evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Modelování

Model je umělý objekt, který reflektuje a reprodukuje základní vlastnosti, vztahy (strukturu) a funkce konkrétního objektu nebo jevu (skutečnosti) jednodušším způsobem a může tedy být využit pro vyšetřování a analýzu skutečnosti.

- redukuje realitu,
- zjednodušuje práci s objektem nebo jevem,
- účel modelu je třeba vzít v úvahu při výběru vlastností modelovaného objektu nebo jevu,
- často má vlastnosti, které neodpovídají ničemu v původním objektu nebo jevu,
- často máme velké množství modelů, které jsou více či méně vhodné k analýze a popisu konkrétních vlastností.

Kyvadlo

- díky tření v závěsu a odporu vzduchu jsou oscilace tlumené,
- budeme vyšetřovat dobu prvního kompletního kmitu kyvadla,
- chceme, aby absolutní hodnota rozdílu vypočtené a skutečné doby byla menší než požadované ϵ : $|T_V - T_S| < \epsilon$.

Matematické kyvadlo

- oscilace jsou netlumené,
- hmota kyvadla je koncentrována do jednoho bodu zavěšeného na tenkém vlákně zanedbatelné hmotnosti,
- z druhého Newtonova pohybového zákona plyne:

$$\varphi''(t) = -\frac{g}{l}\sin(\varphi(t)), \qquad \varphi(0) = \varphi_0, \quad \varphi'(0) = 0,$$

l je délka vlákna, g je tíhové zrychlení, φ je úhel vychýlení vlákna z rovnovážné polohy, φ_0 je úhel počátečního vychýlení.

Linearizace problému

- pro malé hodnoty φ_0 („malé“ závisí na požadované přesnosti řešení) můžeme nahradit funkci sinus lineární funkcí,
- místo nelineární diferenciální rovnice pak dostaneme lineární

$$\varphi''(t) = -\frac{g}{l}\varphi(t), \qquad \varphi(0) = \varphi_0, \quad \varphi'(0) = 0,$$

- tu lze snadno vyřešit: $\varphi(t) = \varphi_0 \cos\left(\sqrt{\frac{g}{l}}t\right)$
- a perioda oscilace je: $T_V = 2\pi\sqrt{\frac{l}{g}}.$

Pro libovolné hodnoty počátečního vychýlení

- je třeba řešit nelineární diferenciální rovnici,
- periodu oscilace lze vyjádřit pomocí eliptického integrálu a následně pomocí nekonečné řady

$$T_V = 2\pi\sqrt{\frac{l}{g}}\left(1 + \left(\frac{1}{2}\right)^2 \sin^2\left(\frac{\varphi_0}{2}\right) + \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 \sin^4\left(\frac{\varphi_0}{2}\right) + \dots\right).$$

Složitější modely

- ztráty třením v závěsu a odporem vzduchu mohou být zahrnuty přidáním dalšího členu

$$\varphi''(t) = -\frac{g}{l}\varphi(t) - T\left(\varphi'(t)\right), \qquad \varphi(0) = \varphi_0, \quad \varphi'(0) = 0,$$

- funkce T může být nelineární,
- výše uvedená rovnice může být již řešena pouze numericky.

Existují i ještě složitější modely zahrnující elastické deformace nebo aerodynamické síly.

Tvorba modelu:

- specifikace problému - problém by měl být formulován co nejpřesněji (zahrnuje třeba i požadovanou přesnost výpočtu),
- návrh modelu
  - model musí umožnit popis problému odpovídající specifikaci,
  - ze dvou stejně vhodných modelů je lepší ten, který vyžaduje méně předpokladů a méně zdrojů (příliš mnoho stupňů volnosti může vést k zavádějící shodě mezi výsledky modelu a experimentu),
  - shoda není důkazem vhodnosti modelu,
- vývoj algoritmů a softwaru,
- testování a ověřování zda-li chování modelu dostatečně odpovídá chování modelovaného systému a také chování modelu blízko limitů jeho použitelnosti.

Numerická (výpočtová) matematika se zabývá vývojem a analýzou metod pro přibližné řešení matematických modelů s pomocí počítače.

Diskretizace je proces při němž spojitý (matematický) problém nahradíme vhodným diskrétním (numerickým) problémem, který pracuje s konečným počtem vyhodnocení aritmetických operací. Například integrál nahradíme konečným součtem, limitu nahradíme diferenčním podílem, ...

Numerický problém = matematický problém + požadovaná přesnost výsledků.

Numerické metody musí obsahovat prostředky k odhadům přesnosti přibližných řešení.

Například Gaussova eliminace neobsahuje prostředky k odhadům přesnosti. V přesné aritmetice obdržíme řešení po konečném počtu kroků, zatímco v počítačové aritmetice se obdržený výsledek může, v závislosti na konkrétní úloze a implementaci algoritmu, významně lišit od přesného řešení. Zlepšení by muselo spočívat v implementaci přesnější počítačové aritmetiky, ale to je většinou velice neefektivní.

Druhy chyb

- chyba modelu - je odchylkou modelu od originálu a někdy ji nelze ani odhadnout (relativistické efekty, kyvadlo),
- chyba dat - je odchylka naměřených dat od skutečných hodnot (kyvadlo - l, g, φ_0, ale třeba i rozdíly v pravých stranách) a její šíření lze odhadnout pomocí analýzy čísla podmíněnosti dané úlohy,
- chyba algoritmu - je odchylka přibližného řešení od přesného řešení matematického problému, rozlišujeme
  - chybu zanedbáním - exponenciální funkce, sinus a cosinus jsou definovány pomocí nekonečné řady, ale v počítači lze implementovat jen částečný součet,
  - diskretizační chybu - je důsledkem nahrazení spojitého problému diskrétním,
- zaokrouhlovací chyba - singulární matice se díky tomu může stát při GE regulární, zaokrouhlování může vést ke ztrátě platných číslic (viz výpočet funkčních hodnot funkce cosinus pomocí částečného součtu Taylorovy řady pro velké hodnoty argumentu), jsou obvykle velice obtížně kvantifikovatelné.

- velikosti jednotlivých druhů chyb by měly být vyvážené,
- chybu modelu je možné ovlivnit při jeho tvorbě, nicméně nejistota ohledně zahrnutí všech vlivů zůstává,
- chybu dat lze někdy ovlivnit přesností měření,
- u špatně podmíněných úloh ani významné zvýšení přesnosti dat a algoritmu nemusí vést k přesnějšímu výsledku,
- je-li možné chybu algoritmu libovolně zmenšit, potom mluvíme o **konvergentním algoritmu**,
- pro mnoho numerických metod je možné odvodit odhad chyby
  - pro $x \in (-\sqrt{2}, \sqrt{2})$ je $\left| \sum_{i=0}^{k} (-1)^k \frac{x^{2k}}{(2k)!} - \cos x \right| < \frac{x^{2k+2}}{(2k+2)!}$,
  - obdélníkové pravidlo: $|I(f) - R_n(f)| < \frac{b-a}{2} ||f'(x)||_{\max} h$,
  - lichoběžníkové pravidlo: $|I(f) - L_n(f)| < \frac{b-a}{12} ||f''(x)||_{\max} h^2$,
  - metoda poloviční kroku,
- zaokrouhlovací chyba může být do jisté míry řízena volbou přesnosti (single, double, ...),
- získat spolehlivé odhady chyby je obtížné a někdy i nemožné.

Měření chyby

- absolutní chyba $= ||\text{přesná hodnota} - \text{aproximace}||$,
- relativní chyba $= \frac{||\text{přesná hodnota} - \text{aproximace}||}{||\text{přesná hodnota}||}$,
- $||.||$ je vhodná norma (absolutní hodnota, norma vektoru),
- například při grafické reprezentaci funkce funkcí po částech lineární - zde je třeba chybu volit tak, aby byla menší než rozlišení zobrazovacího zařízení tedy v normě $||.||_{max}$,

Řád metody

- h - parametr charakterizující algoritmus (např. krok metody)
- chyba metody $e(h)$ je řádu $f(h)$, jestliže existují konstanty a, b tak, že $e(h) \leq b\, f(h) \qquad \forall h \leq a$.

konvergence	řád chyby
lineární	h
kvadratická	h^2
kubická	h^3

Omezení numerických výpočtů

- počítače používají konečnou množinu racionálních čísel (floating point numbers),
- ostatní čísla musí být aproximována těmito čísly,
- výsledky aritmetických operací musí být aproximovány, protože aritmetické operace nejsou uzavřené na této množině,
- funkční hodnoty elementárních funkcí musí být aproximovány,
- neexistují libovolně velká ani libovolně malá čísla,
- výpočty mohou obsahovat jen konečný počet kroků,
- obvykle nelze spočítat přesný výsledek,
- výpočetní náročnost všech výpočtů s maximální přesností nelze zanedbat,
- cílem je nalézt přibližné řešení, které splní požadavky uživatele na přesnost a je co nejméně výpočetně náročné.

Podmíněnost matematických úloh

Matematickou úlohu lze chápat jako zobrazení $y = f(x)$, které vstupním datům $x \in D$ přiřazuje výstupní data $y \in R$. Řekneme, že matematická úloha

$$y = f(x), \quad x \in D, \quad y \in R$$

je **korektní (well-posed)**, když

- $\forall x \in D \; \exists! v \in R$,
- řešení závisí spojitě na vstupních datech, tj. když $x \to a$, potom $f(x) \to f(a)$.

Nekorektní úlohy (ill-posed) jsou tedy například úlohy s nejednoznačným řešením a nelze je většinou numericky řešit. Nejprve je třeba použít nějakou regularizační techniku.

Řekneme, že korektní úloha je dobře podmíněná, jestliže malá změna ve vstupních datech způsobí malou změnu ve výstupních datech. Je-li $y + \Delta y$ respektive y řešením úlohy $y = f(x)$ odpovídající vstupním datům $x + \Delta x$ respektive x, potom číslo

$$C_r = \frac{\frac{||\Delta y||}{||y||}}{\frac{||\Delta x||}{||x||}} \qquad (||y||, ||x|| \neq 0)$$

je relativní číslo podmíněnosti této úlohy. Je-li $C \approx 1$, řekneme, že je úloha dobře podmíněná. Pro velké hodnoty C, řekneme, že je úloha špatně podmíněná. Někdy může být vhodnější použít absolutní číslo podmíněnosti definované předpisem:

$$C_a = \frac{||\Delta y||}{||\Delta x||}.$$

Obvykle obě čísla podmíněnosti definujeme jako infimum vzhledem ke konkrétní množině vstupních dat.

Předpokládejme, že funkce f má spojitou první derivaci a nyní budeme chtít odhadnout číslo podmíněnosti pro výpočet funkční hodnoty $y = f(x)$. Podle věty o střední hodnotě dostaneme

$$\Delta y = f(x + \Delta x) - f(x) = \Delta x\, f'(\xi),$$

$$\text{kde} \qquad \xi \in (x, x + \Delta x) \quad \text{nebo} \quad \xi \in (x + \Delta x, x).$$

Potom

$$\frac{\Delta y}{y} = \frac{\Delta x\ f'(\xi)}{y} = \frac{\Delta x}{x}\frac{x\ f'(\xi)}{f(x)} \qquad \text{a} \qquad C_r = \left|\frac{x\, f'(\xi)}{f(x)}\right|.$$

Například pro funkci $\operatorname{tg} x$ (a malé hodnoty Δx) dostaneme

$$C_r = \left|\frac{x\, f'(\xi)}{f(x)}\right| \approx \left|\frac{x\, f'(x)}{f(x)}\right| = \left|\frac{x\ \cos x}{\cos^2 x\ \sin x}\right| = \left|\frac{2x}{\sin 2x}\right|.$$

Výpočet funkce $\operatorname{tg} x$ je tedy špatně podmíněná úloha pro x blížící se celočíselnému násobku $\pi/2$.

Čísla podmíněnosti základních operací:

- sčítání: $C_r = \left|\frac{a}{a+b}\right|$ a $C_r = \left|\frac{b}{a+b}\right|$,
- odčítání: $C_r = \left|\frac{a}{a-b}\right|$ a $C_r = \left|\frac{b}{a-b}\right|$,
- násobení: $C_r = \left|\frac{ab}{ab}\right| = 1$,
- dělení: $C_r = \left|\frac{a/b}{a/b}\right| = 1 = \left|\frac{-ab/b^2}{a/b}\right|$.

Čísla podmíněnosti vybraných funkcí:

- $\ln a$: $C_r = \left|\frac{a/a}{\ln a}\right| = \frac{1}{|\ln a|}$,
- $\sin a$: $C_r = \left|\frac{a \cos a}{\sin a}\right|$.

Při numerických výpočtech vznikají zaokrouhlovací chyby jednak ve vstupních datech a jednak v průběhu výpočtu. **Stabilním algoritmem** nazveme takový algoritmus, který je

- dobře podmíněný - tedy málo citlivý na změny ve vstupních datech,
- numericky stabilní - tedy málo citlivý na vliv zaokrouhlovacích chyb vznikajících během výpočtu.

Kvadratická rovnice $x^2 - 2bx + c = 0$ má řešení

$$x_{1,2} = b \pm D, \qquad D = \sqrt{b^2 - c}.$$

Nicméně tento výpočet je nestabilní v případě, že $|b| \approx |D|$ protože pak se budou odčítat dvě přibližně stejně velká čísla, což vede k velké relativní chybě. Náprava spočívá v jednoduché modifikaci

$$x_1 = \begin{cases} b + D & \text{pro } b \geq 0 \\ b - D & \text{pro } b < 0 \end{cases} \qquad x_2 = \frac{c}{x_1}.$$

Číselné systémy na počítačích

- čísla mají konečnou délku a lze tedy zakódovat jen 2^N různých čísel (například $2^{32} = 4294967296$)

Celá čísla můžeme reprezentovat například takto:

- posloupností bitů $d_{N-1}d_{N-1}\ldots d_2d_1d_0$, kde $d_j \in \{0, 1\}$,
- první bit představuje znaménko,
- $x = \sum_{j=0}^{N-2} d_j 2^j$ pro $x \geq 0$ a
- $x = -1 - \sum_{j=0}^{N-2}(1 - d_j)2^j$ pro $x < 0$,
- - reprezentuje čísla $x \in [-2^{N-1}, (2^{N-1} - 1)]$, nula je určena jednoznačně, nesymetrické kódování:

$$000\ldots000 \models 0, \qquad 111\ldots111 \models -1,$$

Vede-li výsledek aritmetické operace k přetečení, je buď hlášena chyba nebo je vrácena modulo hodnota (jazyk C).

Číselné systémy s pohyblivou desetinou čárkou - posloupnost bitů je rozdělena do tří částí - znaménka, exponentu (celé číslo) a mantisy (nezáporné reálné číslo). Například $N = 32 = 1 + 8 + 23$ nebo $N = 64 = 1 + 11 + 52$. Systém čísel o základu a se tedy skládá z následujících čísel

$$x = (-1)^z a^b \sum_{j=1}^{m} d_j a^{-j}$$

kde

$$z \in \{0, 1\}, \quad b \in \{e_{\min}, e_{\max}\}, \quad d_j \in \{0, 1, \ldots, a-1\}.$$

Například pro $N = 32$, $e = 8$, $m = 23$, $a = 2$ máme

$$x = (-1)^z 2^b \sum_{j=1}^{23} d_j 2^{-j}$$

kde

$$z \in \{0, 1\}, \quad b \in \{-127, 128\}, \quad d_j \in \{0, 1\}.$$

- normalizovaná čísla jsou taková, že $d_1 \neq 0$,
- normalizací dojde ke ztrátě v absolutní hodnotě malých čísel tzv. denormalizovaných čísel,
- každý systém lze charakterizovat pěti parametry: základem a, velikostí mantisy m, nejmenším exponentem $e_{\min}$, největším exponentem $e_{\max}$ a informací, zda-li obsahuje denormalizovaná čísla,
- například Intel používá (single) $a = 2$, $m = 24$, $e_{\min} = -125$, $e_{\max} = 128$ a denormalizovaná čísla nebo (double) $a = 2$, $m = 53$, $e_{\min} = -1021$, $e_{\max} = 1024$ a denormalizovaná čísla,
- v praxi se exponent obvykle nekóduje symetricky a ukládá se jako kladné číslo,
- díky normalizaci je při základu dva první bit vždy jedna a není ho tedy třeba ukládat,
- pokud jsou povolena i denormalizovaná čísla, je třeba je odlišit pomocí exponentu (obvykle $e_{\min} - 1$),

- IEEE Standard for Floating-Point Arithmetic - IEEE 754 2008, IEEE = Institute of Electrical and Electronics Engineers,
- nula je reprezentována nulou v mantise a hodnotou $e_{\min} - 1$ v exponentu,
- $\pm\infty$ je reprezentováno nulou v mantise a hodnotou $e_{\max} + 1$ v exponentu,
- NaN je reprezentováno nenulovou hodnotou v mantise a hodnotou $e_{\max} + 1$ v exponentu,
- zaokrouhlování k nejbližšímu číslu a dále buď zaokrouhlování typu „pryč od nuly“ nebo typu „na sudé“,
- přetečení, podtečení, sčítání většího počtu čísel
- pro $\forall e \in [e_{\max}, e_{\min}]$ je absolutní chyba zaokrouhlení rovna maximálně $a^{e-m}/2$,
- relativní zaokrouhlovací chyba pro normalizovaná čísla je $a^{1-m}/2$ (strojové epsilon eps),
- pro Intel - single $2^{-24} \approx 5,96 \cdot 10^{-8}$ a pro double $2^{-53} \approx 1,11 \cdot 10^{-16}$.

Složitost algoritmů

Řád složitosti algoritmu je $f(p)$, jestliže $\exists\, a,\ b$ tak, že

$$C(p) \leq b\, f(p) \qquad \forall p \geq a,$$

kde $C(p)$ je složitost algoritmu (v závislosti na parametru p).

Příklady tříd složitosti: konstantní $f(p) = 1$, logaritmická $f(p) = \log\, p$, lineární $f(p) = p$, kvadratická $f(p) = p^2$, kubická $f(p) = p^3$, polynomiální $f(p) = p^m$, $m \in \mathbb{N}$ a exponenciální $f(p) = m^p$, $m > 1$.

	10	20	30	40	50
p	0.00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s
p^2	0.0001 s	0.0004 s	0.0009 s	0.0016 s	0.0036 s
p^3	0.001 s	0.008 s	0.027 s	0.064 s	0.125 s
2^p	0.001 s	1.0 s	17.19 min	12.7 dne	35.7 let
3^p	0.059 s	58 min	6.5 roku	3855 stol.	200 mil. stol.

Tabulka: Srovnání časů pro různé velikosti dat a časové složitosti.

Například:

- quicksort má řád složitosti $p \log p$,
- Gaussova eliminace má řád složitosti p^3,
- násobení dvou matic má řád složitosti p^3,
- násobení dvou matic (Strassen) má řád složitosti $p^{\log_2 7}$.

Floating-point operace (flop)

- dělení a odmocňování je desetkrát až třicetkrát náročnější než sčítání, odčítání nebo násobení,
- výpočet funkčních hodnot exponenciální funkce nebo trigonometrických funkcí je náročnější asi padesátkrát,
- některé procesory poskytují složené instrukce, které zvládají vynásobení dvou čísel a přičtení výsledku ke třetímu ve stejném čase jako samotné násobení (další výhodou je, že mezivýsledky nejsou zaokrouhlovány).

Kvalita numerických programů

- spolehlivost
  - správnost - pro přípustnou množinu vstupních dat vrátí správný výsledek,
  - robustnost - rozpozná nekorektní data a upozorní na ně uživatele,
  - přesnost - v ideálním případě by se výsledek měl rovnat přesnému řešení převedenému do počítačové aritmetiky,
- přenositelnost je charakterizována námahou nutnou k přenosu softwaru z jednoho prostředí do druhého
  - software - užívat standardizované programovací jazyky,
  - hardware - vyhnout se užívání specifických vlastností hardwaru,
- efektivnost je charakterizována vztahem mezi výkonem a množstvím využitých zdrojů
  - potřeba času pro výpočet,
  - potřeba paměti a místa na disku,
- trade-off mezi efektivností a spolehlivostí (přenositelností),
- adaptivní programy (získají a využijí informace o hardwaru a softwaru - např. o počítačové aritmetice).

Důvody neefektivnosti

- nedostatek paralelismu - schopnost kompilátoru využít paralelní zdroje může být limitována tím, že instrukce musí být prováděny v určeném pořadí z důvodu závislosti dat nebo závislosti v řízení (podmínka if),
- nedostatečná lokalita odkazů - jestliže data nejsou nalezena v cache paměti, musí být hledána v hlavní paměti a případně dokonce na disku,
- neefektivní využití paměti - může vést k tomu, že některé části paměti jsou využívány častěji než jiné, zatímco některé části jsou prakticky nevyužity,
- režie (overhead) - části programu, které přímo nepřispívají k dosažení výstupu
  - volání funkcí, výpočet indexů, apod.,
  - skutečně nepotřebné operace.

Měření výkonnosti

- počet aritmetických operací,
- požadavky na paměť,
- vstupní a výstupní operace (obzvlášť v situaci, kdy dochází k vyčerpání paměti),
- doba běhu programu (značně problematické, protože pro větší programy se doba běhu programu i pro stejná data může značně lišit),
- profiling - hledání nejčastěji využívaných úseků programu.

Optimalizace výkonu

- je nutná důkladná znalost systému (obzvláště překladače),
- minimalizovat počet vyhodnocování identických výrazů,
- vyhnout se typovým konverzím,
- inlining (předávání parametrů a volání podprogramů je pro krátké procedury neúměrně náročné),
- snížení složitosti bez vlivu na výsledek ($e^x e^y$ versus e^{x+y}).

Optimalizace cyklů

- většina instrukcí numerických programů je prováděna v cyklu,
- to skýtá velký prostor pro optimalizaci,
- loop unrolling spočívá v několikanásobném kopírování těla cyklu (a případné modifikaci cyklu)
  - tím se sníží frekvence testování ukončovací podmínky,
  - rozšíří se možnosti paralelizace,
  - může také zredukovat množství zbytečných přesunů dat mezi registry,
  - efektivita závisí na velikosti těla cyklu, počtu průchodů cyklem, zda-li obsahuje volání procedur a také na závislostech mezi jednotlivými průchody,
  - nevýhody spočívají ve zvýšení velikosti, složitosti a rizika chyb,
- loop unrolling vnitřního cyklu u vnořených cyklů může zlepšit lokalitu odkazů,
- spojení cyklů má podobný efekt jako loop unrolling,
- větvení programu uvnitř cyklu je většinou neefektivní - je-li to možné, je lepší se mu vyhnout (například udělat jeden cyklus pro liché indexy a druhý pro sudé indexy),

- využít asociativitu (například výpočet skalárního součinu),
- přehození pořadí cyklů,
  - nejvnitřnější cyklus má největší vliv na počet operací a využití paměti,
  - cílem je optimalizovat lokalitu odkazů a (nebo) umožnit další optimalizaci,
  - například pro trojrozměrná pole může být optimální uložení a optimální pořadí až čtyřikrát rychlejší oproti neoptimálním variantám.

Blokové algoritmy

- umožňuji optimalizovat lokalitu odkazů,
- velikost bloků musí být volena tak, aby se všechny bloky potřebné pro výpočet vešly do paměti a zároveň aby počet operací s jejich prvky byl maximální,
- bloky lze ještě dále rozdělit do bloků z důvodu optimálního využití všech úrovní paměti,
- dodatečná pomocná pole obsahující kopie bloků zajistí, že se do paměti uloží jen příslušný blok a navíc umožní toto pole optimálně transponovat.

Paralelní implementace

- v podstatě všechny nové počítače mají paralelní architekturu (procesor s několika jádry), takže využití jejich potenciálu vyžaduje paralelní aplikace,
- vždy se najdou náročné úlohy, pro které je výkon jednoho počítače nedostatečný, takže se na jejím řešení musí podílet více počítačů,
- paralelní program je představován dvěma a více souběžně prováděnými činnostmi, které si spolu vyměňují informace,
- výměna informací se děje předáváním zpráv případně pomocí sdílených proměnných,
- paralelní zpracování probíhá rovněž na úrovni instrukcí, protože moderní procesory umí provádět více instrukcí souběžně,
- optimalizovaný paralelní program závisí na platformě,
- klasifikace víceprocesorových systémů podle Flynna - SISD, SIMD, MISD, MIMD (Single/Multiple Instruction - Single/Multiple Data),

- klasifikace podle uspořádání paměti - společná (sdílená) paměť (multiprocesor) nebo má každý procesor svou (distribuovanou) paměť (multipočítač).
- nejobecnější model pro paralelní zpracování představuje model posílání zpráv, který lze využít na všechny systémy s architekturou MIMD,
- počítačovou realizací tohoto modelu popisuje standard Message Passing Interface (MPI),
- asi nejznámější volně dostupné implementace MPI jsou Open MPI a MPICh.

Paralelní implementace Simpsonova pravidla

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv) {
    int size, rank, i;
    int n = 500000;
    MPI_Status s;
```

```
MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if(rank == 0) {
    double GI = 0.0;
    double PI;

    for(i = 1; i < size; i++) {
        MPI_Recv(&PI, 1, MPI_DOUBLE, i, 1, MPI_COMM_WORLD, &s);
        GI += PI; }

    printf("Integrál = %f\n", GI); }

else {
    double LI;
    double x0 = (rank-1)/(size-1);
    double xn = (rank)/(size-1);

    LI = Simpson(n, x0, xn);
    MPI_Send(&LI, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD); }

MPI_Finalize();

return 0; }
```

Základy MPI

- int MPI_Init(int *argc, char ***argv) - inicializuje prostředí MPI a vytváří komunikátor MPI_COMM_WORLD, v jehož rámci je každému procesu přiřazeno identifikační číslo (rank),
- int MPI_Finalize(void) - ukončuje prostředí MPI,
- int MPI_Comm_rank(MPI_Comm comm, int *rank) - v parametru rank vrací identifikační číslo volajícího procesu v rámci komunikátoru comm,
- int MPI_Comm_size(MPI_Comm comm, int *size) - v parametru size vrací počet procesů v rámci komunikátoru comm,
- int MPI_Send(void *buffer, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm) - standardní odeslání zajistí odeslání count položek v rámci komunikátoru comm typu datatype umístěných v bufferu procesu dest ve zprávě se značkou tag (slouží případně k identifikaci zprávy),

- int MPI_Recv(void *buffer, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status) - standardní blokovací příjem zajistí, že data odeslaná procesem source v rámci komunikátoru comm ve zprávě se značkou tag budou umístěna do bufferu; zpráva může obsahovat maximálně count položek typu datatype; status obsahuje identifikační číslo odesílatele a značku přijímané zprávy,
- pokud nezáleží příjemci na značce přijímané zprávy, použije se MPI_ANY_TAG,
- pokud je třeba přijmout zprávu od libovolného příjemce, použije se MPI_ANY_SOURCE,
- vzhledem k tomu, že odeslání a příjem nejsou obvykle synchronizované, zajišťuje MPI vyrovnávací paměť, v níž se nedoručené zprávy ukládají,
- synchronní (ukončeno příjmem) a asynchronní (ukončeno odesláním) odeslání,
- blokovací (proces čeká na ukončení) a neblokovací (proces může pracovat na něčem jiném) operace.

Numerický software

- kvalitní software je volně dostupný pro většinu základních úloh,
- většinou ve Fortranu nebo v jazyce C,
- BLAS - základní operace lineární algebry (PBLAS),
- LAPACK - přímé řešení soustav lineárních rovnic, metoda nejmenších čtverců a výpočet vlastních čísel a vlastních vektorů (SCALAPACK),
- ATLAS - automaticky vyladěné operace lineární algebry (obsahuje BLAS a malou část LAPACKu) pro konkrétní počítač,
- QUADPACK - numerická integrace,
- PETSc - řešení parciálních diferenciálních rovnic,
- náhrada MALABu - FREEMAT nebo OCTAVE,
- Netlib.