MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 20/1978.

EGY SZÉLES KÖRBEN ALKALMAZHATÓ PROGRAMOPTIMALIZÁLÁSI MÓDSZER

Ruda Mihály

1. Bevezetés

A számológépgyártásban és- felhasználásban sokféle optimalizálási törekvésnek lehetünk tanui. Egy adott hardware-konfiguráció hatékonyságát a gép software-je is nagymértékben befolyásolja, a software fejlesztését pedig a gép felhasználási köre határozza meg. A software-fejlesztés – pl. a magasszintű nyelvek implementásása, a fejlett operációs rendszerek, célnyelvek, felhasználói programrendszerek, programkönyvtárak – célja általában a gépfelhasználás hatásfokának javitása (az emberi munkaerő és a hardware eszközök minél jobb kihasználása).

Vannak azonban olyan tényezők is amelyek a feladatok gépigényének (idő, tároló) minimalizálását követelik. Ez a kérdés különösen élesen vetődik fel akkor, amikor a rendelkezésre álló gépkapacitás határát érintő feladatokat kell megoldani.

Ilyen célt szolgálnak például a forditóprogramok optimalizálási törvényei ([7], [10]), vagy a multiprogramozással kapcsolatban az optimális kiszolgálási stratégiák kidolgozása ([1], [3]). Az utóbbi kérdéskörön belül lényeges szerepet játszanak a statitisztikai és valószinűségszámitási meggondolások ([2]).

Ezeken az optimalizálási törekvéseken túl, más módon is javitható egy számológép működésének hatásfoka. Ide soroljuk azt a feladatot is, amely egy állandó jelleggel használt program (programrendszer) "általános megfogalmazása" és az esetenként futó speciális feladat között lévő lényeges különbség felszámolását jelenti. Részletesebben a következőkről van szó:

Azok a programok amelyek egy általános feladatkör megoldására készülnek (pl. lineáris egyenletrendszerek megoldása), általában paraméterek megadásával vezérelhetők a kitűzött feladat elvégzésére. Ezek a programok esetenként a futás közben sokszor ismétlődő eljárásokon belűl is ujra és ujra megvizsgálják a vezérlő paramétereket, illetve ujra és ujra ismétlődő számitásokat végeznek velük.

A probléma onnan ered, hogy a programok forditási és futtatási fázisa – a szokásos compiler-eknél – élesen szétválik. Ezért olyan adatok, amelyek konkrét esetekben állandó értékek, változóként szerepelnek, sőt néha a program bonyolult eljárásokkal feleslegesen hozza létre őket ujra és ujra. Ugyanigy egyes magasszintű nyelveken (pl. FORTRAN, COBOL) nem valósitható meg a (valóban) dinamikus memóriakihasználás (az ALGOL lehetővé teszi változó méretű tömbök alkalmazását). Egyszerü programoknál a felhasználó (a programozó) kézi beavatkozással – pl. FORTRAN programban egy DIMENSION kártya cseréjével – is módosithatja a programot, a soronlévő feladat igényeinek megfelelően. Egy bonyolult programrendszernél azonban semmiképpen sem lehetséges egy futásonként ismétlődő, kézi beavatkozással történő programváltoztatás – még a legegyszerűbb formában sem. Ez méginkább igy van, ha maga a módositás is bonyolult.

A helyes megoldás egy olyan operációs rendszerváltozat kialakitása, amely az általános használhatóság mellett lehetővé teszi azt, hogy a programok forditása egyben egy "átdolgozás" is legyen, sőt azt is, hogy futás közben módosithassuk a már leforditott programot is (ugyanugy mint a gépikód programoknál). Átdolgozás alatt itt azt értjük, hogy a rendszer, a felhasználó által adott paraméterek függvényében, futásonként más és más módon értelmezi a programot.

Az oprációs rendszerek néhány ujszerű feladaton túl (mint pl. az előbb már emlitett multiprogramozás futásütemezése) elsősorban a hagyományos "kézi" gépkezelési feladatokat (forditás inditás, programbetöltés, tárolók nyitása, zárása, szalagok vezérlése, stb.) látnak el. Felhasználói munkák kezelése más módon és más tipusu területeken is történhet mint a hagyományos "kézi" operátori munka. A következőkben egy ilyen lehetőséggel foglalkozunk. Egy olyan eljárás alkalmazását mutatjuk be, amely a szokásos operációs rendszerek segitségével is lehetővé teszi a programok automatikus átdolgozását (esetenkénti ujraértelmezését) – pl. optimalizálási céllal.

2. Az alkalmazott módszer leirása

A módszer, amelynek alkalmazására a 3. pontban példákat is adunk, a következő: Az optimalizáláni kivánt programot úgy készitjük el, hogy az egyes futásoknál az aktuális paramétereket egy előkészitő program illeszti a futó programba. A feladat tehát két program irása: az első az előkészitő program, amely a futó programot az aktuális paramétereknek megfelelően összeállitja; a második programnak (a tulajdonképpeni feladatot ellátó programnak) csak az állandó részeit készitjük el előre, a változó részeket az előkésztitő program illeszti be. A futó program igy mindig egy a pillanatnyi helyzetnek megfelelő optimális változatban működhet.

A módszer hatékonyságát a következő szakaszban bemutatott példákkal illusztráljuk. Ezekben a példákban bemutatott eljárásokat a szerző sikeresen alkalmazta egy Honeywell Bull 66/60-as gépen készülő nagyméretű információs rendszer megvalósitásánál. Az egyes optimalizálási eljárások programozásában GÁL ANNA, RATKÓ ISTVÁN és VASS RÓZSA vett részt. A rendszer egy vázlatos leirása [13]-ban található. A módszerrel kapcsolatos első eredmények az MTA SzTAKI-ban működő CDC 3300-as gépen születtek 1975 tavaszán (ld. [4], [5], [6]).

Az itt bemutatott módszerrel kapcsolatban két dolgot fontos tisztázni: 1. az előkészitő programok előállitása nem túl bonyolult-e, 2. a program (rendszer) felhasználóját nem terheli-e feleslegesen az uj technikai alkalmazása.

Egy közvetve adódó de igen jelentős előnyre kell még felhivni a figyelmet. Az esetenként ujra és ujra összeállitott program mindig egy-egy speciális feladatot old meg, tehát az általános célhoz viszonyitva egyszerű és könnyen áttekinthető lesz a program. Ez pl. a hibakeresést (elsősorban a programirás fázisában, de a felhasználásnál is) rendkivüli módon megkönnyiti.

Most bemutatunk néhány példát. Hangsúlyozni kell, hogy a vizsgált módszert olyan esetekben célszerű alkalmazni, amikor egy sokszor ismételt futtatásra szánt nagy hely- és időigényű programot készitünk. Csak a futásiidő csökkentését illusztráló példákat közlünk. A tárolóigény csökkentésénel lehetőségét például azzal biztosithatjuk, hogy lehetővé tesszük az adattömbök méretének futásonkénti (vagy futás közbeni) változtatását (úgy mint pl. az ALGOL-ban). Természetesen egy előkészitő (programszerkesztő) program alkalmazásával az adatterületek nagyságának minimalizálásán túl még más lehetőségek is vannak a tárolóigény csökkentésére, például az éppen futó programváltozatban felesleges utasitások elhagyásával.

3. Példák

1. Az itt következő példával csak röviden foglalkozunk, a probléma részletesebb leirása a [8], [11] és [12] dolgozatokban található meg. A feladat a következő: Előre nem ismert, bonyolult logikai kifejezések kiértékelését kivánjuk megoldani. A feladatot még az is nehezíti, hogy a kifejezésekben szereplő változók értékét csak igen nagyméretű értéktáblázatokkal tudjuk megadni.

Változó formájú és tartalmú logikai kifejezések programba való épitése úgy is megoldható, hogy a felhasználó – mondjuk FORTRAN nyelven (FORTRAN program esetén) – leirja a feltételt, mindig az éppen szükséges formájában, és egy szerkesztő eljárással a megfelelő helyre illeszti. Ez tuladonképpen már az általunk javasolt módszer egy változata. Sokkal bonyolultabb a probléma akkor, ha a változók igen sokféle értéket vehetnek fel, esetleg olyan értékek szerepelnek amelyek más eljárások eredményeként adódnak. Ilyenkor bonyolult logikai kifejezések és terjedelmes értéktáblázat rendszerek leirását és vizsgálatát kell megoldani. Ebben az esetben az előkészitő program fő célja az aktuális logikai kifejezések lehető leggazdaságosabban kiértékelhető változatának összeállitása.

2. Ebben a példában azt az általános elvet valósitjuk meg, mely szerint, ha egy függvényt egy ismételt eljárás folyamán sokszor kell kiszámitani, akkor az ismételt számitások helyett egy értéktáblázatot készitünk, és ezután ebből másoljuk ki a szükséges függvényértékeket. (Tulajdonképpen ez az elv lett alkalmazva az 1. példa esetében is – ld. pl. [12]).

A vizsgált feladat a következő: Fix rekordok kijelölt adataiból készitünk gyakoriságtáblázatot (pl. egy populáció kor és nem szerinti megoszlását határozzuk meg). Ha általános formában kivánjuk megoldani a feladatot – akárhány dimenziós táblázatot készithetünk a rekord tetszőleges adatiból – akkor egy bonyolult, többszörös indexezésekkel müködő programot kell irni. A táblázat értékeinek gyüjtése, vagyis az aktuális táblaindex beállitása pl. a következőképpen irható le:

```
      IND = L(S(M))– KA(M) + 1
(1)   DO 1 I = 1,N
    1 IND=IND + T(I) * (L(S(I)) – KA(I)),
```

ahol M a táblázat változóinak száma,

N = M – 1,
L a rekodelemeket tartalmazó vektor,
S a táblázat változóinak (a rekordon belüli) sorszámait tartalmazó vektor,
KA a táblaadatokra adott alsó korlátok vektora,
T(I) az I-edik szintű résztáblák terjedelme,
azaz T(M) = 1, T(M – 1) = KF(M) – KA(M) + 1, stb.,
KF a táblaadatok felső korlátainak vektora,
IND a számitandó index.

Ezt a programrészt a következő FORTRAN nyelvű változatokkal is megoldhatjuk, ha már ismerjük az elvégzendő feladatot, amely mondjuk egy háromdimenziós táblázat készitése:

(2) IND = T1 * L1 + T2 * L2 + L3 + D,

ahol az előző jelölések szerint

T1 = T(1),
T2 = T(2),
L1 = L(S(1)),
L2 = L(S(2)),
L3 = L(S(3)) és
D = 1 – KA(3) – T(2) * KA(2) – T(1) * KA(1).

Ha a futtatás előtt ismerjük a feladat paramétereit, akkor T1, T2 és D kiszámitható, az L1, L2, L3 változónév pedig hozzárendelhető a rekord kivánt elemeihez. Ezeket a feladatokat (a számitásokat és a hozzárendeléseket) az előkészitő program végzi el.

A (2)-höz hasonlóan használhatjuk az

(3) IND = T1(L1) + T2(L2) + T3(L3) . . .

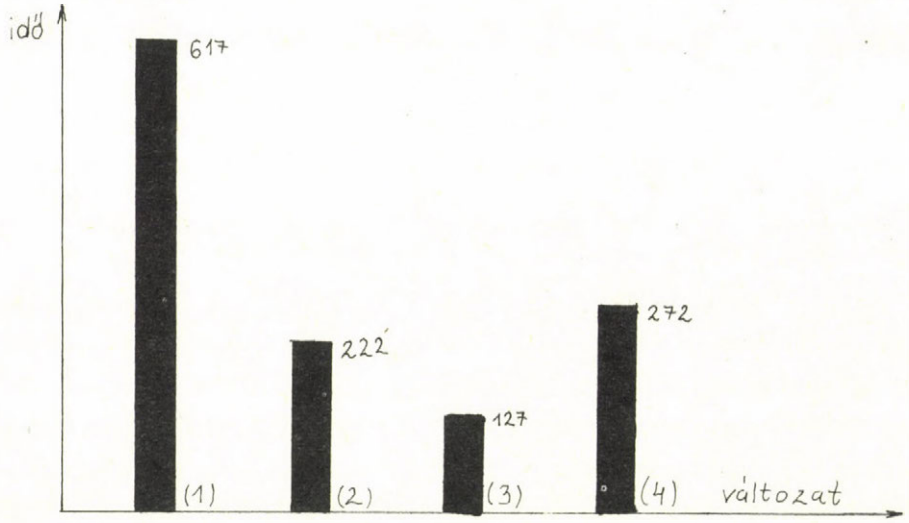
utasitást, ha az L1, L2, L3, . . . adatok lehetséges értékeinek megfelelő indexnövekményeket – azaz a T(I) * (L(S(I)) – KA(I)) , (I = 1, . . . ,N) és az L(S(M)) – KA(M) + 1 értékeket – előre elhelyezzük a T1, T2, T3, . . . táblázatokban.

Ha az adatok (rekordelemek) között negativ értékek is előfordulnak, vagy ha a KA(I) alsó korlátoknál kisebb értékeknek nem kivánunk feleslegesen helyet fenntartani, akkor a következő megoldást használhatjuk:

```
J1 = L1 – KA1
J2 = L2 – KA2
J3 = L3 – KA3
IND = T1(J1) + T2(J2) + T3(J3) . . . ,
```

ahol KA1 = KA(1), stb. (Meg kell jegyezni, hogy az első három utasitást EOUIVALENCE utasitások segitségével kiküszöbölhetjük.)

Jól látható, hogy a (2) – (4) programrészlet – főleg a (3) – lényegesen gyorsabb mint az (1) változat. A tényleges sebességkülönbségek azonban függnek a felhasznált számológéptől is. A négyféle programvariáció futásidő arányát, egy Honeywell Bull 66/60-as gépen történt kisérlet alapján az 1. ábra mutatja.



1. ábra

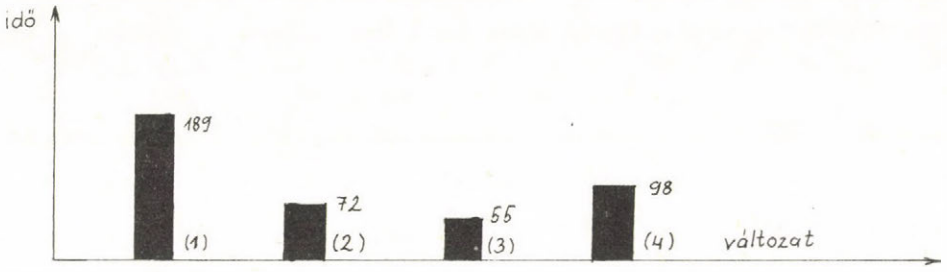
Az ábrán az időértékek tizezred-órában adottak. A kisérleti programban M = 10 volt, és az indexszámitási eljárást egymilliószor ismételtük. Az eredeti (1) változat és a leggyorsabb (3) változat időaránya

$$617/127 \sim 4.9.$$

Mivel a közölt értékek kisérleti adatok, ezért (a pillanatnyi futási környezet függvényében) ingadozhatnak, de ezek az ingadozások nagyságban messze elmaradnak a fenti, közel ötszörös sebességnövekedéshez viszonyitva.

Az időarány függhet az M értékétől is. M = 3-ra (ugyancsak egymillió ismétléssel) a következő időadatokat kaptuk (ld. 2. ábra). Itt a két szélső időérték közti arány:

$$189/55 \sim 3.4.$$



2. ábra

3. A most soronkövetkező példa egy még nagyobb mértékű futásidő csökkenést mutat be.

A legtöbb esetben a feldolgozandó adatok karakteres formában kerülnek a felhasználó kezébe, de a szükséges eljárások lényegesen gyorsabban hajthatók végre bináris adatokkal. Iyen esetekben konverziót alkalmazunk. A konverzió általában időigényes eljárás. Végrehajtási ideje nagymértékben függhet attól, hogy a konverzió egy külső tárolóból való olvasással párosul-e, vagy csak a belső tárolóban történik. Mindkét esetben figyelembe kell venni a felhasználatlan karakterek szerepét is. A külső tárolón, vagy a memóriában hézagosan elhelyezett karaktermezők konverziója általában több időt igényel mint a tömören elhelyezetteké. Néhány példa egy Honeywell Bull 66/60-as gépen mágnesszalagról FORTRAN programmal beolvasott 150 ezer rekord esetén (itt és az előzőkben is, az időértékek a központi egység által felhasznált időt mutatják):

FORTRAN formátum	idő(órában)
77X,I2,I2	0.1508
A80	0.1549
4X,I2,17X,I2	0.0951
22X,I1,84X,I3,31X,I2	0.2421

1. táblázat

A konverzió sebességének növelésére alkalmazott módszer az itt bemutatott formájában csak nem negativ egész értékek konvertálására használható (igaz, hogy adatfeldolgozási feladatoknál ez a leggyakoribb eset). A módszert kétjegyű szám esetére mutatjuk be. A konverziót most nem kiséri beolvasási művelet.

A konvertálandó két karakter az N1, N2, . . . szavakban foglal helyet, mondjuk az

N5 szó 6-11 és 12-17 bitjein (egy karakter 6 bit). Ekkor az M szóba irányuló konverzió a következőképpen oldható meg:

```
I1 = FLD(6,6,N5)
I2 = FLD(12,6,N5)
M = M1(I1) + I2
```

ahol M1(I) = I * 10, és az FLD függvény (Honeywell FORTRAN könyvtári eljárás) értéke a következő: FLD(a,b,c) a c szó b bitje a-adik bittől kezdve. Ez a b drb. bit kerül a baloldalon álló változó alsó helyiértékeire. Az utasitásokat itt is egy előkészitő program állitja össze.

Ezt az eljárást a hagyományos konverzióval összehasonlitva, 300 ezer ismétlés esetén a következő adatokat kaptuk (2. táblázat):

konverziós forma	idő
30X,I2 formátum	0.1169
a gyorsitott eljárás	0.0014

2. táblázat

Az arányokat (majdnem százszoros különbség) az előzőkhöz hasonló diagramon nem is tudjuk bemutatni. A sebességnövekedés egyik nyilvánvaló oka az, hogy a bemutatott eljárás független a felhasználatlan karakterek számától.

4. Összefoglalás

A 2. pontban javasolt általános módszer hasznossága a bemutatott példák alapján nyilvánvaló. A figyelmet arra kell még felhivni, hogy az előzőkben példákon keresztül bemutatott eljárások (az előkészitő program alkalmazása) nagyméretű feladatoknál nem csak hasznosak, de sokszor nélkülözhetetlenek is. A módszer gyakorlati alkalmazhatóságát a 2. pontban emlitett adatfeldolgozó rendszerben történt sikeres felhasználása bizonyitja.

I r o d a l o m

[1] Aho A.V. – Denning P.J. – Ullmann J.D.: Principles of optimal page replacement, J. Assoc. Comput. Mach., 18. (1971).

[2] Benczúr A. – Krámli A. – Pergel J.: On the Bayesian approach to optimal performance of page storage hierarchies, Acta Cybernetika, 3., (1977).

[3] Coffman E.G. – Kleinrock L.: Computer Scheduling Methods and their Countermeasures, Proceedings, AFIPS (1968), SJCC.

[4] Csukás A-né, – Greff L. – Krámli A. – Ruda M.: A kórházi morbiditási vizsgálat számitógépes feldolgozásának tapasztalatai és továbbfejlesztése, Számitástechnikai és kibernetikai módszerek alkalmazása az orvostudományban és a biológiában, 5. Kollokvium, Szeged, (1974).

[5] An approach to the hospital morbidity data system development in Hungary, Symposium on Medical Data Processing, Toulouse, (1975).

[6] Lekérdező rendszer kórházi morbiditási vizsgálat anyagára, Számitátechnikai és kibernetikai módszerek alkalmazása az orvostudományban és a biológiában, 6. Kollokvium, Szeged, (1975).

[7] Finkelstein M.: A Compiler Optimization Technique, Computer Journal, 2., (1968).

[8] Krámli A.,– Ratkó I. – Ruda M. – Soltész J.: A statisztikai adatfeldolgozás matematikai és számitástechnikai problémái, MTA SzTAKI Tanulmányok, 70/1977.

[9] Knuth J.D.E.: The art of computer programing, Sorting and Searching (3. kötet), Addison-Wesly, London, California, (1973).

[10] Nievergelt J.: On the Automatic Simplification of Computer Programs, CACM, 8., (1965).

[11] Ratkó I.: Egy számitástechnikai eszköz bonyolult logikai kifejezések leirása orvosstatisztikai alkalmazásban, Számitástechnikai és kibernetikai módszerek alkalmazása az orvostudományban és a biólogiában, 8. Kollokvium, Szeged, (1977).

[12] Ratkó I.: Bonyolult logikai kifejezések kiértékelésének számitástechnikai és optimalizálási problémái, MTA SzTAKI Közlemények, 20/1978.

[13] Ruda M.: Egy általános információs rendszer kórházi morbiditási adatok feldolgozására, Számitástechnikai és kibernetikai módszerek alkalmazása az orvostudományban és a biológiában, 8. Kollokvium, Szeged, (1977).

Summary

A generally applicable program optimizing method

Mihály Ruda

There are various endeavours for effective use of computers, for example automatic program optimization at compile time, computer scheduling methods, etc. This paper examines a computer program optimization method applicable in operating systems too. Examples (for the decrease of compute time) show the effect of the method (cf. fig. 1., 2. and table 2.). The author applied this method successfully in a medical information system. The method works by a preparatory (editor) program which recomposes the source program before each run.

Р Е З Ю М Е

ОБ ОДНОМ МЕТОДЕ ОПТИМИЗАЦИИ ПРОГРАММ,
ИМЕЮЩЕМ ШИРОКОЕ ПРИМЕНЕНИЕ

Михай Руда

Известны различные способы повышения эффективности ЭВМ, например, автоматическая оптимизация трансляторов, использование различных стратегий при выборе режима работы операционной системы и др.

В данной работе рассматривается метод оптимизации программ, который может быть использован и в операционных системах. Эффективность метода /уменьшение машинного времени/ иллюстрируется на конкретных примерах /см. рис. 1 и 2, а также табл. 2/. Описываемый метод автор с успехом использовал при разработке одной информационной системы. Суть метода: оптимизируемую программу составляет специальная подготовляющая программа /редактор/.