T V O R B A , L A D Ě N Í A T E S T O V Á N Í N U M E R I C K É H O S O F T W A R E

RNDr. Jiří Hřebíček, CSc., RNDr. Ivan Kopeček, CSc.

1. Úvod

Následující pojednání nemá za účel postihnout všechny aspekty týkající se vývoje numerických knihoven, protože to ve vymezeném rozsahu ani není možné. Ostatně mnoho principů je zde společných a všeobecně známých. Nebudeme například zdůrazňovat princip modularity a strukturovaného programování při tvorbě numerických podprogramů, protože tento princip je v podstatě obecný a na toto téma již byla napsána řada jiných učených pojednání. Pokusíme se zde uvést některé aspekty, které jsou buď více méně specifické pro numerický software anebo se jeví jako perspektivní a výhodně použitelné právě v této oblasti.

2. Co je numerický software ?

Numerický software budeme pro účely tohoto pojednání (a autoři doufají, že ve shodě s původním anglickým pojmem "numerical software") charakterizovat jako tu část software pro VTV, která při aplikaci matematických metod používá algoritmů a metod numerické matematiky. Typickými představiteli numerického software jsou tedy knihovny numerických algoritmů (např. NAG, IBM Scientific Soubroutine Package). Optimalizační systémy (např. OPTIPACK a SPONA), systémy pro řešení diferenciálních rovnic metodou konečných prvků (např. PMD, PROKOP, SAPIV) atd. Nebudeme považovat za numerický software systémy řešící problémy např. teorie grafů, logických hypotéz, kombinatorické problémy atd.

Numerický software jako součást disciplíny programování je jednak jednou z jejích nejstarších součástí, ale také i v současné době jednou z velmi rychle se vyvíjejících. A jaké jsou důvody tohoto rozvoje ? "Vnější " důvody spočívají ve stále větším významu aplikací nejrůznějších matematických disciplín

v průmyslu a ve vědě. Tak, jak se věda stává výrobní silou, stávají se programovací systémy "výrobními prostředky" konstruktérů a techniků pro vývoj a konstrukci strojů, při navrhování složitých konstrukčních zařízení a staveb. Využívá se modelování pomocí programových systémů pro optimalizaci, posuzování "slabých" míst konstrukcí pomocí systémů pro metodu konečných prvků atd. "Vnitřní" důvody spočívají ve velmi rychlém rozvoji aplikované matematiky a zvláště numerických metod a ve stále větších možnostech, které nabízí nejnovější počítačové systémy. Vzniká řada nových efektivních algoritmů a zároveň se objevují nově formulované problémy, které je zapotřebí algoritmicky řešit. Je tedy zřejmé, že je zapotřebí dřívější živelnost v tvorbě a vývoji programovacích systémů nahrazovat promyšlenou koncepčností s přihlédnutím k faktu, že vývoj v aplikované matematice a programování jde velmi rychle dopředu.

3. Koncepce a tvorba numerických knihoven

První otázkou, před kterou jsou tvůrci numerického softwaru postaveni, je vymezení okruhu problémů, které mají programovací algoritmy řešit, a vlastní výběr těchto algoritmů. Nebudeme zde brát v úvahu faktory pragmatického charakteru, jako je např. problém, zda vůbec takovýto systém vytvářet nebo zda jej koupit (je-li k dispozici) nebo kolik člověkoroků můžeme na tvorbu systému věnovat. Přirozené požadavky kladené potom na vytvářený systém jsou:

(1) Okruh problémů, které bude systém řešit, by měl být kompaktní a dostatečně obecně formulovaný. Kompaktností zde rozumíme tu vlastnost, že kromě problémů (obvykle s bezprostředními aplikacemi), jež "zapříčinily" vznik systému, bude systém numerických podprogramů schopen řešit i problémy blízké.

(2) Algoritmy, které budou v numerické knihovně naprogramovány, by měly být "up to date" a vysoce efektivní. Zkušenosti také však ukazují, že "osvědčenost" algoritmů (což je požadavek do jisté míry protikladný k požadavku "up to date") je požadavek stejně podstatný.

(3) Spektrum programovaných algoritmů by mělo pokrývat možné uživatelovy požadavky na specifické parametry výpočtu. Např. je vhodné, aby v optimalizačním systému byly zastoupeny jednak rychle konvergující procedury, (které však mohou poměrně často selhat) a jednak robustní procedury, které pomalu, ale spolehlivě konvergují.

Vymezení problémů a volba jisté množiny algoritmů (počáteční, protože numerické systémy jsou svým charakterem zřejmě systémy s otevřeným koncem) je počátkem dalších úvah o programátorských aspektech úkolu. Jedním z těchto aspektů je volba programovacího jazyka z dnes již pestré palety jazyků Algol 60, Algol 68, FORTRAN IV, FORTRAN 77, Pascal atd. Mnoho vlastností programovacích jazyků může mít nezanedbatelný vliv na strukturu podprogramů a rovněž na strukturu systému. Uveďme to na příkladu z knihovny NAG, kde pro řešení systému lineárních rovnic pro vícenásobné pravé strany

$$A\,X = B,$$

A je matice typu NxN a B a X jsou matice typu NxM, je vyvolání podprogramu různých programovacích jazyků následující:

```
Algol 68:          FQ4AAB(A,B,X,FAIL),
Algol 60:          FQ4AAA(A,B,N,M,X,IFAIL),
FORTRAN:    CALL   FQ4AAF(A,IA,B,IB,N,M,X,IX,W,IFAIL),
```

kde FAIL nebo IFAIL je parametr indikující chyby.

Ve vyvolání fortranského podprogramu jsou pole parametrů specifikována takto:

```
REAL A(IA,N),B(IB,N),X(IX,M),W(N)
```

Parametry IA,IB,IX specifikují vnější rozměry 2-dimensionálních polí ve volající úrovni. Tyto parametry nejsou zapotřebí ani v ALGOLu 60 ani v ALGOLu 68, protože jsou předávány implicitně. Jak pro FORTRAN, tak i pro ALGOL 60 mohou být rozměry polí A, B a X větší než rozměry odpovídajících matematických veličin. Podprogram u ALGOLu 68 požaduje shodu dimenzí a není proto nutné uvádět parametry N a M, které lze odvodit z parametrů polí operátory lwb a upb. Všechny podprogramy vyžadují pracovní pole délky N. V ALGOLu 60 a ALGOLu 68 může být toto pole dekla-

rováno dynamicky uvnitř procedury, ve FORTRANu musí být předáno prostřednictvím parametrů. Přesto, že teoretikové programování již dávno vyhlásili křižáckou válku FORTRANu (i uvedený příklad vyznívá v jeho neprospěch), je jeho rozšířenost na našich počítačích a popularita pravděpodobně převládajícím faktorem nad nestrukturovaností a dalšími neřvary tohoto seniora programovacích jazyků. Někteří tvůrci numerického software toto dilema řeší tak, že systém naprogramují ve více programovacích jazycích (to se týká např. systému NAG).

Další faktory, které je nutno v koncepci a architektuře numerického systému brát v úvahu jsou:

- Numerická přesnost; zvolení úrovně numerické přesnosti (např. jednoduché nebo dvojnásobné) je podstatné s ohledem na stabilitu numerických algoritmů. Ukazuje,že je vhodnější zvolit více než jednu úroveň přesnosti, neboť často je žádoucí, a někdy podstatné, část výpočtů dělat ve vyšší přesnosti, než zbytek (např. výpočet vlastních čísel při QR algoritmu). Dále z hlediska prověření stability provést výpočet nejprve v jednoduché a pak v dvojité přesnosti. A na některých počítačích je nejvyšší úroveň přesnosti neúměrná době výpočtu.
- Uživatelovo hledisko; typickým pro numerické procesy (samozřejmě ne jediným) je uživatelův požadavek na vyčíslení chyby (nebo odhadů chyb - např. tzv. estimátory pro metodu konečných prvků - viz /5/). V řadě případů je vhodné, aby často používané algoritmy byly uživateli k dispozici v uživatelsky jednodušší formě "EASY-TO-USE" a dablovány jako "SPECIALIST" podprogramy s plným využitím všech možností.
- Výpočetní"prostředí", na kterém se předpokládá implementace numerické knihovny. Vlastnosti "prostředí" vyplývají ze specifik software, zvláště kompilátorů, příslušných daným systémům. Zde se např. jedná o různě naprogramované vnitřní funkce, které tvoří potenciálně "Achillovu patu" numerického software, přestože jsou pro numerický software nepostradatelné. Jako odstrašující příklad uveďme některé naše zkušenosti z počítače ZPA 600, kde kompilátor chybně překládal funkci

ATAN 2 a zkušenosti z implementace knihovny NAG na počítači Prime 400, kde zhavaroval podprogram pro výpočet vlastních vektorů vlivem nepřesností ve výpočtu funkce DSQRT. Na počítači Honeywell 66 zhavaroval podprogram, který předpokládal, že $|\cos(x)| \leq 1$, což porušila funkce DCOS. Proto nezbývá tvůrcům numerického software nic jiného, než že v zoufalé snaze si jako Trojského koně vymýšlí pomocné konstanty, a tak se dostávají do nepřátelského tábora, který jim brání v implementaci jinde fungujícího software. Proto je vhodné zavést například 2 parametry závislé na "prostředí" a určující nejmenší a největší číslo, pro které lze vyčíslit funkci EXP a řadu dalších strojově závislých konstant (strojové nekonečno a různé další různé konstanty charakterizující na daném počítači zobrazení čísel a jejich rozsah).

- Modularita, strukturovanost, portabilita, vypracování konvencí a dokumentace, atd. Tyto otázky nebudeme podrobně rozebírat a odkážeme se na /1/,/2/.

4. Ladění, testování a udržování numerických knihoven

Otázka ladění a testování programů patří opět mezi okruh problémů, o kterých vycházejí úctyhodně tlusté několikadílné monografie, kterým na tomto místě nemíníme konkurovat. Autoři se zde chtějí ve vší skromnosti zmínit o svých dobrých zkušenostech s interaktivním laděním a testováním numerického software.

Zde se odkážeme na /4/ a uvedeme drobný příklad. Jistý poměrně složitý program dával poněkud jiné výsledky po překladu FORTRANem G a FORTRANem H. Po řadě analýz a kontrolních tisků nakonec zoufalý autor uvěřil, že to je způsobeno chybou překladače. S použitím interaktivního testovacího systému ITS se později asi během 20 minut ukázalo, že rozdílné výsledky byly způsobeny jemnou numerickou nestabilitou v jisté části programu. Tím, že FORTRAN H přeložil jistou sekvenci příkazů sice ekvivalentně, ale fakticky jinak, došlo na tomto místě k odlišnému vyčíslení hodnot těchže proměnných, přesto, že sémantika obou

verzí (G i H) byla stejná.

Ukazuje se, že pro ladění a testování numerického software je velmi efektivní kombinace analýzy cesty a interaktivního testování (viz /3/).

Jednou z otázek, kterou je třeba z hlediska údržby knihoven rozhodnout, je struktura a funkce knihoven v rámci operačního systému. Knihovna podprogramů může být dostupná i ve zdrojovém tvaru, ale jsou i důvody pro to, aby byla dostupná jen v přeloženém tvaru. Hovoří pro to tyto důvody:

- vylučuje se opakování kompilace téhož zdrojového tvaru a neriskuje se změna otestovaného programu při přeložení modifikovaným kompilátorem;
- nikoli nepodstatným důvodem je zde ochrana před zcizením pro přátele uživatelů.

Udržování a manipulace s rozsáhlými systémy numerických podprogramů je úkol jehož náročnost je impulsem ke vzniku software sloužícímu k automatizaci těchto prací. Takový software je využíván např. k

- standardizování textu
- ověřování dialektu
  (např. ověření, zda program odpovídá dialektu PFORT - Portable FORTRAN)
- změnám operačních vlastností zdrojového textu
  (např. změna aritmetické přesnosti)

atd., viz např. /7/.

Přehled literatury :

/1/ Henzl P., Hřebíček J.: Programování vědeckotechnických výpočtů. Programování '81, Sborník přednášek ze semináře 1981.

/2/ Bébr R.: Zajímavosti ze světa vědeckotechnických výpočtů. Programování '80, Sborník přednášek ze semináře, 1980.

/3/ Kopeček I.: Verifikace programů metodou interaktivního testování v systému ICL 2950/10. MOP '82, Sborník přednášek, Pezinok 1982.

/4/ Kučera J.: Zkušenosti z tvorby a ladění programů VTV pod operačními systémy 4. generace. Programování '83, Sborník přednášek ze semináře, 1983.

/5/ Babuška I.: The Reliability Estimates and Adaptivity in F.E.M. Computations. in: Finite Elements in Water Resources, Hanover 1982.

/6/ Du Croz J.: Programming Languages for Numerical Subroutine Libraries. NAg Newsletter 1/82, 1982.

/7/ Hague S.J., Ford B.: Tools for Numerical Software Engineering. NAg Newsletter 1/81, 1981.