými součástkami: rezistorem, kondenzátorem a cívkou. Ukazuje se, že pohled na vedení jako na "běžný" elektrický obvod může pomoci při analýze dějů na vedení.

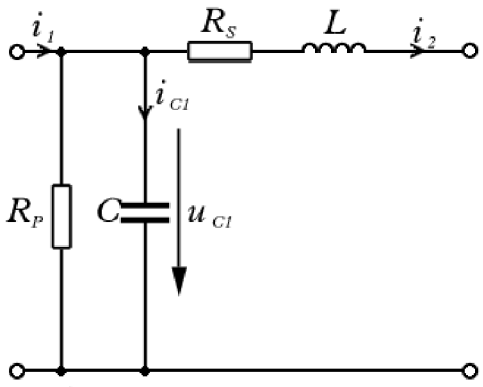
Primární parametry vedení a vztahy od nich odvozené více či méně závisely na vzdálenosti od zdroje napětí a délce měřeného (analyzovaného) úseku (tato se však volila jednotková). Nic tedy nebrání dané vedení rozdělit na části o jednotkové délce a tyto části pak analyzovat.

Na element vedení tedy lze pohlížet jako na náhradní dvojbran, který při zanedbání veličin vyšších řádů vyhovuje základním rovnicím.



Obrázek 3.4: Náhradní dvojbran elementu vedení

Pak lze na celé vedení pohlížet jako na kaskádu takových dvojbranů s tím, že délka jednoho dvojbranu je až nekonečně malá. Předchozí obrázek zachycuje pohled na element vedení nejobecnějším způsobem. V literatuře [7] se vyskytuje i modifikované resp. zjednodušené zapojení, které lze po připojení zdroje napětí a jistým ukončením (např. připojení



Obrázek 3.5: Zjednodušené schéma dvojbranu

rezistoru) modelovat v nástroji TKSL. V tomto případě jsem se omezil pouze na jeden element (tedy dalo by se říci, že celé vedení bylo modelováno pouze jedním dvojbranem) a zápis použitých diferenciálních rovnic v TKSL vypadal následovně:

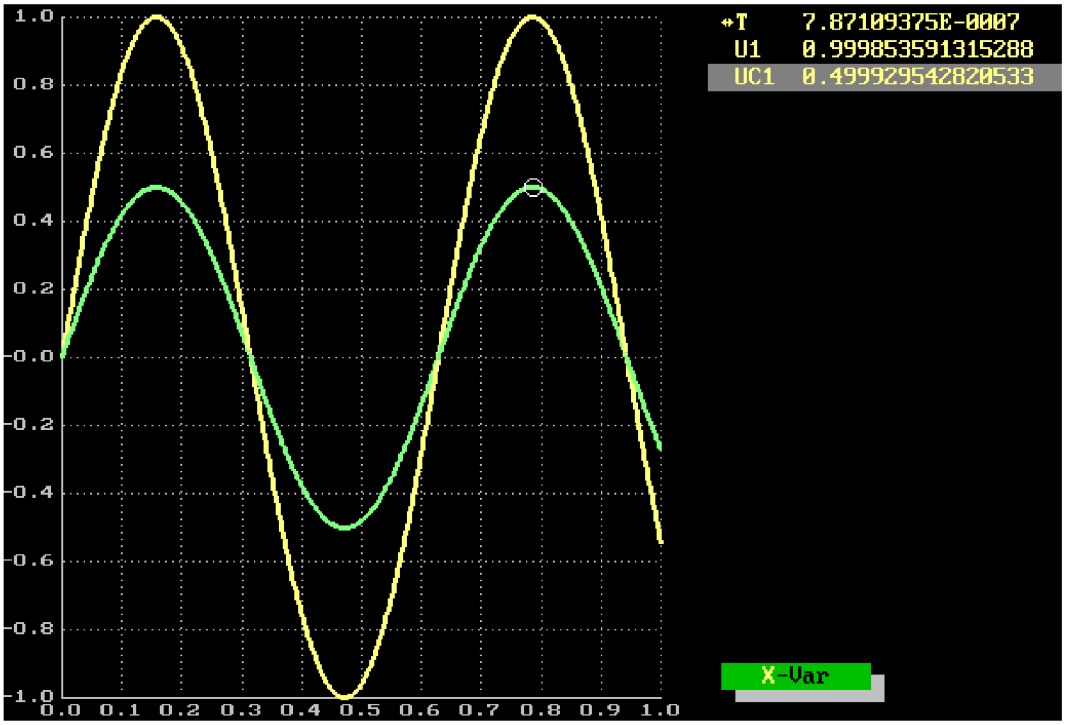
$$u'_{C1} = \frac{1}{C} \cdot i_{C1} \quad u_{C1}(0) = 0 \tag{3.47}$$

$$i'_2 = \frac{1}{L} \cdot (u_{C1} - R_S i_2) \quad i_2(0) = 0 \tag{3.48}$$

3.6 Experimenty s homogenním vedením

V předchozích částech jsem odvodil rovnice pro popis elementu vedení - dvojbranu. Dále jsem sestavil model tohoto dvojbranu, který jsem poté simuloval v nástrojích TKSL/386 a TKSL/c. V následujících experimentech jsem se prozatím omezil na homogenní vedení, tj. kaskáda dvojbranů byla vždy tvořena naprosto stejnými prvky se stejnými hodnotami.

Nejprve jsem provedl experiment pouze s jedním dvojbranem, jehož výstupem je graf na následujícím obrázku 3.6. V grafu je vidět chování jednoho elementu vedení. Na vstup



Obrázek 3.6: Výsledky experimentu

vedení byl přiveden harmonický signál (napětí) $u = U_m \sin(\omega t)$. V grafu je vidět, že napětí u_{C1} má oproti přivedenému napětí u menší amplitudu, což splňuje očekávání.

Další experiment jsem provedl s vedením modelovaným 5ti dvojbrany. V grafu jsem již vypustil vstupní napětí u a ponechal pouze průběhy napětí u_{C1} a u_{C5}. Z grafu je však patrné pouze to, že oba průběhy jsou si velmi podobné a výsledné hodnoty se liší až v řádech tisícin až desetitisícin (voltu).

Proto jsem dále zkoušel zvyšovat počet dvojbranů. Pro vedení tvořené kaskádou 10ti dvojbranů jsem ale dostal velmi podobný, a tedy neprůkazný výsledek.

Obrázek 3.7: Experiment s kaskádou 5ti dvojbranů

Tyto experimenty byly prováděny v prostředí TKSL/386, které má svá určitá omezení, např. v počtu možných řešených rovnic. V případě vedení tvořeného kaskádou 20ti dvojbranů již doba výpočtu byla velmi dlouhá, proto jsem přistoupil k použití novější verze, a to TKSL/c.

S programem TKSL/c jsem prováděl experimenty na kaskádě minimálně 100 dvojbranů, avšak největší vypovídací hodnotu má kaskáda 500 dvojbranů, popřípadě 1000. Pro generování takto rozsáhlých soustav diferenciálních rovnic jsem naimplementoval generátor takovéto soustavy diferenciálních rovnic.

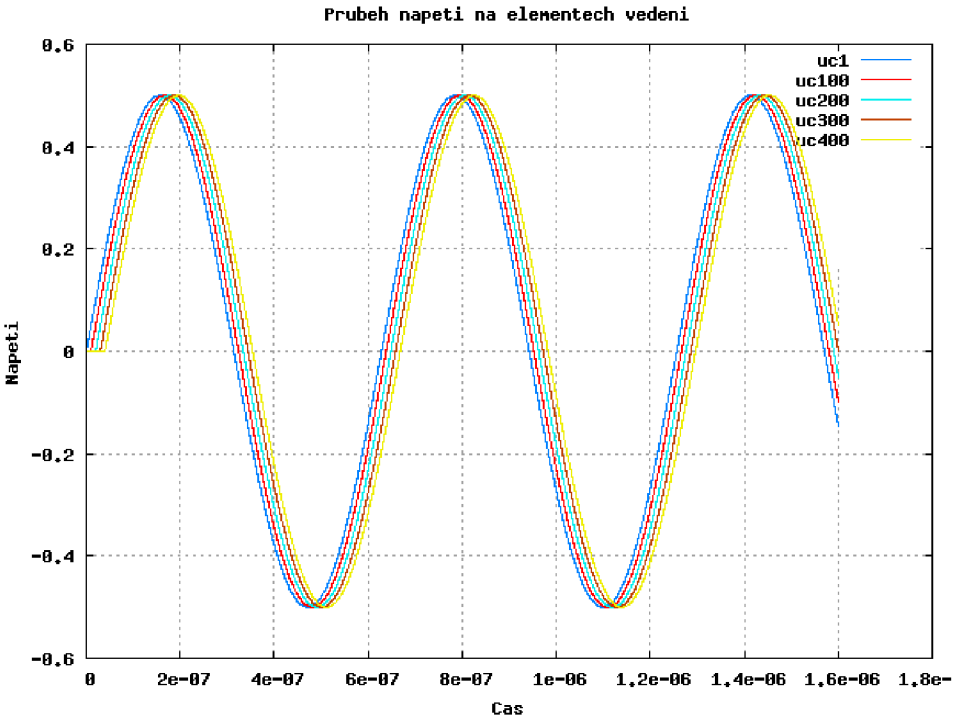
Výstup experimentu popisujícího vedení tvořené 500 dvojbrany je v následujícím grafu 3.8.

Z grafu 3.8 je patrné, že napětí u_{C100} až u_{C500} jsou oproti napětí u_{C1} opožděna. Tento výsledek je prakticky totožný s předchozími výsledky experimentů prováděných ještě se systémem TKSL/386.

Jako zajímavé se pak dále jeví zaměřit se na začátek experimentu a na místa, kde napětí dosahuje maximálních, resp. minimálních hodnot. V grafu 3.9 je tedy zachycen průběh začátku experimentu. Z grafu je pak patrné, jakým způsobem jsou napětí u_{C100} až u_{C500} opožděna. Harmonickému signálu trvá určitou dobu, než projde celou kaskádou dvojbranů, tato doba je způsobena reakcemi použitých prvků (kapacity, indukčnosti) na připojení harmonického signálu.

Dalším pro analýzu výsledků zajímavým místem, je část grafu, kde hodnoty napětí dosahují svých maximálních (resp. minimálních) hodnot. Tato část je znázorněna v následujícím grafu 3.10. Je vidět, že maximální hodnoty průběhů jednotlivých napětí u_{C1} až u_{C500} jsou oproti sobě neustále zpožďovány a navíc dochází k tlumení procházejícího signálu.

Vycházíme-li z předpokladu, že diskutované vedení je modelováno 500 dvojbrany, které



Obrázek 3.8: Experiment s kaskádou 500 dvojbranů

vedení rovnoměrně rozdělují na 500 částí stejné délky s tím, že délka každé takové části se limitně blíží nule, pak lze říci, že harmonický signál procházející takovouto kaskádou dvojbranů je s narůstající vzdáleností od zdroje signálu zpožďován. Také je patrné, že harmonický signál je s narůstající vzdáleností od svého zdroje postupně tlumen.

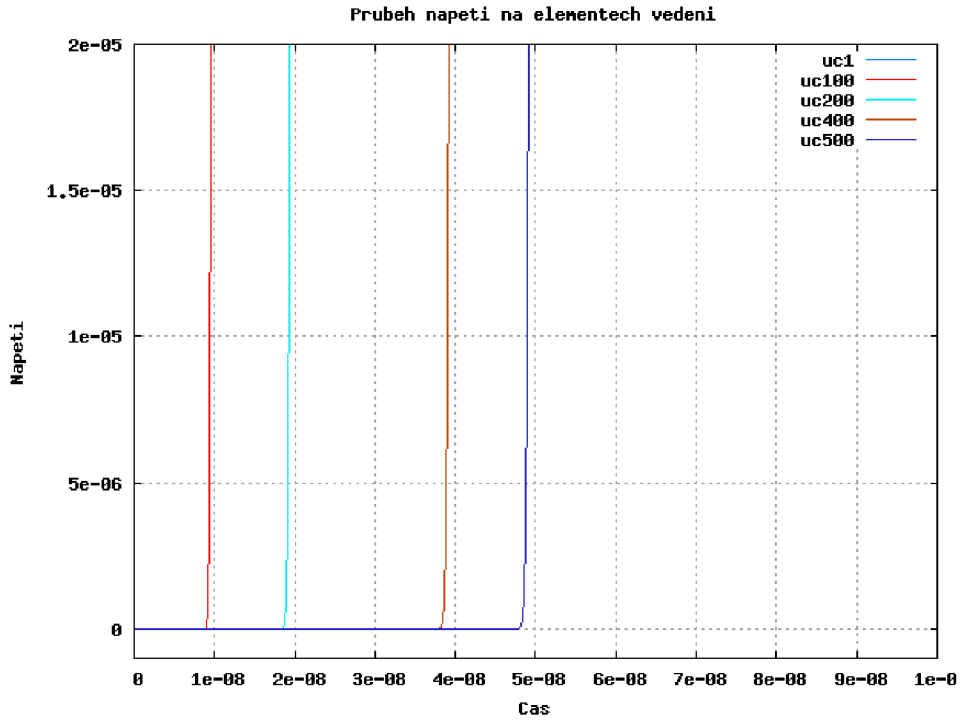
Na závěr lze tedy konstatovat, že výsledky experimentů prováděných s homogenním vedením tvořeným kaskádou 500 dvojbranů splňují očekávání.

Dále jsem provedl změnu hodnoty rezistoru R_s, a to tak, že jsem ji zmenšil z hodnoty $1 \cdot 10^{-4}$ na $1 \cdot 10^{-8}$. Provedl jsem další iteraci experimentu a získané hodnoty jsem vložil do již získaného grafu pro zdůraznění rozdílů.

Každé dvě maximální hodnoty signálu jsem dále spojil úsečkou, aby byla jasnější charakteristika průběhu tlumení signálu. Ukázalo se, že maximální hodnoty lze proložit dokonce přímkou, u níž lze vypočítat její směrnicový vektor. Je však nutné podotknout, maximální body neleží přesně na prokládané přímce, nicméně se k tomu velmi blíží. Tento jev je způsoben hlavně použitou přesností výpočtu, při zvýšení přesnosti (snížení kroku výpočtu) se body k ideální poloze (a tedy k tomu, aby ležely na přímce) velmi blíží. Celou situaci znázorňuje následující graf 3.12.

V grafu jsou znázorněny jak průběhy všech dříve sledovaných napětí, ale i dané prokladové přímky. Hodnoty vycházející z druhého experimentu (snížena hodnota rezistoru R_s) jsou označeny indexem $rs2$. Ze získaných hodnot je tedy možné spočítat směrnice přímky, v prvém případě se jedná o vektory $u_1 = (4.999 \cdot 10^{-8}, -3.36 \cdot 10^{-4})$ a $u_2 = (4.999 \cdot 10^{-8}, -1.06 \cdot 10^{-4})$.

Na základě těchto dvou experimentů vzniká hypotéza, že volba hodnoty rezistoru R_s ovlivňuje průchod napětí vedením dvěma způsoby:



Obrázek 3.9: Detail průběhu začátku experimentu

1. Menší hodnota rezistoru R_s způsobí, že maximální hodnoty signálu (napětí) na jednotlivých dvojbranech se sobě blíží. Přímka proložená maximálními hodnotami napětí na dvojbranech má menší sklon a tedy dochází k menšímu tlumení procházejícího signálu (napětí). Zpoždění signálů je zatím téměř konstantní.

2. Větší hodnota rezistoru R_s pak způsobí pravý opak, tj. přímka proložená maximy je strmější a dochází k většímu tlumení signálu. I v tomto případě zůstává zpoždění signálů téměř konstantní.

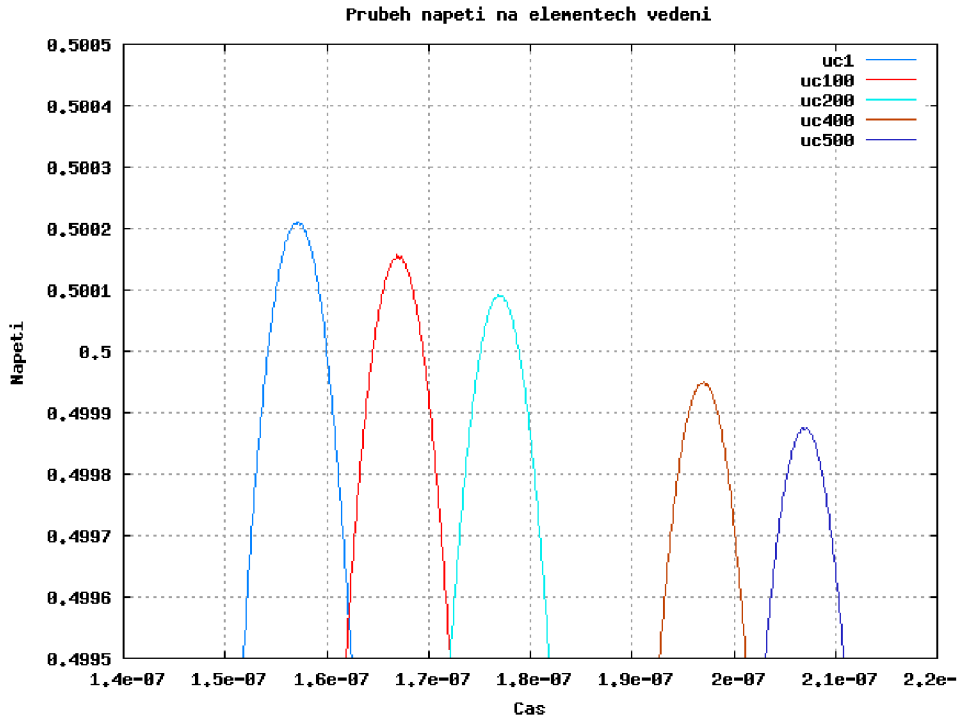
Pro ověření těchto tvrzení jsem provedl další experiment, kde jsem hodnotu rezistoru R_s zvýšil na hodnotu 10^{-2}. V následující dvou grafech jsou znázorněny nejprve průběhy napětí na vybraných elementech vedení pro volbu hodnoty rezistoru $R_s = 10^{-2}$ - graf 3.13 a v dalším grafu 3.14 jsou pak znázorněny samostatně pouze výsledné přímky, kterými jsem prokládal maximální hodnoty napětí.

Získané výsledky pouze potvrzují dříve uvedené hypotézy a získané směrnicové vektory přímek a odpovídajících hodnot rezistoru R_s jsou shrnuty v tabulce 3.1.

Nyní, když je popsán význam a chování rezistoru R_s, je vhodné se zaměřit také na chování rezistoru R_p. I zde jsem změnil jeho hodnotu z původní $R_p = 1 \cdot 10^{10}$ na hodnotu $R_{p1} = 1 \cdot 10^{15}$. Výsledek experimentu znázorňuje graf 3.15, kde je zobrazena maximální hodnota napětí uc_{c1} z prvního experimentu (označená jako uc_{1ref} a maximální hodnota sledovaného napětí u_{c1} (označená jako uc_{1rp}). Je vidět, že hodnoty se liší jen velmi málo.

Lze tedy konstatovat, že zvýšení hodnoty rezistoru R_p mělo na výsledek experimentu jen velmi malý vliv.

Další experiment tedy proběhl tak, že jsem hodnotu R_p snížil na $1 \cdot 10^6$. Výsledek



Obrázek 3.10: Detail místa maximálních hodnot napětí

Tabulka 3.1: Shrnutí vypočtených přímek

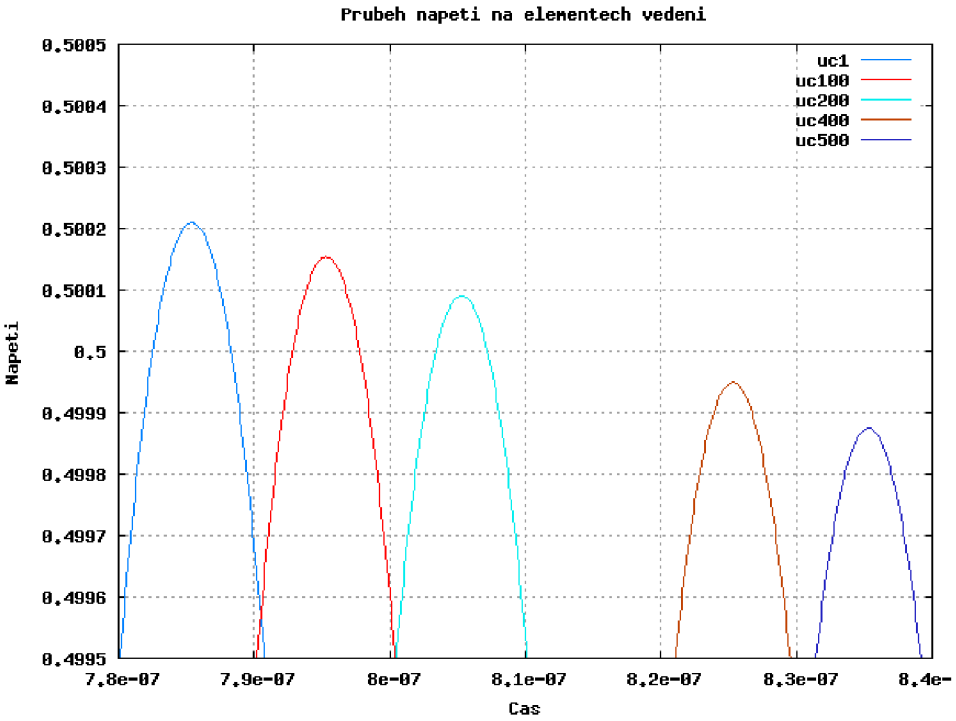
Hodnota rezistoru R_s	**Směrnicový vektor prokládané přímky**
10^{-8}	$u = (4.999 \cdot 10^{-8}, -1.06 \cdot 10^{-4})$
10^{-4}	$u = (4.999 \cdot 10^{-8}, -3.36 \cdot 10^{-4})$
10^{-2}	$u = (4.879 \cdot 10^{-8}, -0.022572)$

experimentu je vidět v grafu 3.16

Napětí označená $uc1_3$ až $uc500_3$ jsou nově získané hodnoty. Zbývající napětí jsou v grafu uvedena jako referenční[1]. Z tohoto grafu tedy již jsou patrny určité rozdíly v chování celého systému. Sledovaná napětí jsou více tlumena v porovnání s referenčními napětími a maxima jednotlivých signálů již zcela jistě nelze proložit přímkou. Začal se tedy projevovat vliv hodnoty rezistoru R_p, proto jsem opět o řád snížil hodnotu rezistoru R_p na hodnotu $1 \cdot 10^5$, jak ukazuje graf 3.17.

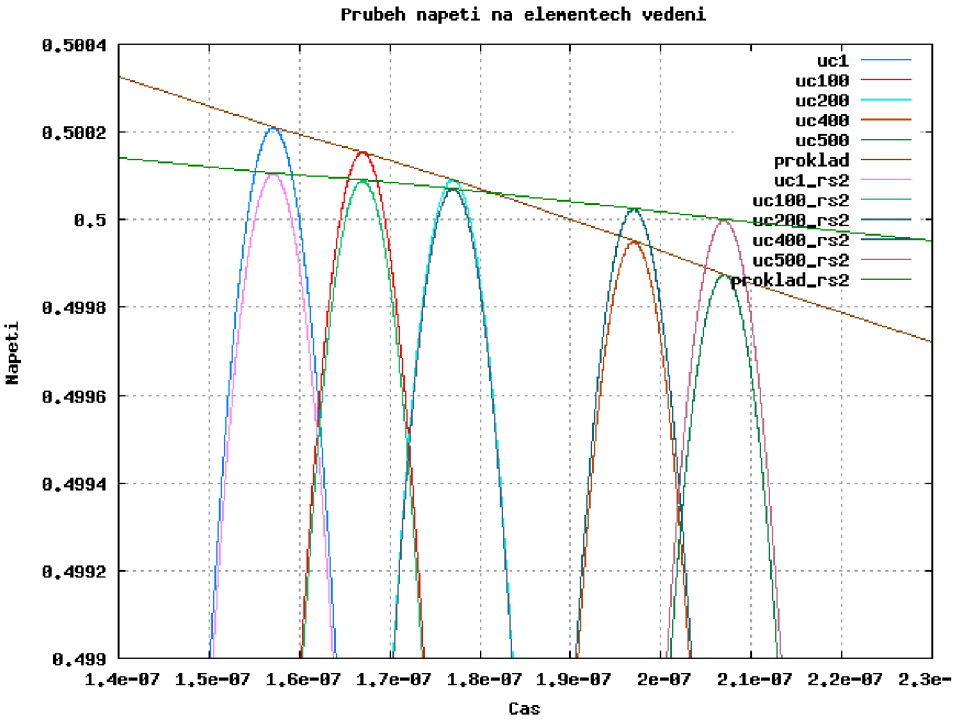
I v tomto případě je patrné značné tlumení signálů (napětí), které je ještě větší než v předchozím případě a navíc se v obou případech zvýšilo zpoždění jednotlivých napětí oproti referenčním. Jedna perioda průběhu signálu (napětí) tedy trvá delší dobu. Nejrazantnější změny však vyvolala až hodnota rezistoru $R_p = 1 \cdot 10^4$. Zde platí nejen dříve zjištěný fakt o zpožďování signálů (napětí) a tlumení, ale v tomto případě došlo poprvé k rozdílnému tlumení jednotlivých napětí v průběhu period. Celkový pohled na jednu periodu signálu poskytuje graf 3.18 a detail maximálních hodnot napětí je zobrazen v grafu 3.19.

[1] Z důvodu přehlednosti grafu byla vypuštěna napětí uc_{100}, uc_{200} a uc_{400}



Obrázek 3.11: Detail místa maximálních hodnot napětí v další periodě

Volba rezistoru R_p tedy zásadním způsobem ovlivnila chování systému.



Obrázek 3.12: Graf průběhu napětí při různých volbách hodnot rezistoru R_s

3.7 Experimenty s nehomogenním vedením

Jak již bylo uvedeno dříve, homogenní vedení je takové vedení, u něhož jsou všechny jeho primární parametry neměnné. Vzhledem k provedeným experimentům s vedením homogenním (viz předchozí kapitola), jsem se rozhodl provést experimenty i vedením nehomogenním.

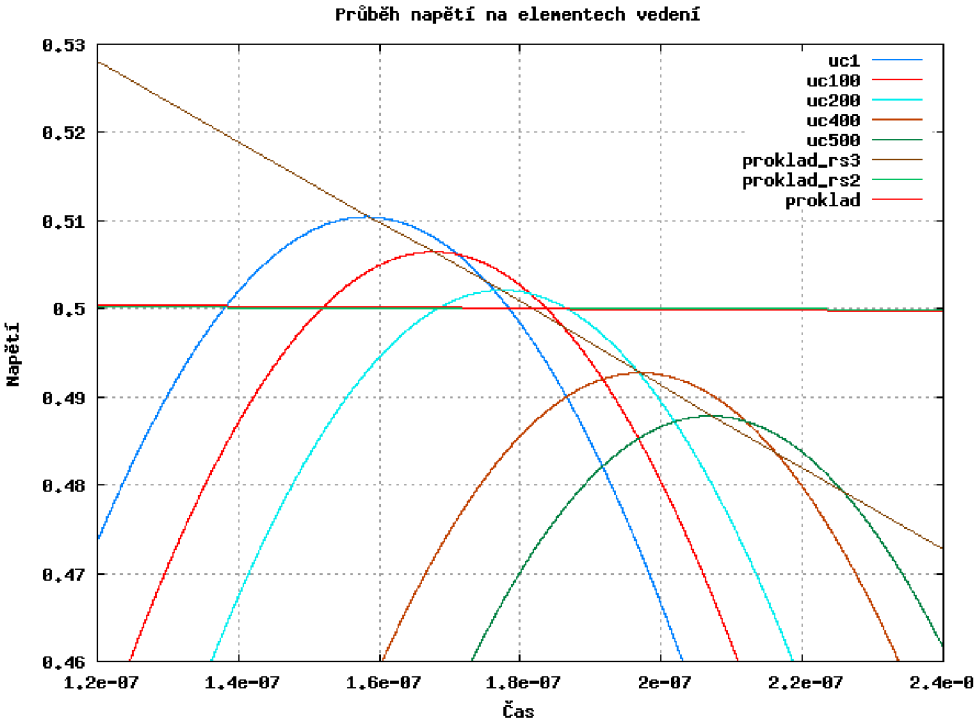
První nehomogenitu jsem do modelu zavedl tak, že jsem změnil všechny hodnoty rezistorů R_s počínaje 250.tým elementem. Z hodnoty $R_s = 10^{-4}$ jsem ji zvětšil na hodnotu $R_{s2} = 10^{-}2$. Provedl jsem pak stejný experiment jako s homogenním vedením.

V grafu 3.20 je vidět celkový průběh napětí v simulovaném modelu, v grafu 3.21 jsem se pak zaměřil opět na místa maximálních hodnot vybraných napětí. Oproti homogennímu vedení je zde patrná změna v chování signálu procházejícího kaskádou dvojbranů. Signál je sice zpožděn, ale až do 250.tého dvojbranu se chová tak, jako by nebyl tlumen. Od 250.tého dvojbranu pak je signál tlumen v souladu s výsledky získanými v předchozí kapitole.

Prováděl jsem další experimenty s takovýmto vedením a porovnával dosažené výsledky. Experimenty jsem prováděl podobně jako s homogenním vedením, tj. zaměřil jsem se na zkoumání hodnoty rezistor R_s.

Protože zavedení jedné nehomogenity dávalo zajímavý výsledek v podobě grafu 3.10, rohodl jsem jsem se zkoumat vedení se zavedením hned dvou nehomogenit. Provedl jsem tedy dva experimenty. Hodnoty rezistoru R_s včetně elementů vedení, kterých se daná hodnota týka, jsou v následující tabulce 3.2. Získané průběhy napětí pak zachycují grafy 3.22 pro první experiment a 3.23 pro druhý experiment.

V obou případech jsem dostal velmi zajímavé výsledky. V případě prvního experi-



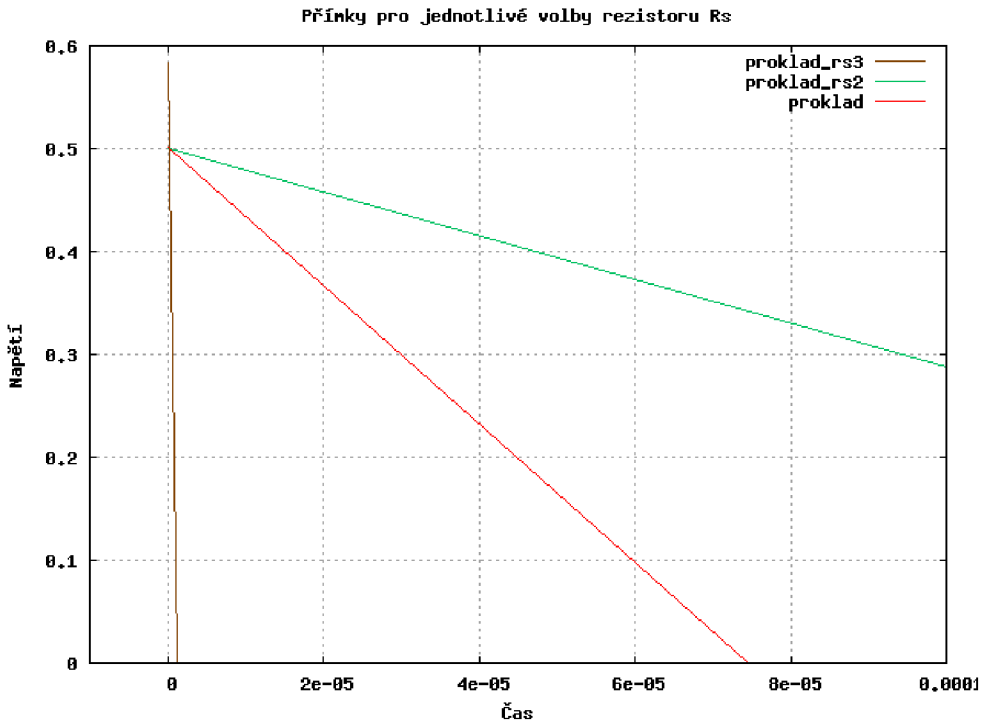
Obrázek 3.13: Graf průběhu napětí při volbě hodnoty rezistoru $R_s = 10^{-2}$

Tabulka 3.2: Hodnoty rezistorů R_s v provedených experimentech

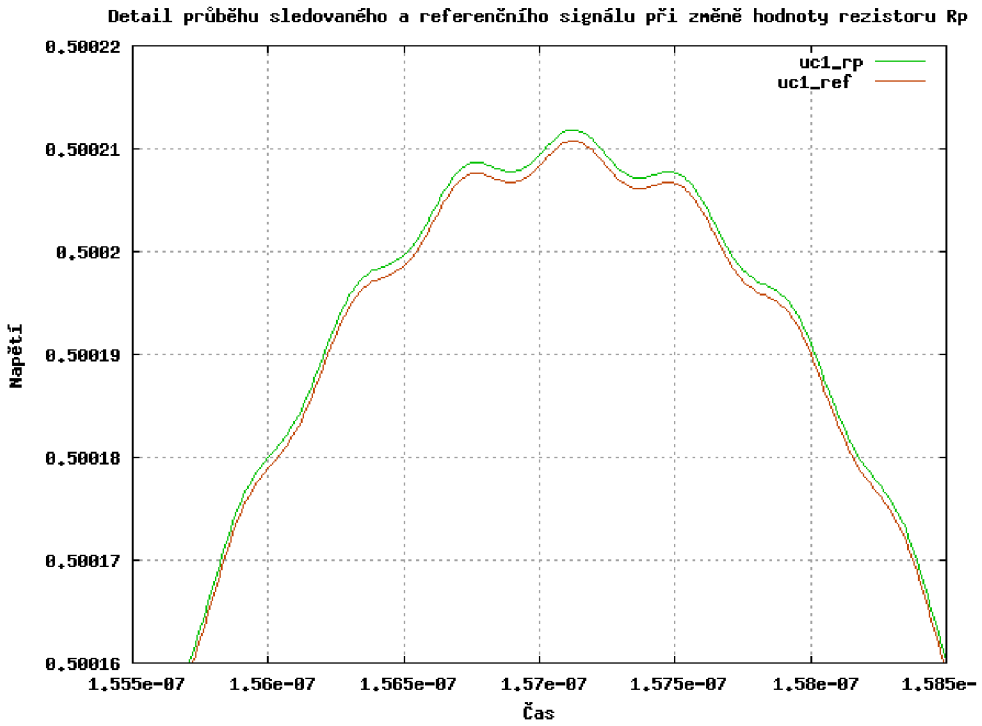
R_s **1.experiment**	R_s **2.experiment**	**Ovlivněné dvojbrany**
$1 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	1 — 166
$1 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	167 — 332
$1 \cdot 10^{-4}$	$1 \cdot 10^{-2}$	333 — 500

mentu se nejvpre napětí na jednotlivých elementech zvyšovala, jakmile byly elementy vedení ovlivněny rezistorem $R_s = 1 \cdot 10^{-2}$, docházelo k postupnému čím dál většímu tlumení procházejících napětí. Jakmile se však napětí dostalu přes tuto "bariéru" k hodnotě rezistoru $R_s = 1 \cdot 10^{-4}$, tlumení se opět zmenšovalo a úroveň napětí se pomalu zvětšovala.

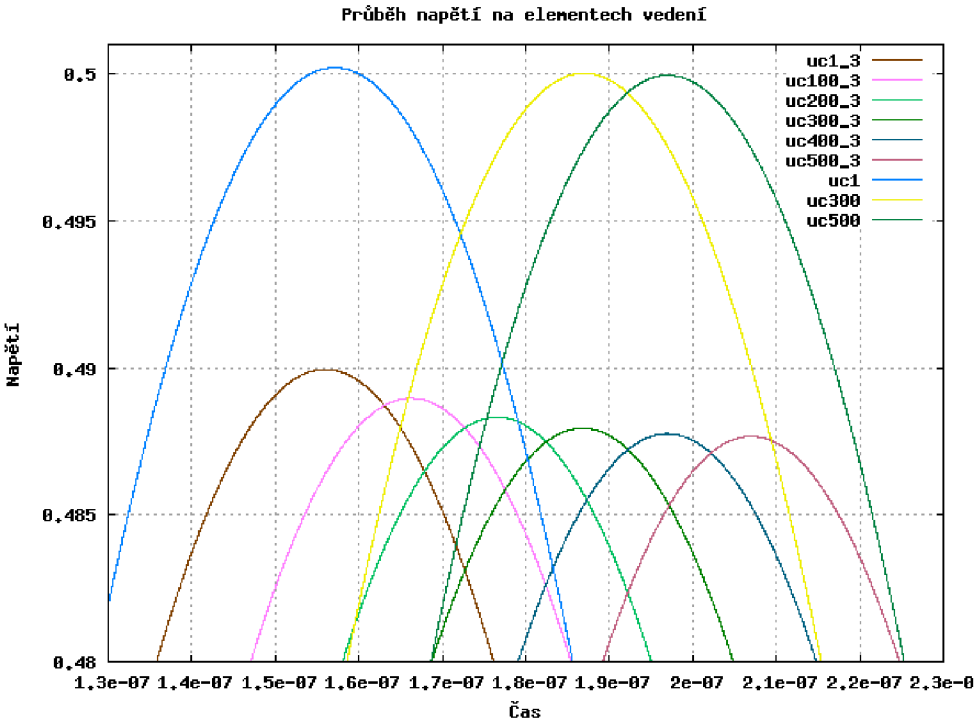
Ve druhém případě se pak zvyšovala hodnota rezisotru R_s postupně až na hodnotu $R_s = \cdot 10^{-2}$. Dokud signál nedorazil k rezistorům se zmiňovanou hodnotou, napětí nejen, že se nezdálo být tlumené, ale navíc se jeho úroveň zvyšovala. Provedené experimenty tak nabízí hypotézu, že úrovně napětí (či obecně signálu) mohou stoupat až po určitou mez, dokud nenarazí na rezistor s nejvyšší hodnotou. Zda je tuto hypotézu možné potvrdit, či vyvrátit, ukáže až další výzkum.



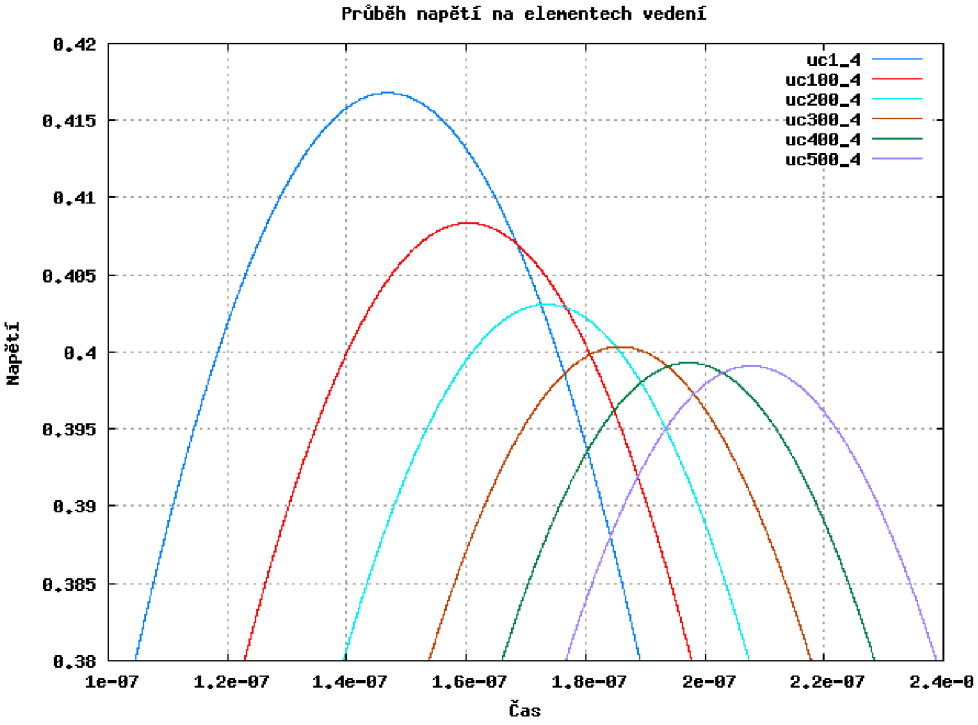
Obrázek 3.14: Přímky proložené maximálními hodnotami napětí při různých volbách hodnoty rezistoru R_s



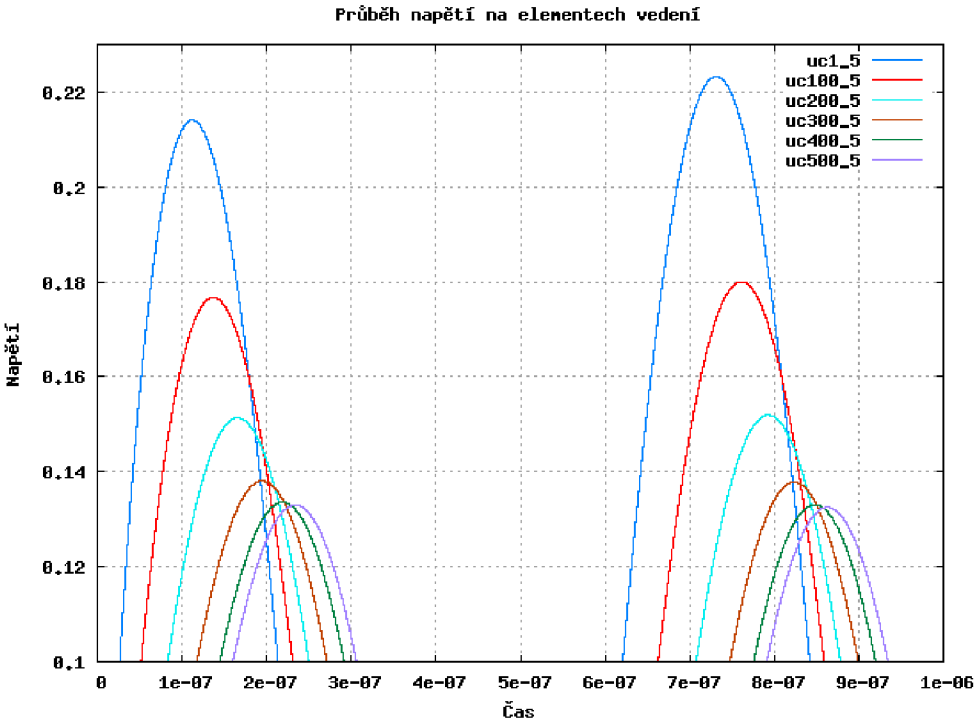
Obrázek 3.15: Detail průběhů sledovaných signálů



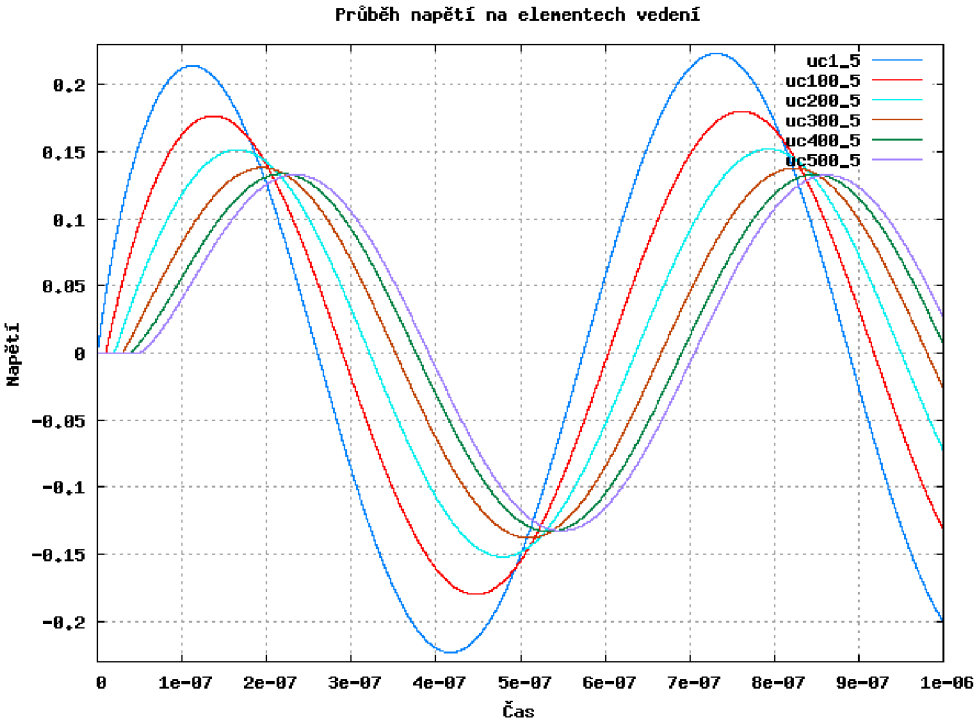
Obrázek 3.16: Detail průběhů sledovaných signálů při volbě $R_p = 1 \cdot 10^6$



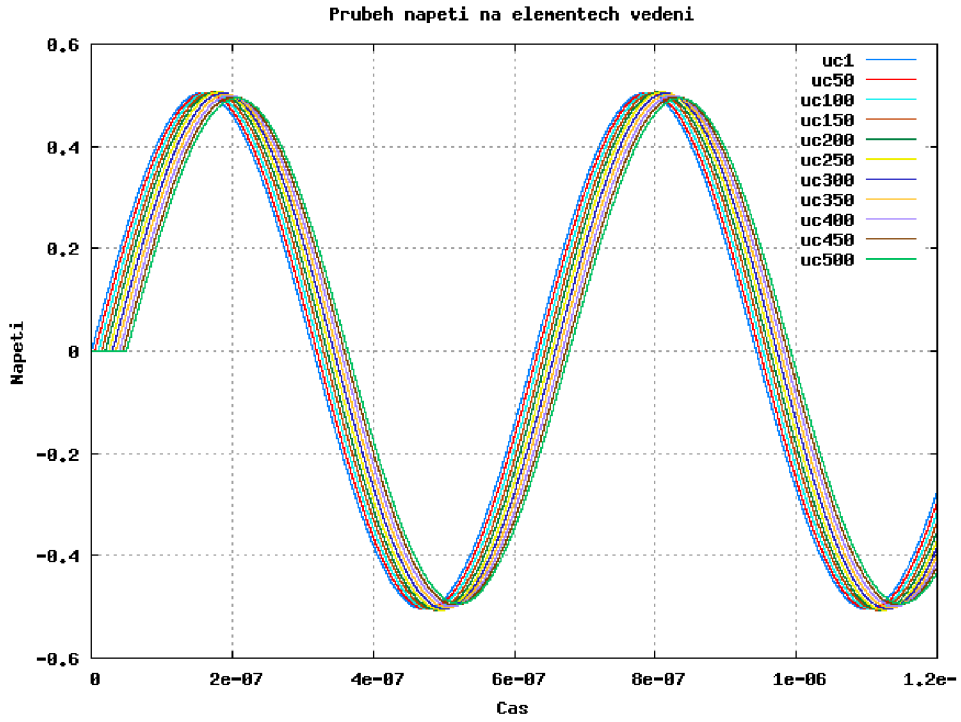
Obrázek 3.17: Detail průběhů sledovaných signálů při volbě $R_p = 1 \cdot 10^5$



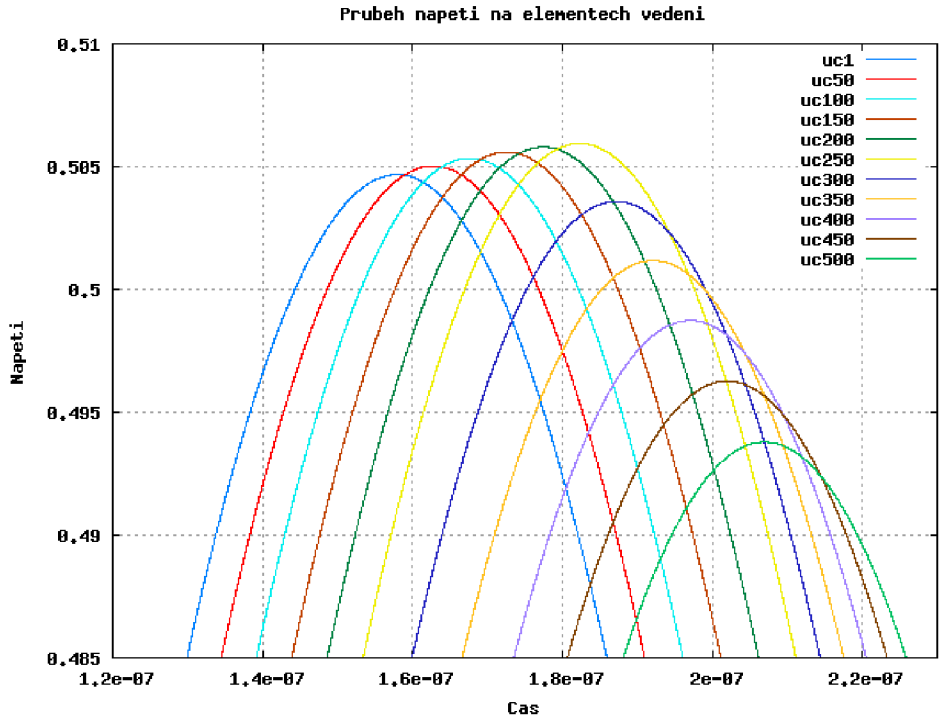
Obrázek 3.18: Detail průběhů sledovaných signálů při volbě $R_p = 1 \cdot 10^4$



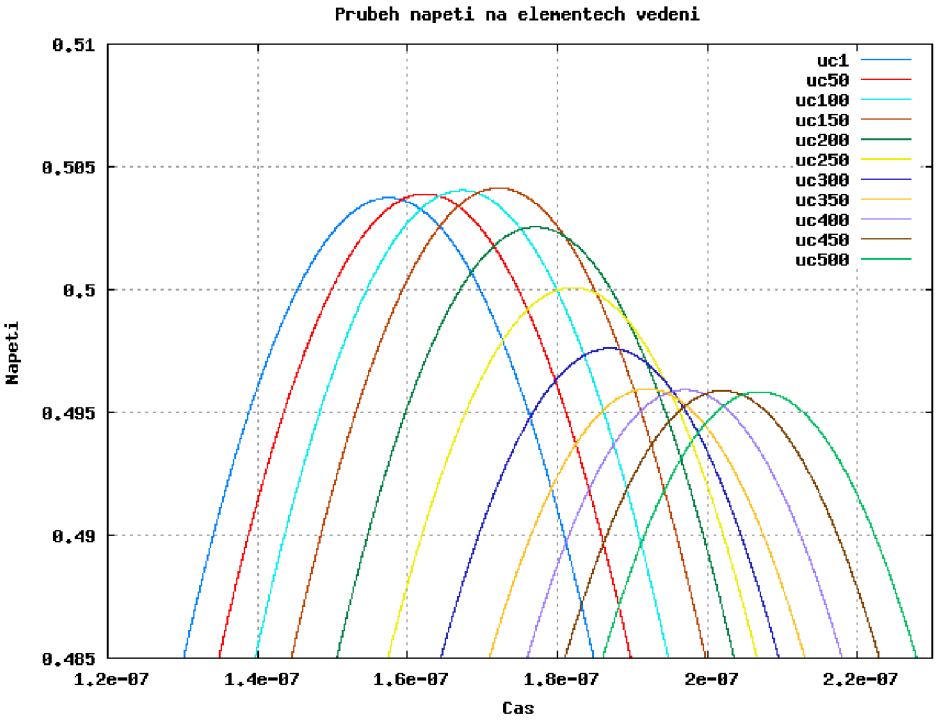
Obrázek 3.19: Detail průběhů sledovaných signálů při volbě $R_p = 1 \cdot 10^4$



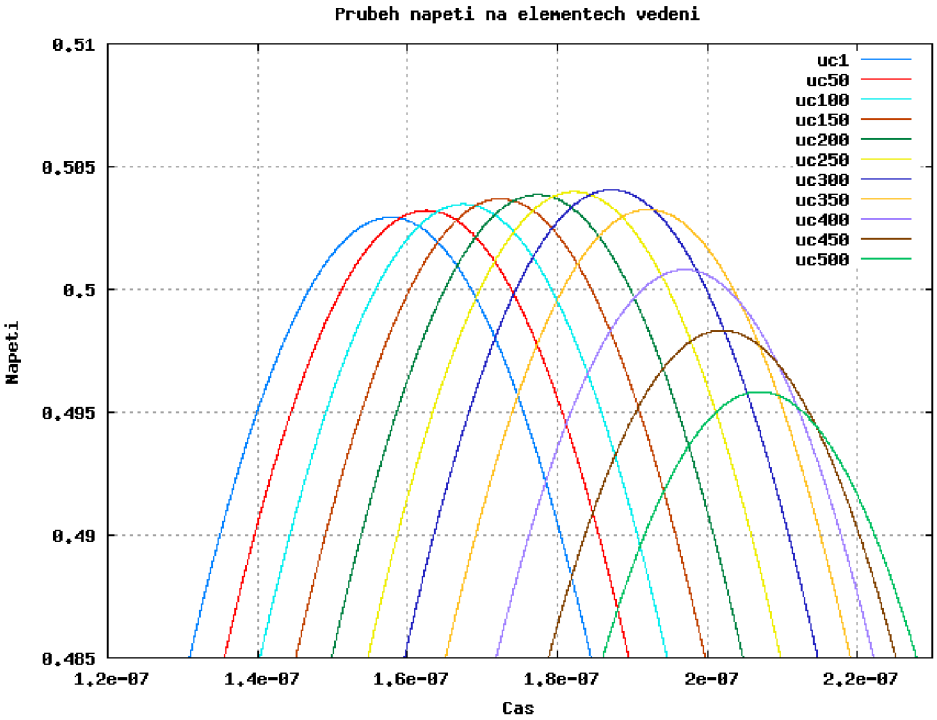
Obrázek 3.20: Průběh napětí v nehomogenním vedení



Obrázek 3.21: Detail místa maximálních hodnot napětí nehomogenní vedení



Obrázek 3.22: Průběh napětí při prvním experimentu



Obrázek 3.23: Průběh napětí pří druhém experimentu

Kapitola 4

Akcelerace výpočtů v GPU

V předchozích kapitole jsem diskutoval použití parciálních diferenciálních rovnic při analýze vedení. Řešení takovýchto rovnic je lepší přenechat nějakému automatizovanému procesu, tedy např. CPU stolního počítače. Tuto činnost provádí systém TKSL. I když vývoj procesorů jde velmi rychle dopředu a výpočetní výkon procesorů se neustále zvyšuje, procesory již nemají pouze jedno jádro, přesto na nich mohou takovéto výpočty trvat neúměrně dlouho.

Naproti tomu čipy grafických karet a jejich procesory se v dnešní době dostávají do popředí s ohledem na výpočetní výkon, který je někdy až mnohonásobně vyšší než výkon běžného CPU.

Proto je vhodné složité výpočty přenechat právě grafickým procesorům, které mají více jader, a větší rychlost a propustnost sběrnic. Pro možnost přesunu složitých výpočtů do GPU je vyvíjen n stroj nVidia CUDA.

4.1 Grafické karty

4.1.1 Historie vývoje grafických karet

V době prvních počítačů byly prvotní grafické karty používány pouze jako D/A převodníky pro převod dat z videopaměti na grafický výstup. Na počátku 70.tých let pak vznikly myšlenky o akceleraci rasterizačních operací specializovaným čipem právě na grafické kartě.

Prvními takovými čipy byly čipy ANTIC a CTIA, které nabízely vykreslování jak v textových, tak grafických režimech, vykreslování spritů.

Sprite je vlastně dvourozměrný obrázek, pomocí něhož se do "scény" vkládají pohybující se objekty. Sprite definoval u každého prvku, který se měl hýbat tzv. animační fázi prvku.

Samotný čip ANTIC byl navržen jako speciální "procesor" pro mapování textu a tehdejších grafických dat na video výstup.

S prvními takovýmito grafickými kartami jsme se mohli setkat u osmibitových počítačů ATARI, či později v 80.tých letech, Commodore Amiga. Amiga se stala průkopníkem na poli grafických čipů, byl to jeden z prvních čipů, který by v dnešní terminologii by mohl být označen jako grafický akcelerátor, protože prakticky všechny vykreslovací funkce byly přesunuty právě do hardwaru.

V 90.tých letech pak probíhal vývoj hlavně 2D hardwarové akcelerace, nicméně se na trhu začaly pomalu objevovat i první 3D akcelerátory. První pokusy o 3D akceleraci vyústily v čipy S3 Virge či ATi Rage, které však byly pouze jistým způsobem - o 3D funkce - vylepšené čipy předchozí generace pro 2D akceleraci.

V polovině 90.tých letech pak přišly na trh první "opravdové" 3D akcelerátory, které umožňovaly zpracování polygonových sítí a aplikaci textury na tyto sítě. Nicméně pořád šlo pouze o jistý druh specializovaných čipů, které nebyly schopny v GPU provádět uživatelsky definované programy, obsahovaly tedy pouze fixní vykreslovací řetězec. Tato doba rovněž znamená období masivnějšího rozšiřování OpenGL a DirectX.

V roce 2001 se společnost nVidia však postarala o vpravdě revoluci ve zpracování realtime grafiky - přivedla na trh grafický čip GeForce3 (codename NV200). Tento čip totiž na rozdíl od svých předchůdců byl jako první schopen zpracovávat pixely určitým krátkým programem, poprvé se tedy objevuje možnost programování grafického čipu. Za zmínku dále stojí čip ATi Radeon 9700 (codename R300), což byl první čip na světě, který implementoval akceleraci rozhraní Direct3D 9.0.

Grafické čipy se tedy v průběhu let stávají ze specializovaných čipů čipy programovatelnými, a tedy flexibilnějšími.

4.1.2 Propojení se systémem a přenosové rychlosti

Grafická karta je do systému připojena pomocí externího adaptéru a pro komunikaci pak využívá sběrnici. V současné době se jedná o sběrnici typu PCI-Express (PCI-E) verze 1.1 nebo novější verze 2.0 (dle [15] se již připravuje specifikace 3.0 plánovaná na konec roku 2009). Srovnání propustnosti obou verzi sběrnic PCI-Express je uvedeno v následující tabulce.

Rychlost PCI-E	**verze 1.1**	**verze 2.0**
1x	250MB/s (500MB/s)	500MB/s (1GB/s)
2x	1GB/s (2GB/s)	2GB/s (4GB/s)
4x	2GB/s (4GB/s)	4GB/s (8GB/s)
16x	4GB/s (8GB/s)	8GB/s (16GB/s)

Tabulka 4.1: Tabulka propustnosti sběrnice PCI-Express, v závorce uvedena rychlost pro obousměrnou komunikaci (převzato z [22])

Komunikace procesoru s pamětí počítače probíhá řádově v rychlosti několika GB/s, zatímco paměť grafické karty bývá v tomto ohledu mnohokráte rychlejší, protože je potřeba zásobovat stream jednotky grafické karty velkým množstvím dat.

Je tedy zřejmé, že přesun dat z lokální paměti procesoru do paměti grafické karty a nazpět naráží na problém rychlosti sběrnice propojující tyto komponenty. Tento problém lze řešit např. tím, že se přesune veškerý výpočet do grafické karty.

4.1.3 Podporované datové typy

Současné GPU zpravidla pracují se spojitým 3D prostorem, kde využívají výpočty v plovoucí řádové čárce. Z hlediska rychlosti jsou však podporovány zatím podporovány pouze datové typy s přesností 32bitů. Nejbližší budoucnost však slibuje i zavedení 64bitové přesnosti.

Kromě datových typů s plovoucí řádovou čárkou, jsou podporovány i datové typy s pevnou řádovou čárkou. Zpravidla jsou však tyto datové typy emulovány z datových typů s plovoucí řádovou čárkou.

Ukazatele současné GPU využívají pouze jako odkazy na data textur a programy, vytváření vlastních datových typů v GPU zatím není možné.

4.1.4 Výkonnost grafických karet

Jak již bylo uvedeno v kapitole zabývající se historií vývoje grafických karet, jejich výkon neustále stoupá. To s sebou přináší hlavně problém, jak tento výkon zvyšovat. Běžně se výkon zvyšoval tak, že se zvětšil počet tranzistorů na čipu, nebo se zvýšilo napětí, případně se zvýšil takt čipů.

čip	**počet tranzistorů** $\cdot 10^6$
AMD Athlon 64 X2 CPU	154
Intel Core 2 Duo CPU	291
ATi Radeon HD3800 GPU	666
nVidia G8800 GTX GPU	670
nVidia 8800 GT GPU	754
ATi Radeon HD4850 GPU	956
nVidia GTX280 GPU	1400
nVidia 9800 GX2 GPU	1500

Tabulka 4.2: Tabulka počtu tranzistorů na čipech ([1], [3], [2])

Avšak z technologického hlediska se ukázalo, že tento trend již není do budoucna možný [8]. Bylo třeba hledat jiné způsoby zvyšování výkonnosti nejen grafických čipů.

Jako nejlepší způsob se tedy logicky ukázalo provádět výpočty paralelně. Prakticky od počátku programovatelnosti grafických čipů se jevilo jako nejefektivnější optimalizovat výpočet pro zpracování typu SIMD[1]. Tento způsob provádění výpočtů je velmi dobře paralelizovatelný, neboť zpravidla neobsahuje žádné nebo jen malé datové závislosti.

Jak ukazuje Obrázek 4.1, teoretická výkonnost grafických karet neustále roste. Zvyšování výkonu se tedy vydává cestou masivní paralelizace výpočtů. Nové GPU tedy na svém čipu mají stovky jednotek, které jsou schopny provádět výpočet ve stejném čase. Tento přístup slibuje obrovský nárůst teoretického výkonu ovšem za cenu skutečně masivní paralelizace. Výpočet je tedy zpravidla rozdělen do mnoha výpočetních vláken. Je třeba si však uvědomit, že na jedno vlákno připadá menší výpočetní výkon. Vytváření a správa vláken je však plně v režii grafického hardware a tedy je režie vytváření vláken a přepínání kontextu téměř nulová. Tento fakt pochopitelně neplatí u vláken CPU.

4.1.5 Srovnání výhod a nevýhod použití GPU

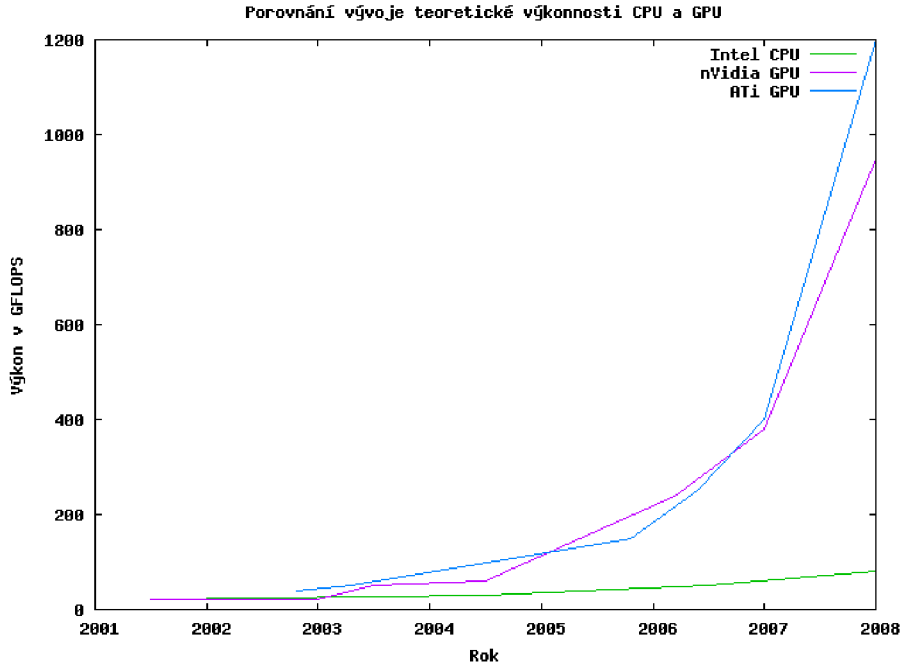
Výhody

- Obrovský teoretický výpočetní výkon
- Vysoká propustnost paměti
- Režie vláken realizována v GPU
- Masivně paralelní prostředí

Nevýhody

- Malá přesnost datových typů

[1] Single Instruction Multiple Data - tedy jedna operace nad mnoha daty



Obrázek 4.1: Srovnání vývoje výkonnosti GPU a CPU

- Nízký výkon na jedno vlákno
- Nemožnost vytváření vlastních datových typů

4.2 Běžně používané knihovny pro práci s GPU

S rozvojem počítačové grafiky, který započal v 70.tých letech minulého století se vyvíjely i grafické karty. Od 80.tých let se pak v počítačích využívají různé grafické symboly, ikony, obrázky proto, aby byla usnadněna komunikace mezi člověkem a počítačem. Od 90.tých let roste popularita 3D grafiky reprezentovaná hlavně počítačovými hrami, či používáním CAD systémů. Nejen pro potřeby tvůrců a uživatelů počítačových her, se na trh dostaly grafické knihovny. V současnosti existují 2 masivně používané grafické knihovny, a to OpenGL a DirectX.

4.2.1 DirectX

Vznik knihovny DirectX je spojen s rokem 1994, kdy na trh přichází operační systém Microsoft Windows 95. V době, kdy na poli osobních počítačů ještě převažoval systém MS-DOS, který umožňoval přímý přístup k zařízením jako grafická karta, myš, klávesnice, apod. začaly vznikat obavy z nového operačního systému, který již začal používat chráněný model paměti, který omezoval přístup k zařízením. Proto se Microsoft pustil do vývoje knihovny DirectX, která by umožňovala programátorům to, co starší MS-DOS.

První verze DirectX byla nakonec vypuštěna v září 1995 jako součást Windows Game SDK. Knihovna DirectX tedy umožňovala - počínaje operačním systémem Windows 95 - práci s multimédii v systému, začala se rovněž využívat pro tvorbu počítačových her.

Zatímco "konkurenční" knihovna OpenGL byla zaměřena na přenositelnost, kompatibilitu a byla hodně závislá na použitém hardwaru, DirectX se vydala jiným směrem - specializovala se hlavně na použití ve 3D hrách. Direct3D tedy byla jakási "light-weight" knihovna podporující pouze systémy Windows, naproti tomu OpenGL se vyvinulo v "cross-platform" knihovnu. Knihovna DirectX postupem času pronikla i do herních konzolí jako např. XBox.

V současné době se DirectX skládá z několika částí podle účelu [19]:

- DirectX Graphics (součásti DirectDraw a Direct3D) - pro podporu 3D vykreslování, grafiky
- DirectInput - podpora vstupních zařízení jako myš, joystick, apod.
- DirectPlay - podpora hraní více hráčů po síti
- DirectSound - podpora přehrávání a záznamu zvuku
- DirectMusic - podpora přehrávání a zpracování hudby
- DirectShow - podpora multimediálních aplikací, přehrávání a tvorby videa a zvuku
- DirectSetup - nástroj pro instalaci DirectX
- DirectX Media Objects - podpora pro tvorbu multimediálních efektů, kodeků, apod.

V době psaní této práce se připravuje vydání verze DirectX11. Poslední verzí knihovny DirectX, která byla určena pro operační systémy Microsoft Windows XP, byla verze 9.0c. S nástupem operačního systému Microsoft Windows Vista přišel i nový model WDDM (Windows Display Driver Model), který již není zpětně kompatibilní se systémem Microsoft Windows XP, a současně s ním i nová verze knihovny DirectX - DirectX 10. Zajímavostí okolo knihovny DirectX je, že verze DirectX 4.0 nebyla nikdy uvolněna, a tak verze 3.0 byla přímo nahrazena verzi DirectX 5.0. Další detaily lze najít v literatuře [19] či [12].

Od verze 8.0 podporuje DirectX (resp. Direct3D) vykreslování pomocí shaderů, k této činnosti využívá speciální jazyk HLSL (High Level Shader Language). Shadery se dají rozdělit na vertex shadery (zpracování každého vrcholu) a pixel shadery (zpravování pixelů).

4.2.2 OpenGL

Knihovna OpenGL je v současné době průmyslovým standardem pro vývoj grafických aplikací specifikující multiplatformní rozhraní. OpenGl nachází uplatně při tvorbě 3D aplikací, her či CAD systémů a jiných např. vědeckých aplikacích.

V 90.tých letech se do popředí na poli počítačové grafiky dostává konsorcium SGI (Silicon Graphics), jehož IRIS GL API se stalo de facto průmyslovým standardem a zastínilo otevřený standard PHIGS. SGI si brzy uvědomilo, že IRIS GL se není schopen stát otevřeným - kvůli problémům s licencemi a patenty. Navíc tato specifikace obsahovala i něco navíc, než jen soubor funkcí pro práci s 3D grafikou (např. klávesnice, myš, práce s okny). Tyto a ještě další aspekty (viz. [14]) vedly k vydání standardu **OpenGL**.

OpenGL standardizovalo přístup ke grafickému hardwaru. S tím souvisí zvýšení požadavků na tvorbu ovladačů grafického hardware, protože je potřeba dodržet specifikaci standardu. Obrovský přínos OpenGL je tedy v tom, že se používá na různých grafických kartách od různých výrobců stejný způsob komunikace s grafickým hardware, navíc je OpenGL od počátku navrhováno jako multiplatformní.

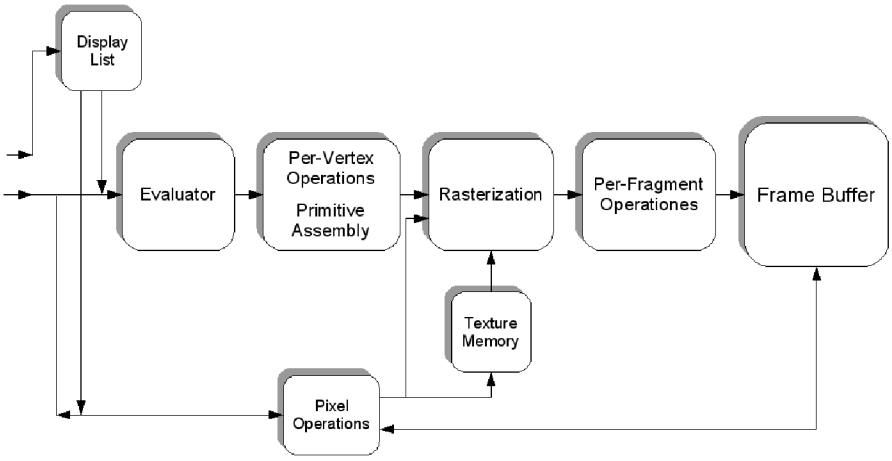
V roce 1992 společnost SGI stála i za vznikem OpenGL Architecture Review Board (OpenGL ARB), což je skupina společností, které mohou zasahovat do rozšíření samotného OpenGL standardu. Mohou např. přidávat vlastní konstanty, či vytvářet nové funkce. Proces obohacování standardu OpenGl o nová rozšíření probíhá v několika krocích. Každý člen má svou vlastní zkratku, která identifikuje navrhované rozšíření OpenGL standardu, např. nVidia používá `NV`.

V prvním kroku tedy některý ze členů sdružení přijde s navrhovaným vylepšením (za název vylepšení se přidá zkratka navrhovatele). Pokud je navrhované vylepšení schváleno ostatními, stává se z něj ve druhé fázi **rozšíření OpenGl** a přidává se zkratka `EXT`. Pokud je v další iteraci otestováno a znovu schváleno, může rozšíření proniknout do standardu OpenGL, pak se stává **standardním rozšířením** a dostává zkratku `ARB`, tedy např. `GL_ARB_Multitexture`. Rozšíření přináší drobnou komplikaci při vytváření programu, protože je potřeba explicitně ověřit, zda je dané rozšíření implementováno i na vlastní grafické kartě, na níž má aplikace fungovat.

Za zmínku stojí fakt, že standard OpenGL se nijak nezabývá prací s okny, se vstupněvýstupními zařízeními, tyto problémy jsou přenechány na rozšiřujících knihovnách, např. GLU, GLUT, SDL.

V roce 2006 se pak OpenGL stalo součástí konzorcia Khronos.

Knihovna OpenGL umožňuje vykreslování 5ti základních primitiv. Jsou to: bod, úsečka, polygon, bitmapa a pixmapa. OpenGL je založeno na architektuře klient-server, tj. klient zadává příkazy a server tyto vykonává. Vykreslování se děje pomocí tzv. graphics pipeline (grafický vykreslovací řetězec). OpenGL je tedy stavový stroj, tzn. provede se výběr barvy,



Obrázek 4.2: Graphics pipeline v OpenGL (převzato z [14])

s níž se bude vykreslovat a tato barva je platná až do doby, kdy je změněna na jinou barvu. Zadávání objektů k vykreslení probíhá tak, že se vybere, jaké primitivum se bude vykreslovat a poté se specifikují jednotlivé body tohoto primitiva. Tento způsob je zvláště pro rozsáhlejší a komplexnější objekty v dané scéně nepohodlný, proto byly zavedeny tzn. **Display listy**, které mohou obsahovat více primitiv k vykreslení. Poté již není třeba volat vykreslování jednotlivých primitiv, ale volá se pouze vykreslování přednastavených display listů.

Stejně jako Direct3D i OpenGL podporuje shadery. K tomu byl vyvinut jazyk GLSL

(OpenGL Shading Language). Chování shaderů je možné také popisovat pomocí jazyka Cg.

4.3 Specializované knihovny pro práci s grafickými kartami

Jak se ukázalo, výpočty prováděné v GPU mají perspektivu, což dalo vzniknout řadě knihoven, které jsou schopny abstrahovat grafický hardware a tím pádem umožňují jednodušší popis výpočtů v GPU.

Existují knihovny, které jsou zaměřeny přímo na konkrétní hardware jako nVidia CUDA či AMD Brook. Naproti nim existují další knihovny, které respektují platformní nezávislost a výpočty provádějí za pomocí shaderů, např. GPU Tech Ecolib. V současné době taktéž vzniká programovací jazyk - OpenCL. Výše uvedeným knihovnám pak budou věnovány následující podkapitoly[2]. Pro popis nVidia CUDA jsem vycházel hlavně z [4] a [5].

4.3.1 AMD Brook+

AMD Brook+ je součástí ATi Stream SDK. Jedná se o knihovnu podporující paralelní výpočty v grafických kartách firmy ATi (resp. AMD). AMD Brook+ využívá rozšíření jazyka C obohaceného o využití paralelních operací. Tato knihovna byla původně určena pouze pro využití na specializovaných grafických akcelerátorech FireStream, nicméně v současné době je zveřejněna pro volné použití na běžnějších desktopových řadách grafických akcelerátorů.

Pro mapování funkcí do GPU používá tzv. kernely, které budou diskutovány dále (4.3.3). Knihovna respektuje práci s daty formou SIMD[3], což (jak bylo uvedeno dříve) patří mezi stavební kameny pro provádění paralelních výpočtů v GPU.

4.3.2 OpenCL

OpenCL je framework, který je od počátku navrhován pro vytváření paralelních programů pro různé GPU, CPU a jiné typy procesorů. OpenCL obsahuje jazyk založený na jazyku C (opět tedy jakési rozšíření), který se pak využívá pro vytváření kernelů, které jsou prováděny na samotném hardwaru. Dále pak obsahuje API pro ovládání různorodých platforem. Přináší také možnosti pro paralelní výpočty. Myšlenkou je odstínit programátora od konkrétního hardware a nechat jej se plně zaměřit na řešení daného problému.

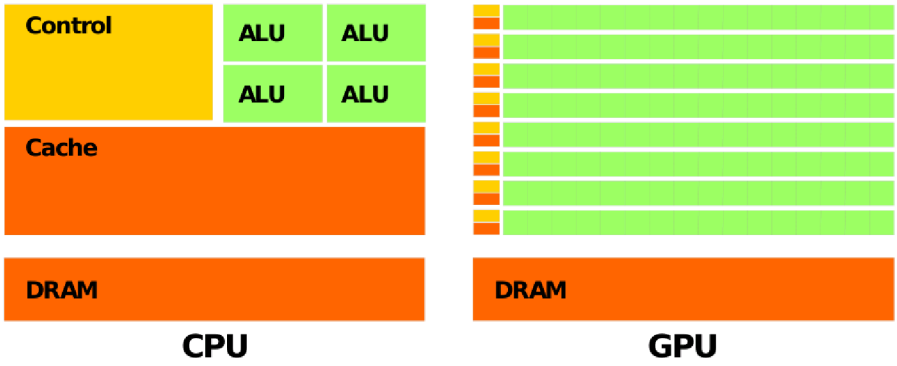
OpenCL získalo podporu u největších výrobců grafických čipů - společností ATi (AMD) a nVidia - a byla dokončena specifikace jazyka verze 1.0. Vzhledem k podpoře lze velmi brzy očekávat implementaci překladačů jazyka OpenCL pro grafické karty. Framework OpenCL tedy slibuje do budoucna jednotný platformně nezávislý jazyk pro popis paralelních výpočtů jak na procesorech, tak na grafických kartách.

4.3.3 nVidia CUDA

Zvyšující se požadavky na výpočetní výkon grafických karet ve 3D aplikacích vedly k tomu, že se běžné GPU vyvinuly v masivně paralelní, vícevláknové, více-jádrové procesory s obrovským výpočetním výkonem. Důvodem takovéhoto vývoje je fakt, že GPU se svými paralelními výpočty je přesně to, co moderní real-time grafika potřebuje pro dostatečnou rychlost vykreslování.

[2] Z hlediska důležitosti architektury CUDA pro tuto práci by bylo vhodné tuto část zahrnout jako kapitolu na vyšší úrovni hierarchie, nicméně z hlediska logického členění práce i CUDA patří do kapitoly o specializovaných knihovnách pro grafické karty, proto se v dalším textu budu držet tohoto členění práce.

[3] Single Instruction Multiple Data



Obrázek 4.3: Porovnání CPU a GPU

Obrázek 4.3 demonstruje rozdíly mezi architekturou CPU a GPU, kde v GPU je použito vícenásobné množství tranzistorů pro výpočty nad daty, než je tomu u běžných CPU. GPU je tedy velmi dobře navrženo pro výpočty datově paralelních problémů. Jedná se tedy o přístup, kdy se jeden program provádí nad mnoha daty[4]. Na základě výše uvedeného tedy vznikla architektura CUDA.

nVidia CUDA (Compute Unified Device Architecture) je relativně nová paralelní výpočetní architektura vyvíjená firmou nVidia. CUDA je tedy výpočetní nástroj pro GPU firmy nVidia, který je vývojáři přístupný přes standardní programovací jazyky, jako jsou C či C++ [10] a v současné době je již portována do jazyků jako java či Python.

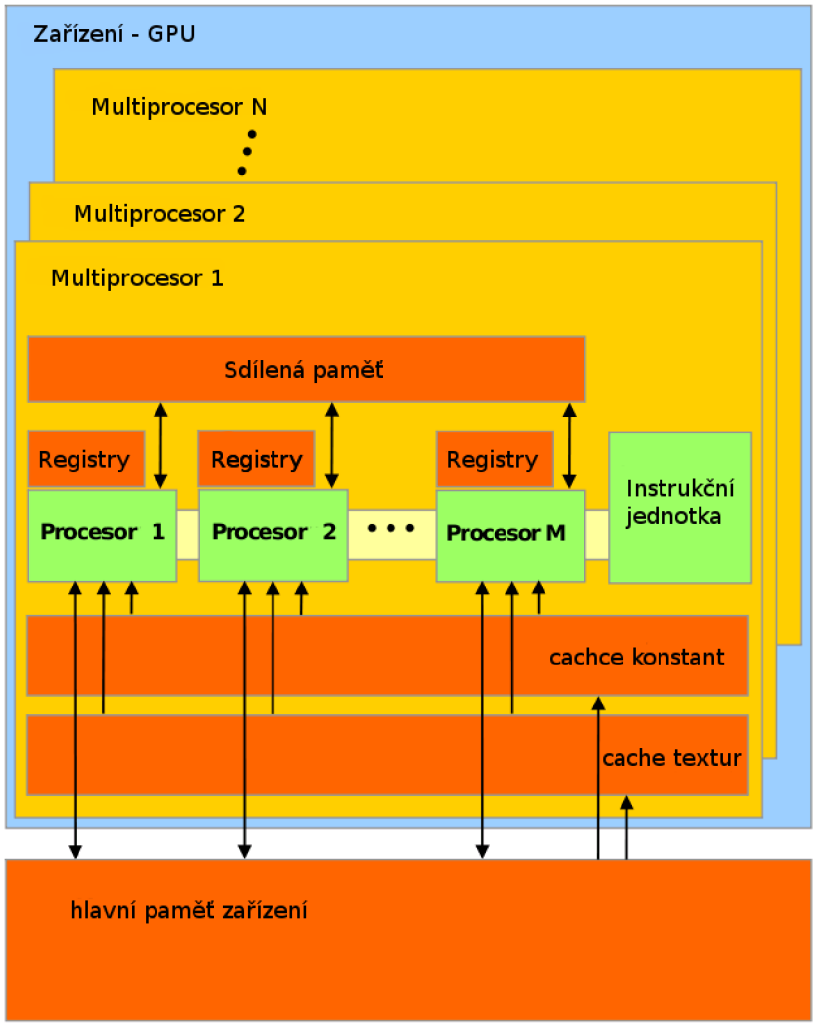
CUDA byla firmou nVidia představena v roce 2006 jako paralelní architektura s novým paralelním programovacím modelem a instrukční sadou zaměřenou na paralelní provádění výpočtů v GPU. Vývoj CUDA byl podřízen tomu, aby za cenu relativně malého rozšíření běžného programovacího jazyka[5] poskytovala jednoduchou a přímočarou implementaci paralelních algoritmů. CUDA rovněž podporuje heterogenní výpočetní model, takže aplikace může využívat jak výpočtů v CPU, tak výpočtů v GPU. CPU a GPU jsou od sebe vzájemně odděleny a chovají se každé jako samostatné programovatelné zařízení a mají své vlastní paměťové modely (prostory). GPU, které je schopno využívat CUDA, sestává tedy z mnoha jader, které mohou současně provádět mnoho výpočetních vláken. Sdílená paměť na čipu pak umožňuje vláknům běžícím paralelně sdílet data bez nutnosti jejich posílání po systémové sběrnici.

Výpočetní model CUDA

Nyní je vhodné se zaměřit na důležité pojmy architektury CUDA[6].

- **Host**
  Pojem *host* v architektuře CUDA označuje CPU. Zde tedy běží program např. v jazyce C.
- **Device**
  Pojem *device* pak označuje GPU, v němž bude prováděn paralelní program - *kernel*.

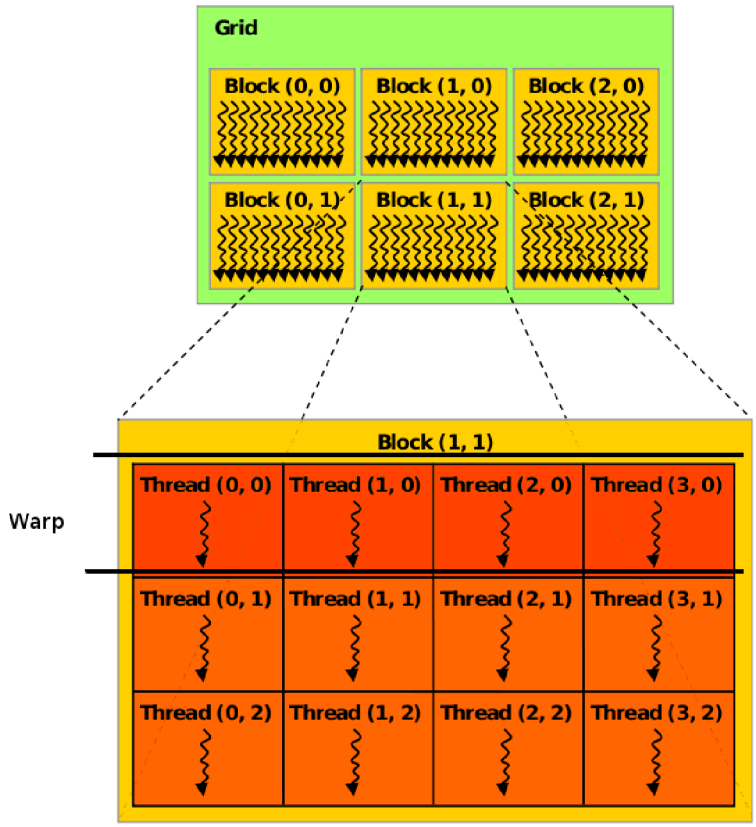
[4] přístup SIMD - jedna sada instrukcí nad mnoha daty
[5] Např. jazyka C
[6] V následujícím textu budu používat anglické termíny tak, jak jsou uvedeny v dokumentaci CUDA tam, kde nebyly zavedeny české ekvivalenty.



Obrázek 4.4: CUDA - hardwarový model, viz [4]

- **Kernely (Kernels)**
  CUDA přináší programátorovi možnost definovat funkci - *kernel*, která je po zavolání vykonána $N-$krát paralelně N různými CUDA *vlákny*. V zápisu programu vypadá jako běžná funkce jazyka C. Definice *kernelu* je je uvozena klíčovým slovem `__global__`, poté zpravidla následuje klíčové slovo `void` a název funkce. Počet vláken je pak specifikován pro každé volání kernelu za použití "závorek" `<<<...>>>`.

```
//definice kernelu
__global__ void ScitejMatice(float *a, float *b, float *c){
    int i = threadIdx.x;
       c[i]  = a[i]+b[i];
}
...
int main(int argc, char **argv){
...
```



Obrázek 4.5: CUDA - hierarchie vláken, viz [4]

```
    ScitejMatice<<<1, N>>>(A, B, C); //volání kernelu
...
}
```

Kernel tedy specifikuje program, který se bude v grafické kartě provádět.

- **Vlákno (Thread)**
  Vlákno je jakousi nejmenší možnou jednotkou provádění v architektuře CUDA. Pro tato vlákna platí výše uvedený předpoklad o velmi malé až téměř nulové režii. Na druhou stranu výkon samostatného vlákna je velmi malý, proto, aby byla využita potenciální výpočetní kapacita GPU, je zapotřebí mnoha (až tisíců) vláken, která běží souběžně.

  Každé vlákno prováděné kernelem má svou jednoznačnou identifikaci. Tato identifikace je přístupná v kernelu skrze vestavěnou proměnnou `threadIdx`, což je zpravidla 3-složkový vektor, takže vlákna mohou být identifikována jedno až tří-dimenzionálním indexem[7] vlákna. Vlákna mohou být dále seskupována do *bloků vláken*, které mohou být opět až tří-dimenzionální.

[7] thread-index

- **Warp**
  Vlákna jsou v rámci jednoho bloku seskupena do tzv. *wapru*. Je to vlastně skupina vláken v rámci bloku, které provádějí stejnou operaci nad různými daty. Každý warp obsahuje stejný počet vláken[8], která jsou pak prováděna multiprocesorem. Aktivním warpům jsou v závislosti na čase plánovačem přepínány pro maximalizaci výpočetního výkonu. [5]

- **Blok vláken (Block)**
  Blok je skupina warpů, která je vykonávána na jednom multiprocesoru. Jeden kernel může být prováděn několika bloky vláken, proto jsou tyto bloky ještě sdruženy v tzv. *gridu*. Stejně jako u vláken, každý blok má přiřazenu jednoznačnou identifikaci v rámci gridu. Tato identifikace je přístupná ve vestavěné proměnné `blockIdx` a rovněž lze získat i "rozměry" daného bloku, ty jsou obsaženy ve vestavěné proměnné `blockDim`. U bloků je požadováno, aby mohly být prováděny v náhodném pořadí, a tedy paralelně.

- **Grid**
  Grid[9] je tedy seskupení bloků v rámci jednoho kernelu. Grid je obvykle jedno či dvourozměrný, jeho rozměry jsou specifikovány v prvním parametru v závorkách `<<<...>>>` při volání kernelu. Následuje příklad zdrojového kódu CUDA pro součet dvourozměrných matic s využitím dříve popsaných struktur.

```
//Definice kernelu
__global__ void MatAdd(float A[N][N], float B[N][N],float C[N][N]){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
      C[i][j] = A[i][j] + B[i][j];
    }

int main(){
    // Nastavení kernelu
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x,
                 (N + dimBlock.y - 1) / dimBlock.y);
    MatAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

Právě popsaná hierarchie vláken je uvedena na obrázku 4.5. Nyní se zaměřím na paměťový model multiprocesoru GPU.

- **Registry**
  Registry představují nejrychlejší paměť na jednom multiprocesoru. Tento typ pamětí je přístupný pouze pro dané vlákno. S tím souvisí fakt, že životnost dat je přímo navázána na dobu života daného vlákna. V uváděné hierarchii jsou tedy registry GPU tou nejrychlejší pamětí, avšak mají též nejmenší kapacitu[10]

[8] warp-size
[9] Nabízí se i české označení - mřížka
[10] Stejně je tomu v případě CPU.

- **Sdílená paměť**
  Sdílená paměť je paměť přítomná na čipu, proto je rychlost sdílené paměti rovněž velmi vysoká. Do sdílené paměti mají přístup všechny procesory bloku jak pro zápis, tak pro čtení. Dokumentace uvádí, že pro všechna vlákna v jednom warpu je přístup do sdílené paměti téměř stejně rychlý jako přístup do registrů. Nesmí ovšem nastat konflikty mezi jednotlivými paměťovými banky, což jsou části sdílené paměti, které mají stejnou velikost a jsou současně přístupné pro všechna vlákna warpu.

- **Paměť textur a konstant**
  Obě tyto paměti se vyznačují tím, že obě slouží jako rychlá vyrovnávací paměť mezi procesory a mají nízkou přístupovou dobu a velmi dobrý hit-ratio.

- **Globální paměť**
  Globální paměť je nejpomalejší pamětí v uváděné hierarchii. Je přístupná jak pro hostitele (host), tak pro zařízení (device). Její životnost je stejná jako životnost dané aplikace. Zdroj [6] uvádí, že globální paměť může být až 150x pomalejší než paměť sdílená a registry.

CUDA rozšiřuje jazyk C o možnosti provádění výpočtů paralelně. K tomu je zapotřebí definovat jistý druh označení funkcí a proměnných. Funkce se mohou provádět jak na zařízení (GPU), tak v hostiteli. Tyto skutečnosti je potřeba rozlišovat, proto uvádím nejdůležitější specifika.

- **`__device__` funkce**
  Tento kvalifikátor označuje funkci, která může být provedena pouze na daném zařízení a nelze ji volat z hostitelského programu, jediná možnost volání je z programu zařízení. Pro tento typ funkcí existují určitá omezení: není podporována rekurze, funkce musí mít konstantní počet parametrů a není možné, aby tento typ funkcí ve svém těle definoval statickou proměnnou. Kvalifikátor nesmí být použit společně s kvalifikátorem `__global__`.

- **`__global__` funkce**
  Tímto kvalifikátorem je označena funkce - kernel. Takovouto funkci je možné zavolat pouze z hostitelského programu a je vykonávána výhradně v zařízení. Pro kernely platí stejná omezení jako pro funkce s kvalifikátorem `__device__`, tj. omezení týkající se rekurze, konstantního počtu parametrů a vytváření statických proměnných. Dále není možné společně použít kvalifikátor `__host__`. Funkce označené jako `__global__` - kernely - **musí** mít návratovou hodnotu typu `void`. Volání kernelu je asynchronní, tzn. že vrací řízení hostitelskému programu ještě před tím, než bylo dokončeno provádění daného kernelu.

- **`__host__` funkce**
  Tento kvalifikátor označuje funkci, která je volána jen a pouze z hostitelského programu a rovněž je na hostitelském zařízení prováděna. Zajímavostí je, že kvalifikátory `__host__` a `__device__` mohou být použity společně v deklaraci funkce, taková konstrukce pak znamená, že kód funkce bude přeložen jak pro hostitelskou architekturu, tak pro provádění v zařízení.

- **`__device__` proměnná**
  Je vytvořena v pamětí daného zařízení a existuje stejně dlouho jako program. Je přístupná ze všech vláken gridu a také z hostitele skrze knihovní volání.

- **`__constant__` proměnná**
  Kvalifikátor může být použit společně s kvalifikátorem `__device__`, deklaruje proměnnou, a to takovou, která je uložena v prostoru paměti konstant. Její životnost je stejná jako životnost aplikace a je přístupná ze všech vláken gridu a přes knihovní volání také z hostitele. Do takovéto proměnné může přiřazovat pouze hostitel, opět skrze knihovní volání.

- **`__shared__` proměnná**
  Kvalifikátor `shared` může být použit společně s kvalifikátorem `device`, deklaruje proměnnou, která je uložena v paměti bloku vláken, její životnost je tedy stejná jako životnost bloku a je přístupná pouze a jen z daného bloku vláken.

Psaní programu s pomocí CUDA se tedy velmi podobá psaní běžného programu v jazyce C pouze s uvedenými rozšířením. Program napsaný v CUDA může běžet na grafické kartě, nebo lze použít emulovaný režim, např. pro ladění, nebo v případě, že není k dispozici odpovídající hardware. Je jasné, že při použití emulovaného režimu se dosahuje podstatně menšího výpočetního výkonu, než kdyby výpočet probíhal v grafické kartě. Proto sám výrobce nedoporučuje emulovaný režim používat hlavně pro výkonnostní ladění dané aplikace.

Kapitola 5

Návrh programu v CUDA

Součástí této diplomové práce je vytvoření programu s požitím prostředí CUDA, který bude simulovat děje probíhající v homogenním a nehomogenním vedení. Ve svých experimentech s prostředím CUDA jsem vycházel z příkladů jednoduchých programů dodaných přímo s prostředím CUDA.

5.1 Počáteční analýza

Diferenciální rovnice popisující jednotlivé elementy vedení lze simulovat využitím spojité simulace. Proto bude potřeba tento algoritmus implementovat. Pseudokód algoritmu spojité simulace ukazuje následují blok.

```
inicializace, nastavení počátečních podmínek integrátorům
while(t < tEND){
       vyhodnocení vstupů integrátorů;
       provedení výpočtu;
       t = t+h - posun času o daný krok h
}
```

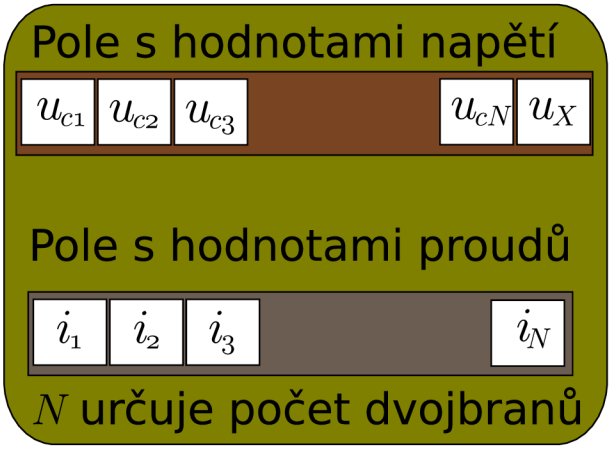
Z povahy řešeného problému vyplývá, že právě posloupnost pseudo-příkazů z bloku `while` se bude provádět paralelně. Uvažujme nyní zápis programu v TSKL z přílohy B. Pokud se povede najít způsob, aby bylo možné jednotlivé rovnice provádět nezávisle na sobě, pak bude problém paralelizovatelný.

Proto uvažujme jeden některý krok výpočtu. V zápisu programu se nevyskytují rychlé smyčky, které by výpočet zkomplikovaly. Navíc, v době, kdy se budou počítat hodnoty pro daný krok výpočtu jsou k dispozici hodnoty pro krok předchozí[1]. Proto není problém tento výpočet paralelizovat.

V návaznosti na prostředí CUDA bude důležitým parametrem počet vláken, který v tomto případě bude reprezentovat počet dvojbranů simulovaného vedení. Je třeba si uvědomit, že tento postup je možný pouze pro simulaci homogenního vedení.

V případě, že se bude simulovat chování vedení nehomogenního, je potřeba vyřešit problémy nehomogenit, jako např. měnící se konstanty.

[1] Resp. počáteční hodnoty pro krok první v čase $t = 0$



Obrázek 5.1: Návrh datových struktur

5.2 Návrh datových struktur

Nyní je potřeba navrhnout použitelné datové struktury. Současné možnosti prostředí CUDA dovolí používat pouze čísla v plovoucí řádové čárce s přesností 32bitů, tj. typu `float`, což může vést ke znepřesnění výpočtů a jeho případné degradaci. Vzhledem k tomu, že se vypočítávají 2 diferenciální rovnice pro každý dvojbran, nabízí se zavést abstrakci v podobě buď dvou jednorozměrných polí, přičemž jedno bude reprezentovat výsledná napětí a druhé pak výsledné proudy. Druhá možnost je požít jedno dvojrozměrné pole a rozhodnout, na kterých indexech se budou nacházet jednotlivé proměnné reprezentující jednotlivá napětí a proud.

Vycházíme-li z předpokladu, že kernely jsou v rámci CUDA spouštěny asynchronně, nabízí se možnost rozdělení výpočtů napětí a proudů do dvou nezávislých kernelů jako efektivnější varianta, proto jsem se rozhodl ji použít. Parametrem velikosti polí je počet dvojbranů reprezentujících vedení. Situaci znázorňuje následující obrázek 5.1.

Z filozofie výpočtu plyne, že celkově budou potřeba 4 takovéto pole, a to 2 pro uchování vstupních hodnot napětí u_{c_x} a i_{c_x} a dvě pole pro uložení výstupních hodnot. Pro zjednodušení a zefektivnění výpočtu je v poli pro hodnoty napětí ještě přidána položka napětí u_x, která popisuje průběh napětí na rezistoru, které je připojen na konci vedení. Tato hodnota se však nebude počítat v kernelech, tuto hodnotu je možno předpočítat před spuštěním kernelů.

Rovněž by pole hodnot mohla obsahovat napětí u_1, toto je opět zbytečné, neboť napětí u_1 se použije pouze pro výpočet počátečního proudu i_0 a v žádné další rovnici již nefiguruje. Situace u hodnot proudu i_0 je podobná. Hodnotu této veličiny lze opět předpočítat, její hodnota se použije pouze při výpočtu rovnic v prvním dvojbranu, proto rovněž nebude zahrnuta v jednotlivých polích a do kernelu se předá jako běžný parametr a v kernelu se bude potřeba provést test minimální index vlákna. Tato analýza je opět vytvořena na základě Přílohy B.

5.3 Návrh samotného programu

Na základě provedené analýzy lze přejít k návrhu samotných kernelů a celého programu v CUDA. Jako první uvádím kernel pro výpočet nových hodnot napětí.

```
01 __global__ void compute_U(float * u_input, float *u_output,
02                           float *i_field,  float C_to_m_1,
03                           float Rp_to_m_1,  float step, float i0){
04        const unsigned int tid = threadIdx.x;
05        ///uc1' = (1/C)*(i0 - (1/Rp)*uc1 - i1)
06        if(tid == 0){
07                u_output[tid] = u_input[tid] + step*(
08                       C_to_m_1 *(i0 - (Rp_to_m_1*u_input[tid])
09                         -i_field[tid] ) );
10        }
11        else{
12                u_output[tid] = u_input[tid] + step*(
13                       C_to_m_1 *(i_field[tid-1] - (Rp_to_m_1*u_input[tid])
14                       -i_field[tid] ) );
15        }
16        __syncthreads();
17 }
```

Proměnná `tid` definovaná na řádku 4 reprezentuje jednoznačnou identifikaci vlákna a je jí použito pro výpočty nad polem hodnot.

V případě, že se se jedná o vlákno s indexem 0, pak se jedná o výpočet hodnot pro první dvojbran a je tedy nutné vzít v úvahu proměnnou `i0`, jak bylo uvedeno v předchozí kapitole, v opačném případě se použije hodnota z pole proudů `i_field` na patřičné pozici. Nově vypočtená hodnota je pak uložena do výstupního pole na patřičný index, jak ukazují řádky 7 a 13.

Význam parametrů `[C,Rp]_to_m_1` značí hodnoty konstant[2] $\frac{1}{C}$ a $\frac{1}{Rp}$.

Kernel pro výpočet nových hodnot proudů bude vypadat velmi podobně.

```
01 __global__ void compute_I(float * i_input, float *i_output,
02                           float *u_field, float L_to_m_1,
03                           float Rs,  float step){
04         const unsigned int tid = threadIdx.x;
05         const unsigned int tid_plus_one = threadIdx.x+1;
06         i_output[tid] = i_input[tid] + step* (L_to_m_1 *(u_field[tid]
07                          -u_field[tid_plus_one] - (Rs*i_input[tid]) ) ) ;
08         __syncthreads();
09 }
```

Jak je vidět, kernel pro výpočet nových hodnot proudů je o něco jednodušší, než předchozí. Opět se zde snažím co nejvíce konstant předpočítat z důvodu urychlení výpočtu (`L_to_m_1`). Proměnná `step` reprezentuje v obou případech krok výpočtu. Na konci každého kernelu je potřeba provést synchronizaci vláken, může se totiž stát, že doby výpočtů se budou lišit. Navíc jsou kernely volány asynchronně a celá simulace běží v cyklu, mohlo by tedy dojít k nechtěnému přepisu dat v jednotlivých krocích výpočtu.

[2] viz Příloha B

Samotný program zapsaný v pseudokódu pak bude vypadat následovně

```
void runTest( int argc, char** argv)
{
      ...
      unsigned int rovnic = 5;  //pocet rovnic
      definice vstupních výstupních polí a jejich inicializace
      ...
      float time = 0.0; //modelový čas
      float t_END = (float)1e-06; //konec experimentu
      float step = (float)1e-12; //krok
      ...
      definice konstant a nastavení gridu
      ...
      while(time < t_END){
           předvýpočet hodnot uX a i0
           u_data[uX_pos] = uX; //umistim hodnoty do pole vstupu
           u_o_data[uX_pos] = uX; //i do pole vystupu, uX se nikde nemeni
           //volani kernelů
           compute_U<<<ui_grid, ui_threads, mem_u>>>(u_data, u_o_data,
                                    i_data, C, Rp, step, i0);
           compute_I<<<ui_grid, ui_threads, mem_i>>>(i_data, i_o_data,
                                      u_data,  L, Rs,  step);
           ...
           time+=step; //posun času
      }
      ...
      uvolnění paměti
      cudaThreadExit();
}
```

Zbývá dodat, že pro návrh programu byla použita Eulerova metoda řešení diferenciálních rovnic. Uvedený algoritmus vychází z dříve uvedeného algoritmu spojité simulace. V pseudokódu nejsou zahrnuty funkce pro kopii lokáních dat do paměti zařízení.

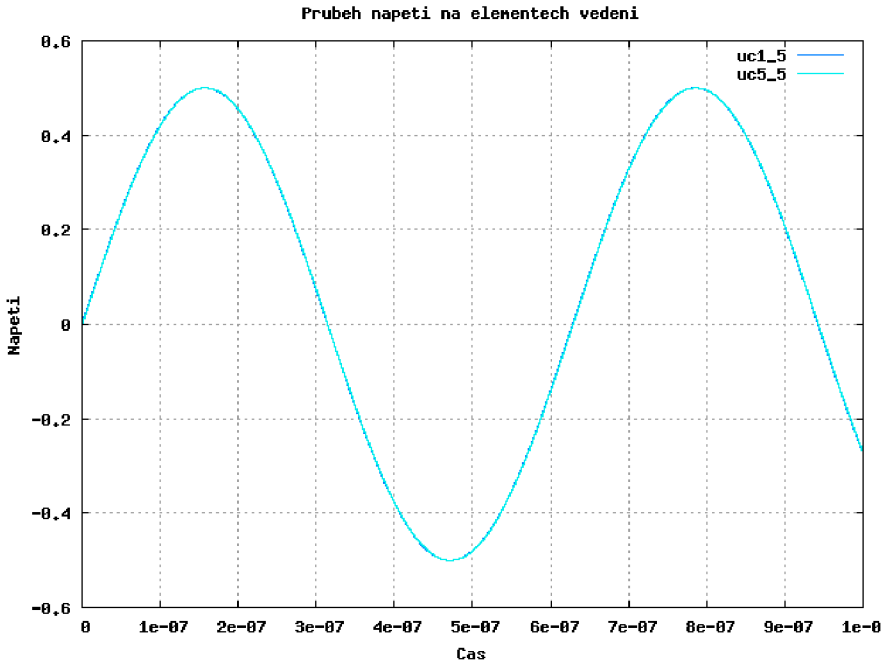
5.4 Zhodnocení použití prostředí CUDA

S navrženým programem popsaným v předchozí kapitole jsem provedl několik experimentů. Jako první jsem zkoušel, zda bude takto sestavený výpočet fungovat pro vedení tvořené pouze jedním dvojbranem. Ukázalo se, že program dával přibližně tytéž výsledky jako TKSL, proto jsem zvýšil počet dvojbranů (a tedy rovnic) na 5.

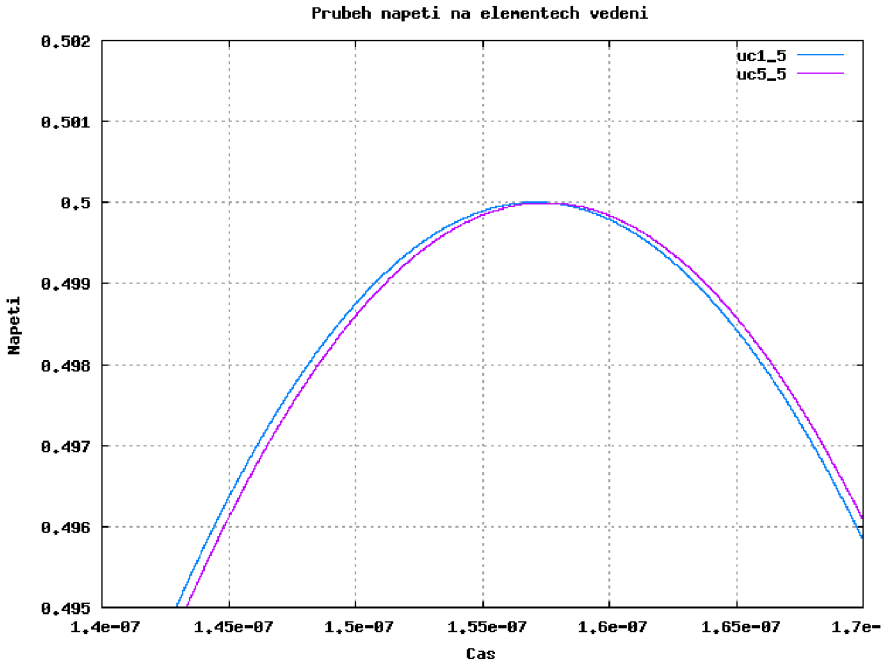
Na grafech 5.2 a 5.3 jsou zachyceny vypočtené průběhy napětí na prvním u_{c1} a posledním u_{c5} elementu vedení. Tyto výsledky jsou srovnatelné s výsledky z programu TKSL, nicméně i zde už se projevily drobné nepřesnosti ve výpočtu. Tyto nepřesnosti jsou způsobeny jednak volbou numerické metody (Eulerova) a také zaokrouhlovacími chybami v průběhu výpočtu.

Protože výsledky lze označit za relativně dobré, zvýšil jsem počet rovnic na 100 a opět zkoumal obdržené výsledky. Výsledky tohoto experimentu znázorňuje graf 5.4.

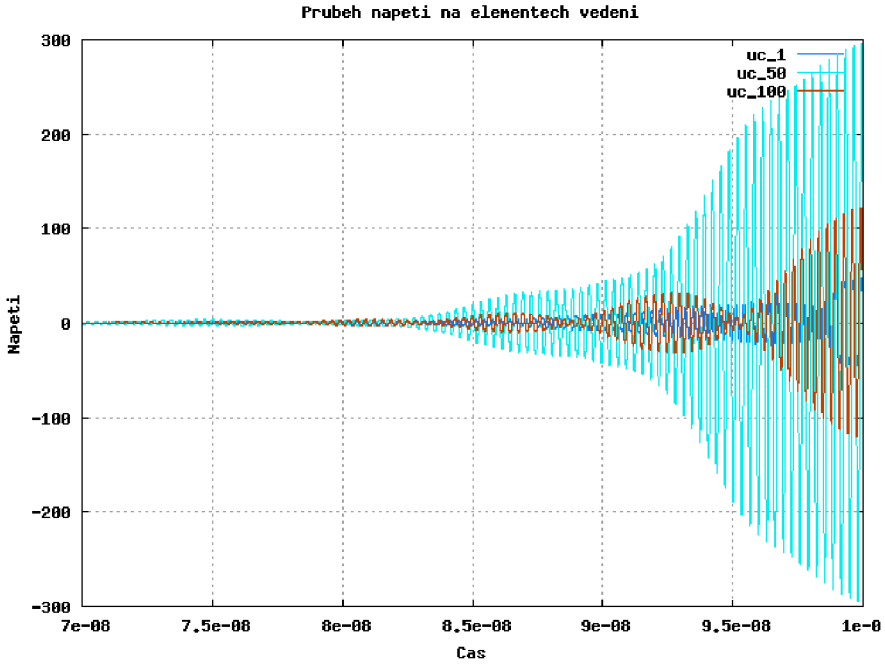
Jak je z grafu vidět, průběhy napětí jsou vysoce nestabilní, naplno se projevily chyby metody a použitá přesnost čísel v plovoucí řádové čárce typu `float`. Jak již bylo uvedeno



Obrázek 5.2: Průběhy napětí u_{c1} a u_{c5}



Obrázek 5.3: Detail průběhu napětí



Obrázek 5.4: Průběhy napětí pro 100 dvojbranů

v kapitole 4.3.3, CUDA v současnosti používá pouze typ `float` a typ `double`, který by se lépe pro tento druh výpočtů hodil, je na tento typ interně konvertován.

Dále jsem zkoušel zmenšovat délku kroku, což ovšem nepřineslo žádné rozumné výsledky. Experimentálně se ukázalo, že maximální možný počet rovnic, který ještě dával stabilní řešení, byl 5 rovnic, tedy toto řešení bylo schopno analyzovat chování 5ti dvojbranů.

Celkově tedy lze říci, že z hlediska stability takto napsaný program s použitím CUDA nevyhovuje pro větší počty řešených rovnic. Z hlediska přesnosti výpočtu CUDA rovněž není momentálně vhodným nástrojem, protože použitá maximální přesnost čísel v plovoucí řádové čárce je pro tento typ výpočtu nedostačující.

Možným teoretickým řešením by mohlo být použití vlastní aritmetiky, nicméně tato problematika je nad rámec této práce.

Kapitola 6

Srovnání se světovými standardy

Při srovnávání práce v systému CUDA a TKSL jsem se zaměřil i na srovnávání se standardy dnešní doby, které tvoří převážně produkty Matlab a Maple. V případě obou produktů jde o vysoce ceněná prostředí, která jsou schopna řešit širokou škálu problémů.

6.1 Zadávání a řešení diferenciálních rovnic

Nejprve se zaměřím na způsob zadávání diferenciálních rovnic do jednotlivých systémů.

6.1.1 TKSL/386 a TKSL/c

V případě nástroje TKSL, je tato problematika jistě patrná z předchozího textu. Diferenciální rovnice se intuitivně zapíší - zápis jako takový se neliší v prostředí TKSL/386 a TKSL/c, nicméně "program" pro TKSL/c je poněkud zjednodušen - odpadají deklarace proměnných a přechází se rovnou k zadávání problému. Takto tedy vypadá program v prostředí TKSL/386:

```
var u1,
    uc1,
    i2,i1,
    ic1,
    u2;
const R1=100,    R2=100,
      Rs=1e-3,   Rp=1e+10,
      C =1e-12,  L =1e-8,
      dt=1e-5,   tmax=1e-6,
      eps=1e-10, omega=1e7;

system
i1=1/R1*(u1-uC1);
uc1'= 1/C*ic1            &0;
...
sysend.
```

V prostředí TKSL/c pak odpadá část deklarace proměnných *var* a z předchozího příkladu tak zůstává zachována pouze sekce *const*, avšak bez tohoto klíčového slova a bez konstant *tmax*, *eps* a *dt*, které se zadávají jako parametry příkazové řádky. Odpadají rovněž

systémové závorky *system* a *sysend*. Zápis samotných diferenciálních rovnic se pak již neliší.

6.1.2 Matlab

V prostředí Matlab je výhodné zadávat diferenciální rovnice formou blokového schematu. Zde se použijí stavební bloky jako integrátor, sumátor, invertor, apod. Sestavené schema se pak dá velmi jednoduše odsimulovat. U všech použitých komponent lze nastavit, zda budou tyto využívat invertované či neinvertované vstupy.

Uvažujme příklad, kdy budeme chtít generovat funkci sinus za pomoci diferenciální rovnice (resp. soustavy diferenciálních rovnic). Lze psát

$$\begin{aligned} y &= sin(t) \\ y' &= cos(t) \\ y'' &= -sin(t), \end{aligned} \tag{6.1}$$

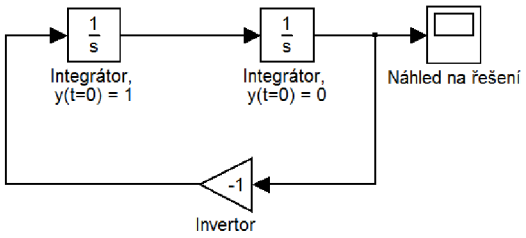
což lze upravit na

$$y'' = -y. \tag{6.2}$$

Dostali jsme tedy tzv. tvořící diferenciální rovnici. Tuto diferenciální rovnici druhého řádu lze řešit např. metodou snižování řádu derivace. Pro názornost je diferenciální rovnice 6.2 ještě převedena na operátorový tvar. Metodou snižování řádu derivace dostaneme

$$\begin{aligned} py &= -\frac{1}{p}p^2y \\ y &= \frac{1}{p}py \end{aligned}$$

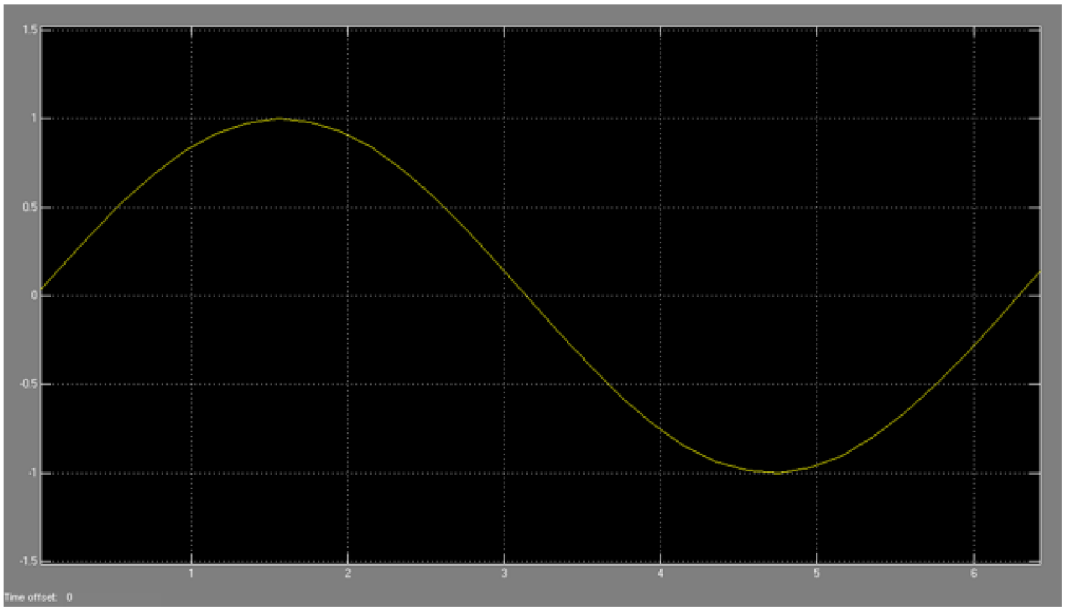
s počátečními podmínkami $py(0) = 1$ a $y(0) = 0$. Pro takovouto soustavu diferenciálních rovnic již není problém sestavit blokové schema v prostředí Matlab, viz. blokové schema na obrázku 6.1. Řešení takovéto diferenciální rovnice zadané blokovým schematem je pak



Obrázek 6.1: Blokové schema diferenciální rovnice generující funkci sinus

v grafu 6.2.

Matlab jako takový však pro řešení, na rozdíl od prostředí TKSL, nepoužívá Taylorova rozvoje a nedosahuje takové přesnosti řešení, jako právě systém TKSL.



Obrázek 6.2: Graf řešení generující funkce sinus

6.1.3 Maple

Prostředí Maple se na první pohled od prostředí Matlab liší. Jedná se hlavně o matematický software, není zde problém řešit soustavy n rovnic o n neznámých, apod. Maple rovněž zvládá řešení integrálů, derivací i diferenciálních rovnic. Pro potřeby řešení Maple obsahuje i sadu příkazů pro vizualizaci spočtených výsledků.

Uvažujme tedy příklad diskutovaný v předchozí kapitole zabývající se prostředím Matlab, tj. řešením diferenciální rovnice druhého řádu

$$y'' + y = 0 \tag{6.3}$$

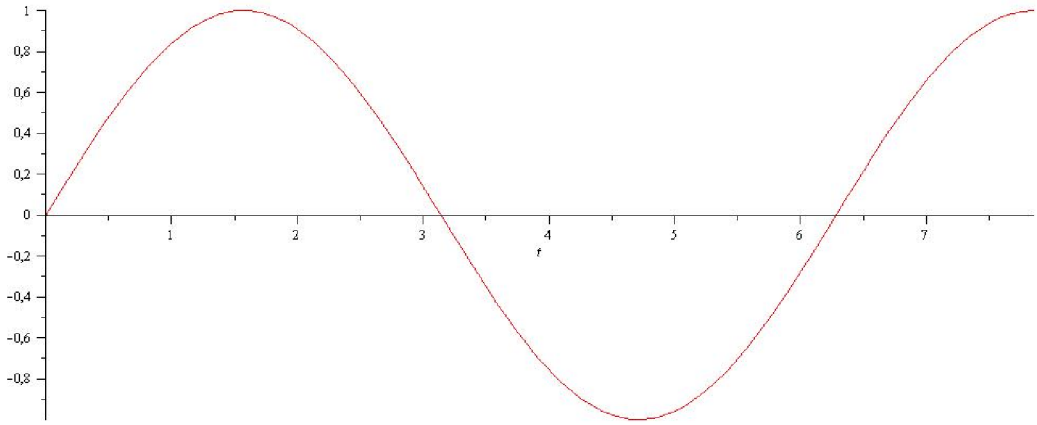
s počátečními podmínkami $y(0) = 0$ a $y'(0) = 1$. Zápis tohoto problému v prostředí Maple by mohl vypadat následovně:

```
> restart;
> de := diff(y(t), t$2) + y(t) = 0;
> Y := rhs( dsolve( {de, y(0) = 0, D(y)(0) = 1}, y(t) ) );
> plot(Y, t = 0..(5*Pi/2));
```

Výsledný graf řešení této diferenciální rovnice je pak zobrazen v Obrázku 6.3.

6.2 Numerické řešení (soustav) diferenciálních rovnic

Všechna zde diskutovaná prostředí disponují nástroji pro řešení diferenciálních rovnic a jejich soustav. Prostředí Maple navíc přidává možnost symbolického řešení diferenciálních rovnic. Pro tuto práci je však nejdůležitější partií numerické řešení diferenciálních rovnic, které je možné mnoha metodami (Eulerova metoda, metody Runge-Kutta, Taylorova metoda, Adams-Bashforth, apod.). Použitá metoda determinuje přesnost daného výpočtu. Srovnání jednotlivých prostředí z tohoto pohledu představuje následující tabulka 6.2.



Obrázek 6.3: Graf řešení generující funkce sinus

Prostředí	Použitá funkce	Numerická metoda
Maple	dsolve	Runge-Kutta 4.-5. řádu
Matlab	ode23	Runge-Kutta 2.-3. řádu
Matlab	ode45	Runge-Kutta 4.-5. řádu
TKSL/386	-	Taylorův rozvoj (běžně 10. - 60. řád)
TKSL/c	-	Taylorův rozvoj (vyzkoušen do řádu 2000)

Tabulka 6.1: Shrnutí vypočtených přímek

Z této tabulky tedy jasně vyplývá, že nejpřesnější výsledky bude dávat právě TKSL/C.

6.3 Shrnutí

Maple i Matlab jsou vysoce propracovaná prostředí a řešení diferenciálních rovnic je pouze jedna část ze široké sady nástrojů, které tato prostředí přinášejí. Orientace prostředí Maple je spíše na matematické výpočty, záběr Matlabu je v tomto směru širší. Obě prostředí jsou schopna vizualizovat vypočtené výsledky.

Naproti tomu systém TKSL[1] je zaměřen hlavně na náročné a přesné výpočty. Je to tedy oproti Maple a Matlabu specializovanější software, neexistují do něj rozšiřující pluginy apod., ve verzi TKSL/C dokonce odpadla možnost vizualizace výpočtu (ovšem zvýšila se rychlost, přesnost) a jakékoli grafické uživatelské rozhraní - TKSL/C se ovládá výhradně z příkazové řádky.

V Matlabu i Maple je potřeba diferenciální rovnice zadávat přes další funkce, které mají různé parametry, které vyžadují určité nezbytné studium dokumentace. Naproti tomu TKSL nabízí velmi přímočarý a tedy jednodušší přístup, který již byl diskutován dříve.

Z hlediska použitých metod pro numerické řešení soustav diferenciálních rovnic je však TKSL předurčen k tomu, aby produkoval výsledky s největší přesností. Jak přesný může výpočet s TKSL/C být demonstruje Příloha C, kde je uveden příklad výstupu ze systému TKSL/C pro vedení tvořeného kaskádou 500 dvojbranů.

[1] Uvažujme obě verze - TKSL/386 a TKSL/C

Kapitola 7

Závěr

Při vytváření této práce jsem se seznámil s různými druhy diferenciálních rovnic, jejich klasifikací a hlavně s jejich využitím pro analýzu přechodných dějů na dvouvodičovém homogenním a nehomogenním vedení. Velmi přínosným se ukázalo na takovéto vedení pohlížet jako na běžný elektrický obvod tvořený součástkami, u nichž jsou známy charakteristiky jejich přechodných dějů.

S využitím těchto znalostí pak byly sestaveny modely vedení, nejprve byl definován náhradní dvojbran vedení, který je již výhradně tvořen pouze elektrickými součástkami. Z takovýchto dvojbranů byly postupně vytvářeny kaskády, které čítaly až 1000 dvojbranů, simulace takového vedení však byla neúnosně dlouhá a navíc se ukázalo, že stejnou vypovídací hodnotu pro analýzu přechodných dějů, má vedení tvořené 500dvojbrany, proto byly další experimenty prováděny s takovýmto modelem. Výsledky experimentů ukázaly různé zajímavé vlastnosti a závislosti, které na vedení vznikaly. Homogenní vedení poskytovalo relativně očekávané výsledky, s vedením nehomogenním byly rovněž provedeny určité experimenty a popis chování nehomogenních vedení bude předmětem dalšího výzkumu.

Dále jsem se seznámil s prostředími umožňujícími provádět složité výpočty v čipech grafických karet, a to paralelně. Jako nejzajímavější se jevila prostředí ATi Brook+ (ATi Stream) a nVidia CUDA. Byla diskutována omezení prostředí CUDA, výhody i nevýhody provádění výpočtů v grafických kartách.

Při implementaci programu v prostředí CUDA se naplno projevil problém malé použitelné přesnosti čísel v plovoucí řádové čárce. Dle aktuálního vývoje by další verze prostředí CUDA měly začít podporovat dvojitou přesnost [1] čísel v plovoucí řádové čárce, nicméně v současné době je CUDA pro přesné výpočty téměř nepoužitelná. Dochází k maximalizaci globálních chyb a výsledek celého výpočtu je vysoce degradován.

Na závěr práce jsem se snažil srovnat některá dostupná komerční prostředí pro řešení diferenciálních rovnic s prostředím TKSL. Pominu-li u novější verze TKSL absenci GUI a nutnost vizualizace získaných výsledků externím programem, z hlediska jednoduchosti použití a přesnosti jasně vítězí systém TKSL. V případě, že by se povedlo výpočty v TKSL paralelizovat, jistě by se z něj stal velmi výkonný, přesný a jednoduchý nástroj pro řešení nejen diferenciálních rovnic.

Práce staví na řadě podkladů, které jsou uvedeny v přehledu literatury, přílohy pak tvoří ukázky kódu pro modely vedení a ukázka výstupu z prostředí TKSL/C. S ohledem na diskutovanou problematiku jsem zadání práce splnil.

[1] double, ale tato bude muset být podporována i v samotných GPU

Literatura

[1] AnandTech: NVIDIA's 1.4 Billion Transistor GPU: GT200 Arrives as the GeForce GTX 280 and 260. [Online; navštíveno 20. 04. 2009].
URL http://www.anandtech.com/video/showdoc.aspx?i=3334

[2] ATI Radeon(tm) HD 3800 Series - GPU Specifications. [Online; navštíveno 15.05. 2009].
URL http://ati.amd.com/products/radeonhd3800/specs.html

[3] ATI Radeon(tm) HD 4850 - GPU Specifications. [Online; navštíveno 15. 05. 2009].
URL http://ati.amd.com/products/Radeonhd4800/specs.html

[4] nVidia Corporation: NVIDIA CUDA Programming Guide 2.2. [Online; accessed 2-May-2009].

[5] Farber, R.: Dr. Dobbs's: CUDA, Supercomputing for the Masses: Part 4. 2009, [Online; accessed 15-May-2009].
URL http://www.ddj.com/architect/208401741;jsessionid=FE2KHIIJ15LI0QSNDLPSKHSCJUNN2JVN?pgno=1

[6] Farber, R.: Dr. Dobbs's: CUDA, Supercomputing for the Masses: Part 5. 2009, [Online; accessed 15-May-2009].
URL http://www.ddj.com/hpc-high-performance-computing/208801731;jsessionid=MZMDOS5HYCVE4QSNDLPSKHSCJUNN2JVN?pgno=1

[7] Kraus, M.; Kunovský, J.; Petřek, J.; aj.: GPU Based Accelleration of Telegraph equation. 2008.

[8] Owens, J.: GPU Architecture Overview. [Online; navštíveno 20. 03. 2009].
URL
http://www.gpgpu.org/s2007/slides/02-gpu-architecture-overview-s07.pdf

[9] Wikipedia: Partial differential equation — Wikipedia, The Free Encyclopedia. 2008, [Online; accessed 2-January-2009].
URL
http://en.wikipedia.org/w/index.php?title=Partial_differential_equation

[10] Wikipedia: CUDA — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 5-January-2009].
URL http://en.wikipedia.org/w/index.php?title=CUDA

[11] Wikipedia: Differential equation — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 15-May-2009].

URL http://en.wikipedia.org/w/index.php?title=Differential_equation&oldid=288542720

[12] Wikipedia: DirectX — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 15-May-2009].
URL http://en.wikipedia.org/w/index.php?title=DirectX&oldid=288626877

[13] Wikipedia: Euler method — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 15-May-2009].
URL
http://en.wikipedia.org/w/index.php?title=Euler_method&oldid=291013770

[14] Wikipedia: OpenGL — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 15-May-2009].
URL http://en.wikipedia.org/w/index.php?title=OpenGL&oldid=289039478

[15] Wikipedia: PCI Express — Wikipedia, The Free Encyclopedia. 2009, [Online; accessed 14-May-2009].
URL
http://en.wikipedia.org/w/index.php?title=PCI_Express&oldid=289825449

[16] Wikipedie: Obyčejné diferenciální rovnice — Wikipedie: Otevřená encyklopedie. 2008, [Online; navštíveno 29. 12. 2008].
URL http://cs.wikipedia.org/w/index.php?title=Oby%C4%8Dejn%C3%A9_diferenci%C3%A1ln%C3%AD_rovnice

[17] Wikipedie: Parciální diferenciální rovnice — Wikipedie: Otevřená encyklopedie. 2008, [Online; navštíveno 29. 12. 2008].
URL http://cs.wikipedia.org/w/index.php?title=Parci%C3%A1ln%C3%AD_diferenci%C3%A1ln%C3%AD_rovnice

[18] Wikipedie: Diferenciální rovnice — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 16. 05. 2009].
URL http://cs.wikipedia.org/w/index.php?title=Diferenci%C3%A1ln%C3%AD_rovnice&oldid=3499769

[19] Wikipedie: DirectX — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 15. 05. 2009].
URL http://cs.wikipedia.org/w/index.php?title=DirectX&oldid=3947995

[20] Wikipedie: Eulerova metoda — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 15. 05. 2009].
URL
http://cs.wikipedia.org/w/index.php?title=Eulerova_metoda&oldid=3890796

[21] Wikipedie: Hyperbolická diferenciální rovnice — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 2. 01. 2009].
URL http://cs.wikipedia.org/w/index.php?title=Hyperbolick%C3%A1_diferenci%C3%A1ln%C3%AD_rovnice

[22] Wikipedie: PCI-Express — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 14. 05. 2009].

URL
http://cs.wikipedia.org/w/index.php?title=PCI-Express&oldid=3881124

[23] Wikipedie: Taylorova řada — Wikipedie: Otevřená encyklopedie. 2009, [Online; navštíveno 25. 05. 2009].
URL http:
//cs.wikipedia.org/w/index.php?title=Taylorova_%C5%99ada&oldid=3820134

Seznam použitých zkratek a symbolů

Cg – C for graphics

CPU – Central Processing Unit

CUDA – Compute Unified Device Architecture

GFLOP – Giga Floating Point Operations per second

GLSL – OpenGL Shading Language

GPU – Graphics Processing Unit

HLSL – High Level Shading Language

OpenGL – Open Graphics Library

RK4 – Numerická intergrační metoda Runge-Kutta 4. řádu

SIMD – Single Instruction, Multiple Data

Příloha A

Program pro TKSL/386 popisující element vedení

```
var u1,
    uc1,
    i2,i1,
    ic1,
    u2;

const R1=100,    R2=100,
      Rs=1e-3,   Rp=1e+10,
      C =1e-12,  L =1e-8,
      dt=1e-5,   tmax=1e-6,
      eps=1e-10, omega=1e7;

system

i1=1/R1*(u1-uC1);

uc1'= 1/C*ic1             &0;
iC1 = i1 -1/Rp*uC1 - i2 ;
i2'= 1/L*(uc1-R2*i2-Rs*i2) &0;

u1=sin(omega*t);

sysend.
```

Příloha B

Program pro TKSL/386 popisující kaskádu 5ti dvojbranů

```
var     u1, uc1, uc2, uc3,uc4,uc5,uX,
        i0, i1, i2, i3,i4,i5,
        ic1, ic2, ic3, ic4,ic5;
const   R1=100,R2=100,
        Rs=1e-2,Rp=1e+10,{3}
        C=1e-12, L=1e-8,
        dt=1e-6, tmax=5e-7,
        eps=1e-10, omega=1e7;
system
i0=(1/R1)*(u1-uc1); uX  = R2*i5; u1=10*sin(omega*t);

   uc1' = (1/C)*ic1  &0;
   ic1  = i0-(1/Rp)*uc1 -i1;
   i1'  = (1/L)*(uc1- uc2 - Rs*i1) &0;

   uc2' = 1/C * ic2 &0;
   ic2  = i1 -(1/Rp)*uc2 - i2;
   i2'  = (1/L) * (uc2 - uc3- Rs*i2) &0;

   uc3' = 1/C * ic3 &0;
   ic3  = i2 -(1/Rp)*uc3 - i3;
   i3' = (1/L)*(uc3 - uc4 - Rs*i3) &0;

   uc4' = (1/C)*ic4 &0;
   ic4  = i3 -(1/Rp)*uc4 - i4;
   i4'  = (1/L)*(uc4 - uc5 -Rs*i4) &0;

   uc5' = (1/C)*ic5 &0;
   ic5  = i4-(1/Rp)*uc5 - i5;
   i5'  = (1/L)*(uc5-uX -Rs*i5)  &0;

sysend.
```

Příloha C

Ukázka výstupu TKSL/C - Program pro analýzu vedení o 500 dvojbranech

```
t  uc1  uc300  uc500
0.0000000000e+00  0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
1.0000000000e-11  4.8334166214e-06  7.9001177735e-2012  2.4848950244e-3571
2.0000000000e-11  1.8669328187e-05  3.2774579449e-1831  2.6622407832e-3270
3.0000000000e-11  4.0520163147e-05  1.4797363970e-1725  3.2842889766e-3094
4.0000000000e-11  6.9418019111e-05  1.3592637908e-1650  2.8517030416e-2969
5.0000000000e-11  1.0442400758e-04  1.9018427751e-1592  2.3175813527e-2872
6.0000000000e-11  1.4463702452e-04  6.1343835895e-1545  3.5171433509e-2793
7.0000000000e-11  1.8920177759e-04  9.0298186649e-1505  3.1108572822e-2726
8.0000000000e-11  2.3731591965e-04  5.6320586439e-1470  3.0529437684e-2668
9.0000000000e-11  2.8823615958e-04  2.7668538576e-1439  4.3363501765e-2617
1.0000000000e-10  3.4128324688e-04  7.8753846101e-1412  2.4802151321e-2571
1.1000000000e-10  3.9584575375e-04  5.3908806170e-1387  6.1240723170e-2530
1.2000000000e-10  4.5138260698e-04  2.5383922085e-1364  3.7623427069e-2492
1.3000000000e-10  5.0742435205e-04  1.8262935735e-1343  2.1747689162e-2457
1.4000000000e-10  5.6357316103e-04  3.7334845140e-1324  3.3261080013e-2425
1.5000000000e-10  6.1950162525e-04  3.5462369648e-1306  3.0548573557e-2395
1.6000000000e-10  6.7495040038e-04  2.3265206700e-1289  3.2623991917e-2367
1.7000000000e-10  7.2972479715e-04  1.4580109399e-1273  6.9548815973e-2341
1.8000000000e-10  7.8369043248e-04  1.1417927539e-1258  4.6310421780e-2316
1.9000000000e-10  8.3676807489e-04  1.3992866414e-1244  1.4014293529e-2292
2.0000000000e-10  8.8892783241e-04  3.2463178628e-1231  2.6469997317e-2270
2.1000000000e-10  9.4018284212e-04  1.6772725558e-1218  4.0910265386e-2249
2.2000000000e-10  9.9058262618e-04  2.2194945462e-1206  6.5311411173e-2229
2.3000000000e-10  1.0402062812e-03  8.4908587932e-1195  1.3179723477e-2209
2.4000000000e-10  1.0891556650e-03  1.0437225927e-1183  4.0092707066e-2191
2.5000000000e-10  1.1375487372e-03  4.5229411905e-1173  2.1457399279e-2173
2.6000000000e-10  1.1855132012e-03  7.4986974384e-1163  2.3155410024e-2156
2.7000000000e-10  1.2331805769e-03  5.1142562522e-1153  5.6858593309e-2140
2.8000000000e-10  1.2806808212e-03  1.5306481330e-1143  3.5381976868e-2124
```