Doktori értekezés

Varjasi Norbert

Széchenyi István Egyetem, Műszaki Tudományi Kar

2012

Varjasi Norbert

Nagy számításigényű feladatok megoldására szolgáló párhuzamos algoritmusok vizsgálata sokprocesszoros környezetben

doktori értekezés

témavezető:
Dr. Molnárka Győző
Széchenyi István Egyetem

Infrastrukturális Rendszerek Modellezése és Fejlesztése
Multidiszciplináris Műszaki Tudományi Doktori Iskola

Tartalomjegyzék

1. BEVEZETÉS

Az informatikai rendszerek utóbbi évtizedekben végbemenő gyors fejlődésével, és a processzorgyártás technológiai újdonságainak széleskörű elterjedésével nyilvánvalóvá vált, hogy a hardver-fejlesztés iránya a sokprocesszoros rendszerek felé mozdul el. A számítástechnikai modellek tervezésének és fejlesztésének ezt a változó környezetet követnie kell, ezért szükséges az olyan szoftverrendszerek megalkotása, amelyekkel az ilyen sokprocesszoros rendszereken hatékony és gyors alkalmazások fejleszthetők.

A sokprocesszoros rendszerekre történő szoftverfejlesztések szerteágazó volta miatt az érdeklődésünk egyre inkább az olyan területekre irányult, amelyek a konkrét feladatok megoldása mellett a párhuzamos számítási rendszerek univerzális felhasználhatóságát jelentik. Egy konkrét ipari feladat sokprocesszoros alkalmazásának kidolgozása során nyilvánvalóvá vált, hogy a nagyméretű lineáris egyenletrendszerek párhuzamos megoldása a párhuzamos programozás egyik központi és hangsúlyos problémája. Ennek az állításnak az igazolása a párhuzamos programozási szakirodalom feldolgozásakor is megerősítést nyert. Egyes vélemények szerint az egyre bonyolultabb problémák megoldása azon is múlhat, hogy ki milyen méretű és milyen bonyolultságú lineáris egyenletrendszereket tud gyorsan és hatékonyan megoldani. A sokprocesszoros környezetre optimalizált, lineáris egyenletrendszereket megoldó algoritmusokon alapuló szoftverek felhasználhatók többek között az adatbányászatban, a nagy adatrendszerek optimalizálásában, a processzortervezésben, a repülőgép- és űriparban, az energetikában, és nem utolsó sorban az időjárási és környezetvédelmi feladatokban.

A magyarországi adottságok mellett, a kutatások megkezdésekor abban nem reménykedhettünk, hogy ilyen feladatokhoz és az ezekhez használható óriási számítógépekhez hozzáférhetünk, ezért a figyelmünk a felsorolt témakörök helyett inkább az alapkutatások felé fordult. A lineáris egyenletrendszerekkel való foglalkozást ez indokolja.

A kutatómunka folyamán figyelmünk egy másik informatikai szakterület felé, de szintén univerzális problémára irányult. Ebben a párhuzamos algoritmusok nyomkövetésének, teljesítmény mérésének és a nemdeterminisztikus működés jelenségeinek elméleti modellekkel való alátámasztását tűztük ki célul.

A dolgozatban egy – a homogén párhuzamos rendszerekben használható – speciális modellt mutatunk be, amely a sokprocesszoros rendszerek hierarchikus topológiájának kiegyensúlyozottságának mérésére ad leírást. A témaválasztást az indokolja, hogy a kifejlesztett algoritmusok tesztelése során ugyanazok a párhuzamos programok eltérő architektúrákon, vagy eltérő kezdőértékekkel elindítva nagyon különböző viselkedésű futási eredményeket produkáltak.

A kutatás kezdetén kijelölt és bemutatásra kerülő témakörök időszerűségét indokolja a munka során felmerült számos nyitott és napjainkban is aktívan kutatott probléma jelenléte. A témaválasztás aktualitását igazolja továbbá az a számos – és folyamatosan szaporodó – nemzetközi konferencia, ahol az érintett témákban aktív kutatói tevékenység folyik.

A disszertáció felépítése és az alkalmazott módszerek

A dolgozat három fő részből áll. Az első rész a kutatásban érintett három fő tudományterület tömör és összefoglaló elemzésével foglalkozik. Ezen belül a *2. fejezet* a sokprocesszoros és elosztott rendszerek bemutatásáról, a *3. fejezet* a párhuzamos algoritmusok tervezéséről és a párhuzamos programok mérési elveinek és szoftver komponenseinek bemutatásáról szól. A *4. fejezet* a kutatások során érintett lineáris egyenletrendszerek megoldására szolgáló numerikus matematikai tanulmányok összefoglalását tartalmazza. A három témakör összekapcsolása azért indokolt, mert a mai heterogén, vagy hibrid számítógépes rendszerekben a hatékony működés elérése érdekében mindhárom témakör érintett.

A második fő részben fejtjük ki a kutatás során kidolgozott párhuzamos algoritmusokat illetve a méréshez használt modelleket. Az *5. fejezetben* egy valósidejű, ipari környezetben működő nagy mennyiségű adatot feldolgozó elosztott-párhuzamos rendszer leírása található. A *6. fejezetben* bemutatjuk egy reziduum minimalizáción alapuló algoritmus párhuzamos modelljét, majd ennek számítógépes realizációját, és a mért adatok kvantitatív elemzését. Ezután – a homogén és heterogén rendszereken való futtatások vizsgálata alapján – közöljük a párhuzamos modell egyenrangú és hierarchikus topológiájú modellel megvalósított eltérő működésének összehasonlító elemzését is.

A *7. fejezetben* egy újabb, az ún. speciális altér-dekompozíciót használó általános, nagyméretű és rosszul kondicionált lineáris egyenletrendszerek megoldására szolgáló algoritmus párhuzamos környezetben elkészített modelljét mutatjuk be. A számítógépes futási

eredmények elemzése mellett megadjuk a működést javító változat leírását is, figyelembe véve az algoritmus evolúciós jellegét. Ugyanitt megmutatjuk, hogy a párhuzamos algoritmusok optimalizációjában a műveletek száma és a futás ideje egymástól független, külön-külön optimalizálható probléma. Az algoritmusok modellezése és fejlesztés során kiindulásként PRAM, illetve üzenetküldés alapú modelleket használunk. Az elkészült szoftvereket ismert modellek alapján validáltuk.

A *8. fejezetben* bemutatunk egy olyan új, párhuzamos és elosztott algoritmusok fejlesztésénél használható speciális modellt – homogén, egységes költségű rendszerekre – amely a hierarchikus szervezésű párhuzamos algoritmusok szervezésében a processzorok egyenletes terhelés-eloszlására mutat be mérhető értékeket. Az általunk definiált új modell megadásánál a *Sleator* és *Tarján* által kidolgozott (bináris fák kiegyensúlyozását leíró) modell kiterjesztését általános fákra, és az amortizációs potenciálok módszerét használtuk fel.

A harmadik részbe került a dolgozat tézisek összefoglalása és legvégül a feldolgozott irodalom jegyzéke.

Köszönetnyilvánítás

A disszertáció elkészítéséhez először is szeretném megköszönni konzulensem Dr. Molnárka Győző türelmes és segítőkész munkáját. A kutatási eredmények elérését a Széchenyi István Egyetem kollégáinak támogató munkája segítette. Így köszönöm Dr. Keviczky László és Dr. Kóczy T. László professzor uraknak a Doktori Iskoláért tett fáradozásukat, Dr. Horváth Zoltánnak a *Szimuláció és optimalizáció* c. projekt keretében nyújtott támogatását, Dr. Lencse Gábornak a kutatólaboratóriumi hozzáférést, Dr. Benyó Balázsnak a szakmai támogatást, Hatwágner Miklósnak és Dr. Varga Ágnesnek a közös kutatást és a szaknyelvi segítséget.

A kutatómunka anyagi hátterét OTKA, TÁMOP, GOP projektek támogatása és az Universitas-Győr Alapítvány ösztöndíja jelentette.

2. PÁRHUZAMOS ÉS ELOSZTOTT RENDSZEREK

2.1. Több processzort használó rendszerek kialakulása

Az ipari, vagy tudományos munka során, nagyon sok feladat megoldásakor a hagyományos egyprocesszoros számítógép sok esetben már nem képes ésszerű idő alatt eredményt adni. Vagy a futás ideje bizonyul túlzottan hosszúnak, máskor – a feladat bonyolultsága miatt – a memória, vagy a háttértár találtatik szűkösnek. Előfordul az az eset is, hogy egy kutatás, vagy laboratóriumi mérés során a pontos megoldás megtalálására hosszú idő áll rendelkezésre, azonban egy napi rutintevékenység megoldásakor (pl. gyártósorokon) az eredményeket szigorú időkorlátok terhe alatt kellene megkapni. Az ilyen feladatokhoz a hagyományos szekvenciális algoritmusokkal megvalósított szoftver-megoldások nem elégségesek, helyettük párhuzamos algoritmusokra és ezek futtatására alkalmas hardverre van szükség.

Ez a fejezet a jelenleg leginkább elterjedt párhuzamos és elosztott rendszerű architektúrák bemutatásával, majd ezek elemzésével foglalkozik. Rövid történeti összefoglaló után a párhuzamosan végrehajtható alkalmazások leggyakoribb hardver- és szoftver-komponenseit mutatjuk be, és hasonlítjuk össze. A bemutatás a problémakör kiterjedtségére való tekintettel csak a legfontosabb elemekre fókuszál.

Az elosztott alkalmazásokat használó rendszerek kialakulása

A párhuzamosan futtatható algoritmusokról már Lady Lovelance Byron is írt feljegyzéseket 1842-ben, még jóval a számítógépes korszak előtt. A párhuzamos technológia fogalma már Neumann-nál megjelent 1949-ben, melyet később a Cray és Cyber vektorszámítógépekben realizáltak [Kallós 04]. A '80-as években a számítógépek számítási kapacitásának és adattárolási lehetőségeinek növekedésével megjelennek a többprocesszoros, nagy teljesítményű számítógépek. A processzorok és a memória közötti elérési idő csökkentésével speciális (pl. a hiperkocka) kapcsolású architektúrák kifejlesztésével már valódi párhuzamos gépek álltak rendelkezésre.

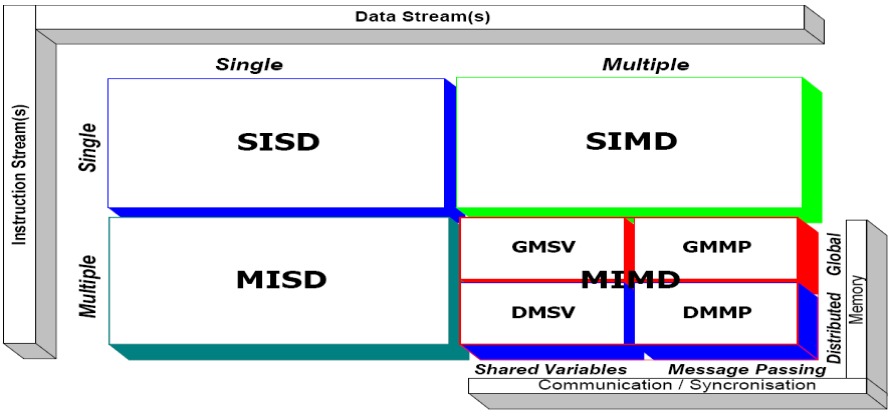
A tudományos igényű feladatok megoldásához az egyprocesszoros, Neumann-elvű rendszerek már a kezdet kezdetén kevésnek bizonyultak [Yasar 01]. Az elmúlt 20-30 évben a számítógépek teljesítménye rohamosan nőtt, így mára hatékony párhuzamosítással a nagy bonyolultságú feladatok is megoldhatókká váltak. A '80-as évektől kezdődően egyes

tengerentúli kutatások (Egyesült Államok, Japán) célkeresztjében az elosztott feladatok álltak. Ezekben a kutatásokban a fejlesztők a párhuzamos kódokat megadott architektúra alá, adott célszámítógépekre fejlesztették. A '90-es években a processzorok közötti hatékony kommunikáció, a fejlesztő és futtatókörnyezet kialakítása állt a középpontban. Kialakultak a skálázható és egységesített környezetek (*PVM – Parallel Virtual Machine, MPI – Message Passing Interface, HPF – High Performance Fortran, OpenMP*). A sikeresen kifejlesztett és alkalmazott párhuzamos programozási paradigma ezután nemcsak a homogén rendszerek esetén (szuperszámítógépek, szorosan integrált rendszerek, klaszterek), hanem a heterogén elosztott, vagy a fizikailag távol eső gépekre (ún. grid-rendszerek) is kiterjeszthetővé vált [MSZGRID].

Napjainkra a grid-rendszerek behálózzák az egész világot, melyben a három nagy kontinens (Amerika, Európa, Ázsia) együttműködő szervezetein keresztül hihetetlenül nagy számítási kapacitás áll a kutatók rendelkezésére (metaszámítógépek).

2.1.1. Multiprocesszoros és multi-core rendszerek

Akkor beszélünk többprocesszoros – vagy multiprocesszoros – rendszerről, ha egy számítógépen belül kettő, vagy több processzort használunk. Ezek hardveres megvalósítása különböző lehet (több processzor egy alaplapon, több mag egy processzoron belül, stb.).



1. ábra: Architektúrák Flynn-féle felosztása. Forrás: [Morrison 03, p. 19.]

A többprocesszoros számítógépes rendszereket legelőször Flynn osztályozta a '70-es évek elején [Tanenbaum 01]. Felosztását az adatokon egy időben végzett műveletek számával jellemezte. Az *1. ábrán* az alábbi négy fő kategória található [Kallós 04]:

- **SISD** – (Single Instruction stream / Single Data stream): A hagyományos Neumann--elvű szekvenciális adatfeldolgozásra képes számítógépek halmaza.
- **SIMD** – (Single Instruction stream / Multiple Data stream): A vektor- vagy tömbprocesszoros gépcsalád.
- **MISD** – (Multiple Instruction stream / Single Data stream): A gyakorlatban nem használt kategória.
- **MIMD** – (Multiple Instruction stream / Multiple Data stream): A valódi elosztott alkalmazásokra alkalmas gépcsaládok.

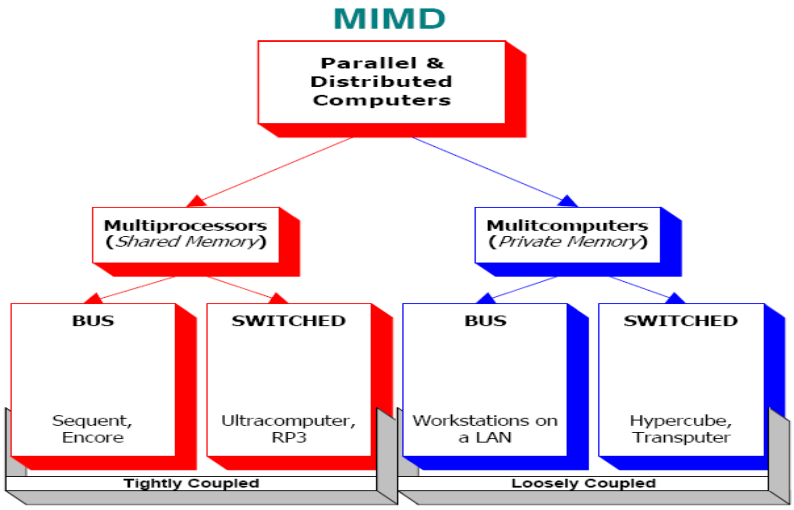
Napjainkra a tisztán *SISD*, és a *SIMD* rendszerű gépek már ritkák, illetve ezen működési elvek beépültek a modern processzorokba. Az *MISD* kategória elméletig lehetséges ugyan, de nem alkalmazzák. A többprocesszoros és elosztott rendszerekben gyakorlatilag az *MIMD* rendszerek használatosak, így az alábbiakban ezeket részletezzük. Általánosságban elmondhatjuk, hogy ezen felosztás szerint, a klasszikus többprocesszoros rendszerekben minden processzor egyenrangú, szimmetrikusan kiépített, bár egyes processzorok elláthatnak speciális vezérlési feladatokat is. Az MIMD rendszereket (lásd: 1. ábra) a *memóriahozzáférés* és a *kommunikáció* szempontjából további négy alkategóriára bonthatjuk [Morrison 03] alapján:

- **GMSV** – (Global Memory / Shared Variables): Közös memória- és megosztott változókezelést használó rendszerek.
- **GMMP** – (Global Memory / Message Passing): Közös memóriakezelés üzenetküldéssel.
- **DMSV** – (Distributed Memory / Shared Variables): Elosztott memóriakezelés megosztott változóhasználattal.
- **DMMP** – (Distributed Memory / Message Passing): Elosztott memóriakezelés üzenetküldéssel.

Ettől eltérő csoportosítást ad a Tanenbaum-féle felosztás (*lásd: 2. ábra*). Ebben az *MIMD* csoportot *szorosan- (tightly-coupled)*, illetve *lazán kapcsolt (loosley-coupled)* rendszerekre bontjuk (más szóhasználattal multiprocesszoros, illetve multiszámítógépek). Tovább bővíthetjük a felosztást a processzorok közötti kapcsolatok megadásánál, hiszen a hatékony működés feltétele a jól menedzselt összeköttetések biztosítása. A legelterjedtebbek a busz (vagy

hálózatos) rendszerű, illetve a kapcsológépekkel megadott összeköttetést használó architektúrák.

Láthatjuk, hogy míg a Flynn-féle felosztás a rendszerek programozási modelljei alapján, addig a Tanenbaum-féle felosztás a hardver architektúra alapján osztályozza az elosztott rendszereket.



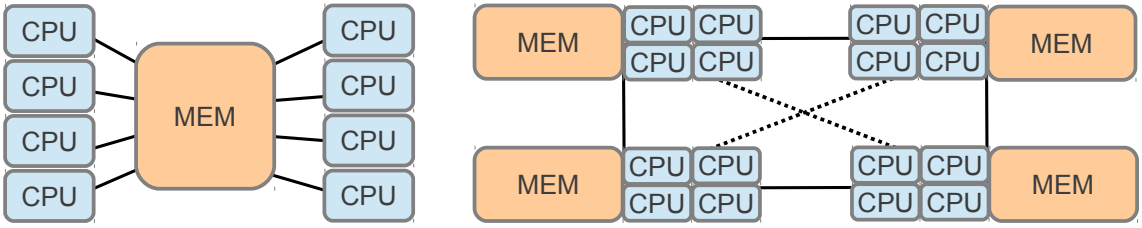
2. ábra: Az MIMD rendszerek Tanenbaum-féle felosztása. Forrás: [Morrison 03b, p. 19.]

A szorosan integrált rendszerekben – ahol a processzorok között saját busz-kapcsolatot építettek ki – a processzorok egyenrangúak (*SMP – Symmetric Multiprocessing*). A kommunikációt a központi osztott memórián keresztül hajtják végre (*UMA - Uniform Memory Access*), lásd 3.a. ábra. Az ilyen multiprocesszoros rendszerek esetén a globális memóriahasználat problémákat okozhat, hiszen előfordulhatnak olyan állapotok, amikor a processzorok ugyanazzal a változóval kívánnak műveletet végrehajtani, és az így keletkező memóriaverseny feloldását a hardvernek le kell tudni kezelni [Herlihy 08].

A *NUMA* (Non Uniform Memory Access) architektúrát használó gépek esetén az egyes processzorok (processzorcsoportok) saját memóriával rendelkeznek, de közösen használhatják a távoli processzorokhoz tartozó memóriát is (*lásd 3. b. ábra*) közvetett, vagy közvetlen eléréssel. Az ilyen rendszerű gépekkel a nagy memóriaigényű feladatok is megoldhatóak.

A lazán integrált rendszerek több, kis teljesítményű – jellemzően egy processzor egységet tartalmazó – számítógép hálózati összekapcsolásával állnak össze. Az ilyen rendszereket az elosztott kategóriába sorolhatjuk és alább a klaszterek alfejezetben fejtjük ki.

Újabb irányt képviselnek az egychipes multiprocesszorok azaz az egy processzortokba épített több processzormag (dual-core, multi-core) használata. A *„magok”* kifejlesztését a hagyományos processzorgyártási technológia indokolta, hiszen a félvezetők gyártásánál már a hagyományos miniatürizálási technikáknál fizikai korlátok merültek fel, így a Moore törvény várakozásainak megfelelő számítási teljesítményt már nem tudták elérni. A megoldást az egy tokon belüli többprocesszoros feldolgozás nyújtotta. A processzormagok gyors cache-eléréssel, nagy buszsebességgel növelhetik a kommunikációs sebességet és a teljesítményüket.



3. a,b. ábra: Közös memóriát használó (shared memory) rendszerek UMA (a) és NUMA (b) memóriamodellje

A szorosan integrált számítógéprendszerek azokon a területeken hatékonyak, ahol fontos a valósidejű, vagy nagyon gyors válaszidő. A jelenlegi processzor fejlesztési trendeket megfigyelve kijelenthetjük, hogy a hagyományos processzorok sorozatgyártását befejezik, és helyette már csak a többmagos processzorokat gyártanak a vezető processzorgyártók. Az ilyen hardvereken nagy szükség lesz a hatékony, elosztott és párhuzamos alkalmazásokra.

2.1.2. Klaszter rendszerek (cluster)

A lazán integrált rendszereket, melyben több kis teljesítményű PC összekapcsolásával lehet elérni nagyobb számítási teljesítményt, először a '90-es évek elején állította össze Sterling és Becker [BEO]. A Beowulf projektben szereplő 16 számítógép, az akkori viszonylatban gyorsnak számító hálózati kapcsolattal, a kutatólaboratóriumok számára a sokprocesszoros rendszerek árának töredékéért tette lehetővé a nagy teljesítményű, párhuzamos alkalmazások fejlesztését, kutatását. Az egyes számítógépek közötti adatforgalmat jellemzően üzenetküldéses modellel oldják meg. Az elérhető számítógépes és hálózati infrastruktúra fejlődésével a klaszterek teljesítménye is jelentősen megnőtt.

A klaszterek két fő csoportba sorolhatók:

- A *dedikált klaszterekben* speciálisan, egy feladat elosztott futtatására és megoldására

beállított számítógépek szerepelnek, melyek saját gyors és nagy teljesítményű hálózati kapcsolattal rendelkeznek. A dedikált klaszterek nagyobb egységekbe is szervezhetők (grid-rendszerek).

- A munkaállomások szabad kapacitásának kihasználására a gépek *ideiglenes klaszterekbe* szervezhetők [MSZGRID]. Ilyen lazán csatolt rendszert alkotnak a *NOW (Networks of Workstations)* és a *COW (Cluster of Workstations)* rendszerek. A lazán csatolt rendszerekben a futó algoritmusnak nagymértékben hibatűrőnek kell lennie, és számolni kell az eredmények jelentős késleltetésével is.

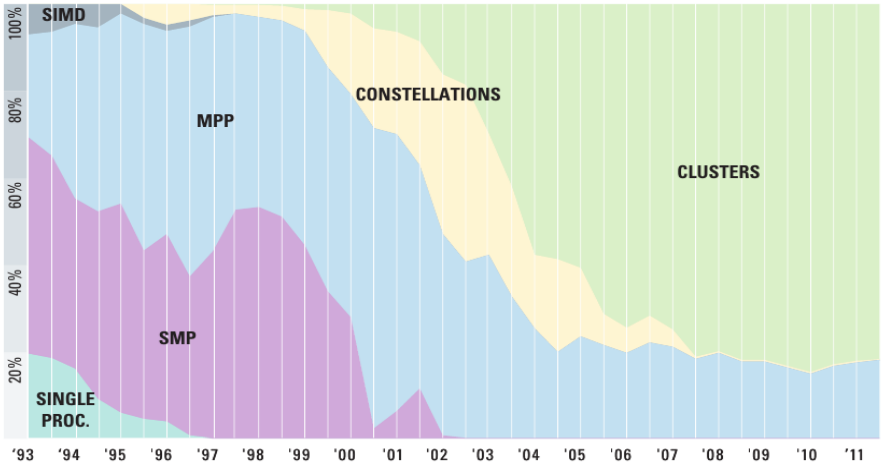
A klaszterekben az elosztott memóriakezelés során nem tapasztalhatók a konkurens írási és olvasási konfliktusok, míg a szorosan integrált rendszereknél gyakran előfordulhat, hogy valamelyik processzornak várnia kell a memóriaművelettel. Ezzel szemben a processzorok közötti kommunikáció a hálózati átvitel miatt jóval lassabb lesz. Így a klaszterekre írt párhuzamos algoritmusok akkor hatékonyak, ha a folyamatok közötti adatcsere kisebb mértékű.

A világ legnagyobb számítási kapacitásával rendelkező számítógépeinek ötszázas listája [Top500] alapján a grid-rendszereket a '90-es évek végéig még főleg a kutatóintézetek használták, mára azonban a fő felhasználói az ipari szektorból kerülnek ki. Ezeket a rendszereket elsősorban modellezésre, vagy szimulációra alkalmazzák (pl: csillagászatban, földtudományokban, meteorológiában, magfizikában, gazdasági modellekben, vegyészetben, agykutatásban, génkutatásban és bioinformatikában). Az ipari alkalmazások között elsősorban a járműtervezésben használnak metaszámítógépeket, ahol ma már mind a motor-, mind a formatervezéshez szükséges aerodinamikai modellezések, vagy az ütközés- és törésmodellezés elképzelhetetlen ilyen számítógépek nélkül [Kacsuk02].

A *4. ábrán* látatjuk az elmúlt húsz év legerősebb számítógépeinek architektúra és kiépítés szerinti csoportosítását. Az ábráról leolvasható, hogy a '90-es évek végére az egyprocesszoros és a vektorszámítógépek helyett a legjobb teljesítményt a többprocesszoros architektúrák biztosítják.

Megfigyelhetjük, hogy az ezredfordulóra a szorosan integrált, homogén kiépítésű rendszerek is háttérbe szorultak (SMP). Napjainkra a lazán csatolt rendszerek (MPP – masszív párhuzamos rendszerek) és a klaszterek szolgáltatják a legnagyobb teljesítményt [Strohmai-

er et al. 05]. (Megjegyzés: az ábrán a *„constellation”* kategória a klaszter-rendszerek egy speciális – kizárólag SMP gépekből épített – változata).



4. ábra: A világ legnagyobb kapacitású számítógépek típus szerinti csoportosítása évenkénti bontásban. Forrás: [Top500]

A 2012 júniusában kiadott statisztika szerint a világ jelenlegi legnagyobb metaszámítógépe a *„Sequoia”*, $16{,}32\,PFlop/s = 16{,}32 \cdot 10^{15}\,Flop/s$ maximális teljesítményével, 1 572 864 processzormaggal. Második helyre került a $10{,}5\,PFlop/s$ teljesítményre képes *„K computer”* 705 024 processzormaggal. A metaszámítógépek listáján túlnyomó többségben vannak japán, kínai és amerikai rendszerek. Az utóbbi évek fejlesztésinek köszönhetően a legjobb 10 közé meglepetésre három európai rendszer jutott (német, olasz, francia kutatóközpontokból) [Top500]. A Magyarországon működő klaszterek közül egyetlen egy sem került fel az aktuális ötszázas listára.

2.1.3. Grid-rendszerek

A technológia fejlődésével egyre bonyolultabb, korábban még megvalósíthatatlannak tűnő feladatok is megoldhatóvá válhatnak. Viszont az egyre pontosabb tudományos vagy műszaki modellek egyre nagyobb számítási sebességet igényelnek [Berman et al. 03]. Az igények terén ma már ott tartunk, hogy a legnagyobb, több *TeraFlop/s* sebességű szuperszámítógépek sem képesek egyénileg kielégíteni a felmerült számítási feladatokat. (Ilyen esetre egy példa a japán kezdeményezésre indult *EarthSimulator* projekt [EARTH]).

A probléma megoldása érdekében '90-es évek közepe óta a kutatók a multiprocesszoros és multikomputeres rendszerekből összekötött grid-rendszerek létrehozásán dolgoznak. 2000-től világszerte egyre nagyobb anyagi forrásokat koncentrálnak erre a területre. Az első két ilyen rendszer a FANFER *(Factoring via Network-Enabled Recursion* – az RSA kulcsok feltörésére indított prímszám-faktorizációs projekt) és az I-WAY (*Information Wide Area Year* – az Egyesült Államok kutatóintézeteiben található rendszereket összekapcsoló számítási hálózat) [IWAY]. A két projekt nagymértékben meghatározta a további fejlődési irányokat. Az eredményként előállt „*GLOBUS toolkit*" a további grid-rendszerek fejlesztésének alapjául szolgált. A grid-kutatások lényege annak vizsgálata, hogy hogyan lehet nagy távolságú, hálózattal összekötött, egymástól lényegesen különböző szuperszámítógépeket és/vagy klasztereket úgy használni, mintha egyetlen „nagyméretű szuperszámítógép" (metacomputer) alkotórészei lennének. Az eredeti cél az volt, hogy a metaszámítógépen úgy lehessen párhuzamos programokat futtatni, mint egyetlen szuperszámítógép, vagy klaszter processzorai között. A fizikailag is távol levő metaszámítógépek további összekapcsolásával (összetett grid-rendszerek) alkothatók olyan számítási hálózatok, amelyek egyetlen számítási egységként használhatók. A XXI. század jelentős feladatait már ilyen, kontinenseken átívelő grid-rendszereken lehet majd megoldani (pl. globális kereső és adatbányász rendszerek). Ilyen rendszer építése, és üzemeltetése azonban igen jelentős költséget ró az adott országra, vagy ipari szereplőre (pl. egy metaszámítógép éves áramfogyasztása egy kisebb városéval egyezik meg).

Az akadémiai kutatásokban nem elhanyagolható az önkéntesen felajánlott számítógépes erőforrások által végzett számítási rendszerek használata. Az ilyen lazán, illetve ideiglenesen kapcsolódó, bizalmi alapon működő rendszerekben a résztvevők a világhálón keresztül kapcsolódnak valamilyen grid-rendszerhez, amelytől véletlenszerűen feladatokat kapnak. Ezen feladatokat a számítógépük szabad processzoridejében végrehajtják, majd az eredményt a grid központjának visszaküldik. A BOINC rendszert használó grid-rendszerekben jelenleg kb. 17 PetaFlop/s az összes elérhető felajánlott számítási teljesítmény [BOINCST]. Az ilyen önkéntes jellegű grid-rendszerek olyan kutatásokban sikeresek, amelyek népszerűek, vagy társadalmilag elfogadható célokat valósítanak meg. A legelső ilyen modell a *SETI@home* például idegen civilizációk keresésére indult. Ezután indultak el a rák- vagy gyógyszerkutatások (*rosetta@home, poem@home*), a klímakutatás (*climaprediction.net*), vagy

valamilyen természettudományos célt szolgáló elosztott grid projektek (*einstein@home, primegrid*). A kutatóintézetek sikeresen használják az önkéntes számítási hálózatokat, a projektek száma folyamatosan bővül.

2.1.4. CPU-GPU hibrid rendszerek

A grafikus gyorsítókártyák megjelenésének elsődleges célja a 3D grafika megjelenítése és a CPU egységek tehermentesítése volt. A GPU egységek egyéb feladatokra való felhasználási lehetőségeit már a 2000-es évek elején felismerték [Thompson 02]. Az így elérhető számítási többblet-teljesítmény (és a GPU egységek relatív olcsósága) napjainkra kikerülhetetlen tényezővé tette a felhasználásukat a párhuzamos és elosztott rendszerekben (Pl: a kínai *„Tianhe-1A"* szuperszámítógép 14336 db Xeon X5670 processzorral, 7168 Nvidia Tesla M2050 GPU-val: 2,54 PetaFlop/s teljesítményű [Top500]). Mivel a GPU egységeket elsősorban képi gyorsításra tervezték, ezért ezek általános felhasználása, és a CPU egységgel való szoros együttműködése csak bizonyos korlátozásokkal érhető el. Összességében elmondható, hogy a GPU egységek használatával jelentős számítási teljesítmény és gyorsulás érhető el [Khanna 10][Preis 99][Tomov et al. 10].

2.1.5. A szorosan és a lazán integrált rendszerek összehasonlítása

A lazán, vagy szorosan integrált többprocesszoros rendszerek összehasonlításakor több szempont szerint is el lehet végezni a vizsgálatokat.

- Mind a többprocesszoros, mind a klaszter vagy grid-rendszerek használatánál elengedhetetlen a használt algoritmusok nagyfokú párhuzamosítása.
- A többprocesszoros számítógépek esetén a processzorok közötti üzenetküldés gyors, hiszen az a belső buszt, vagy kapcsolórendszert használja. A többmagos rendszerekben az üzenetküldés még gyorsabb a belső cache-mechanizmus használatával. Ezzel szemben az osztott memóriahasználatból adódó konfliktushelyzeteket az operációs rendszernek és a futtató környezetnek le kell tudnia kezelni.
- A klaszterekben a szűk keresztmetszetet a hálózati kapcsolatok okozzák. Az üzenetküldéses rendszereket fel kell készíteni az üzenetek késleltetésének, vagy kimaradásának kezelésére.

- A klaszter-rendszerek hibatűrőbbek lehetnek, mint a multiprocesszoros rendszerek, hiszen ha egy csomópont kiesik, attól a rendszer még működőképes marad, míg egy multiprocesszoros gépnél egy processzor, vagy a központi memória hibája az egész rendszert megbéníthatja.
- A klaszter/grid rendszerek ár/teljesítmény aránya jobb, mint a multiprocesszoros gépeké.
- A decentralizált rendszerek menedzselése több munkát kíván.
- A klaszter és grid-rendszerekre kevés újrafelhasználható szoftver-komponens áll rendelkezésre.
- Mindkét technológia transzparens, azaz a felhasználó egyetlen számítógépként látja a háttérben álló többprocesszoros rendszereket.
- A klaszter és grid-rendszerek jól skálázhatók, egy adott feladathoz további számítógépek, vagy alrendszerek vonhatóak be. A multiprocesszoros rendszerek bővítése bonyolultabb és költségesebb.

Összességében elmondható, hogy a jövő nagy számításigényű feladatait az elosztott és/vagy grid-rendszerek fogják megoldani, melyekben számos multiprocesszoros gép vagy klaszter is részt vehet [Riedel et. al 09]. A kisebb méretű ipari feladatokhoz elegendő lehet egy kisebb méretű klaszter, vagy egy multiprocesszoros számítógép is.

2.2. Egyenrangú és kliens-szerver rendszerek

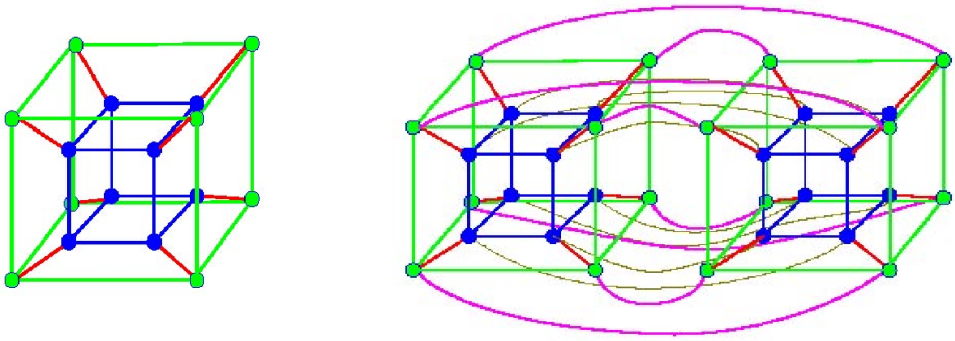
A több processzort használó rendszerek összekapcsolása a fentiek alapján számos módon lehetséges. A felhasznált processzorok egy általános leírására használhatjuk a *homogén* (azonos technológiát használó processzorok, azonos kommunikációs hálózattal, azonos rendelkezésre álló memóriával), és a *heterogén* kategóriákat (különböző technológiájú processzorok, különböző kommunikációs csatolókkal, különböző memóriával).

Ezeket a fogalmakat Tanenbaum eredetileg csak a multiszámítógépekre értelmezte [Tanenbaum 04], de ezen fogalmak napjainkra kiterjeszthetők minden összetett rendszerre. Mára számos gyártó kínál összetett, hibrid architektúrával szerelt „homogén” gépeket, illetve klaszter is építhető azonos tulajdonságú komponensekből (*konstellációk*).

A klasszikus párhuzamos algoritmusok tervezésekor (*lásd 3.1. fejezet*) csak homogén rendszereket vettek alapul. A processzorszám növelésével azonban a tisztán homogén hardver megalkotása bonyolult feladat, ezért mára a tiszta homogén megoldások ritkák. Egyre gyakoribbak a moduláris felépítésű, bővíthető multiszámítógépek. A heterogén rendszerekre általában a kliens-szerver jellegű szoftvermegoldások a jellemzőek, ahol az egyes számítást végző csomópontok más-más feladatot kapnak. A processzorok egymáshoz kapcsolásának leírását a hálózatkezelésben használatos topológia fogalommal írjuk le, függetlenül attól, hogy a többprocesszoros rendszer szorosan- vagy lazán csatolt.

2.2.1. Hálózati topológiák

A több processzort használó rendszerek esetén a számítási feladatok hatékony megoldása érdekében lényeges a processzorok közötti összeköttetések struktúrájának megtervezése. Az összeköttetések leírására leggyakrabban a hálózatoknál ismert topológia fogalom terjedt el. Így a többprocesszoros rendszereknél egy csomópont egy processzort (vagy számítógépet), egy él pedig két csomópont közötti fizikai összeköttetést jelent.

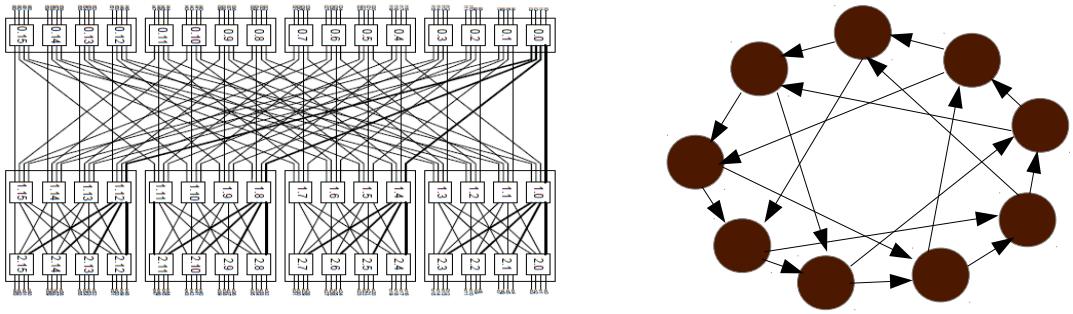


5. ábra: Hiperkocka topológia 4 és 5 dimenziós esetben. Forrás: [Kallós 04]

Kezdetben az SIMD gépek felépítéséhez az egyszerűbb gyűrű, rács, busz topológiákat használták. Az összetettebb architektúrákban gyakori a hiperkocka és a pillangó jellegű topológia [Hord 93] (*lásd: 5., 6 a. ábra*). Ezen topológiák megalkotásakor minden egyes processzor azonos típusú és feladatában egyenrangú volt.

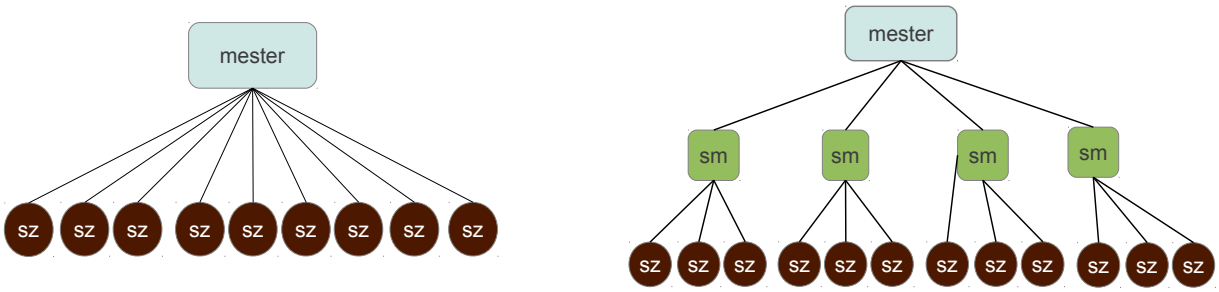
Az MIMD gépek megjelenésével a topológia fogalma továbbra is érvényben maradt, hiszen a multiprocesszoros rendszerekben jellemzően azonos processzorok szerepeltek. A multiszámítógép-rendszerek megjelenésével a topológia ábrázolása megmaradt, azzal a különbséggel, hogy a csomópontok szintén egy-egy processzort jelentenek, de a gráf élei

már nem csak a szorosan kapcsolt huzalozást, hanem valamilyen hálózati kommunikációs vonalat is reprezentálhatnak. Így az egyes sémák alkalmasak a heterogén rendszerek leírására is.



6. a,b. ábra: Pillangó és gyűrű elrendezésű hálózat

A gyűrű topológia (*lásd: 6 b. ábra*) egy speciális esete a teljes gráffal leírható topológia, de ezt az elrendezést csak kisebb processzorszám esetén célszerű használni . A processzorszám növelésével a kommunikációs csatolások száma nő, és a későbbiekben részletesen kifejtett kommunikációs torlódásokat, vagy végső esetben számítási instabilitást is okozhatnak.



7. ábra: Egy- és kétszintű hierarchiát megvalósító hálózat

A heterogén többprocesszoros rendszerekben gyakran alkalmazott a hierarchikus szervezésű modell (*7. ábra*). Ebben az esetben a topológia egy egy gyökerű fa struktúrával ábrázolható. A gyökérpontban jelen levő processzor feladata elkülönül a többiétől, jelölésére jellemzően a mester-szolga (más terminológiában szülő-gyermek) szerepeket használunk.

Amennyiben a számítást végző processzorok száma nagy (például klaszter, vagy grid rendszerekben) és a feladat jól darabolható, gyakori a két-, vagy többszintű hierarchia megadása. Ezekben az esetekben a hierarchia minden egyes szintje más-más feladatot lát el.

A párhuzamos és elosztott számításokban gyakran a struktúra szervezése vagy a rendelkezésre álló hardver sajátja, vagy logikai kapcsolatokkal felépíthető. Mivel az egyes csomópontokban hagyományos Neumann-elven működő egységek helyezkednek el, ezért azt várnánk, hogy bármelyik topológia hasonló számítási eredményeket szolgáltat.

A *8. fejezetben* megmutatjuk, hogy ez sokkal összetettebb probléma. A feladatok modellezésekor a megoldandó feladathoz legjobban igazodó struktúrát érdemes felhasználni, ennek megtalálása azonban nem triviális feladat.

2.2.2. Kommunikáció és szinkronizálás

Mind a szorosan-, illetve a lazán csatolt rendszerekben a processzorok közötti összeköttetések esetében kulcsfontosságú a kommunikáció sebessége. Az egyes feldolgozó egységek közötti késleltetés a kiépített rendszer fizikai adottságaiból, illetve az alkalmazott kommunikációs protokoll szerkezetéből eredhet. Az adatok és üzenetek továbbításakor figyelembe kell venni, hogy az egyes processzorok különböző órajel frekvencián üzemelnek, így egy összetett rendszerben olyan alacsony szintű kommunikációs és szinkronizációs protokollokra van szükség, amelyek megteremtik a lehetőségét a többprocesszoros, vagy elosztott rendszerek hatékony működésének [Tanenbaum 04]. Az MIMD rendszerekben a közös memóriát használó architektúrákban nagyságrendekkel kisebb ez a késleltetési érték, mint az elosztott memóriát használóknál (késleltetési nagyságrend: ~ns a „shared memory" modellben illetve ~ms a „distributed memory" esetén)[Környei et al. 10].

2.3. A sokprocesszoros rendszerek mai helyzete Magyarországon

A magyarországi elosztott kutatási projektek már a '90-es évek elején megkezdődtek. A VISzSzKI (Virtuális Szuperszámítógép Szolgáltatás Kialakítása) projektet követően a *DemoGrid*, *SzuperGrid*, *JGrid*, *Chemistry Grid, ClusterGrid, HungarianGrid, MEGA* projektek mindmind az elosztott nagy teljesítményű számításokat nyújtó rendszerek kifejlesztésén dolgoznak [Szeberényi 06]. A hazai kutatók számos európai projektben is részt vesznek, így a CERN (*European Organization for Nuclear Research*) és az EGEE (*Enabling Grids for E-science*) EU-támogatott keretprogramjában futó projektjében [EUDG]. A 2000-es évektől kezdve számos magyarországi egyetemet és kutatóintézetet felszereltek közepes méretű multiszámítógépekkel (NIIFI, BME, Szeged, Debrecen, Pécs, Győr), amelyek önállóan, vagy együttműködve képesek nagy számításigényű párhuzamosítható feladatokat végrehajtani [MSZGRID].

3. PÁRHUZAMOS ALGORITMUSOK

A több processzort használó számítógépeken, vagy számítógép-rendszerekben különleges szerepet kapnak a feladatok megoldását leíró párhuzamos algoritmusok. A klasszikus Turing és RAM (random access machine – véletlen hozzáférésű gép) gépmodellek kiterjesztéseként Fortune és Wyllie 1978-ban bevezette a PRAM modellt (parallel random-access machine – párhuzamos véletlen hozzáférésű gép), amelyben egy elméleti többprocesszoros rendszer jellemezhető [Cormen et al. 01]. A PRAM modellben a processzorok szinkronizáltan, egyetlen, végtelen méretű közös memórián osztoznak, amelyet egységnyi idő alatt érnek el. A szakirodalomban fellelhető párhuzamos algoritmusok jellemzően ezen PRAM modellen alapulnak [Van de Velde 94]. A PRAM modell óriási hátránya, hogy a gyakorlatban használt és folyamatosan változó hardver jellegzetességeket az algoritmus tervezéskor figyelmen kívül hagyja.

3.1. Párhuzamos algoritmusok tervezése és elemzése

A klasszikus PRAM modellben a párhuzamos algoritmusokat a műveletek száma, pontosabban a memória-hozzáférések száma jellemzi [Iványi 03]. Ezt annak a megfigyelésnek a kiterjesztése adta, hogy a klasszikus egyprocesszoros rendszerben a memória-hozzáférések száma aszimptotikusan közelíti a futási időt [Kallós 04] (RAM modell). Ezen alapfeltevésen kívül figyelembe kell venni azt is, hogy a felhasznált processzorok számától függhet a végrehajtás ideje. Így a vizsgálat során a PRAM modellben a műveletszám mellett a processzorszámot is vizsgálni kell, hiszen már a modell megalkotásakor nyilvánvaló volt, hogy a párhuzamos algoritmusok hatékonysága ezen két paramétertől függ, közöttük szoros összefüggés figyelhető meg. A több processzort és párhuzamos algoritmust használó rendszerekben a processzorok és a memória kapcsolatában az alábbi csoportosítást használhatjuk:

- EREW (Exclusive Read – Exclusive Write): kizárólagos olvasás és írás. A memóriából egy időben csak egy processzor olvashat, oda csak egy processzor írhat.
- CRCW (Concurrent Read – Concurrent Write): egyidejű olvasás és írás. Több processzor olvashat és írhat a memóriába egyidőben.

- CREW (Concurrent Read – Exclusive Write): egyidejű, konkurens olvasás, kizárólagos írás.
- ERCW (Exclusive Read – Concurrent Write): kizárólagos olvasás, egyidejű írás.

A PRAM modell használatával a fenti hozzáférési kategóriák közül az EREW és a CRCW modell terjedt el. Az EREW modell gyakorlatilag a hardverre bízza a teljes ütemezést és vezérlést, és nem foglalkozik az ütközésekkel, míg a CRCW modell képes leírni a valódi nem-determinisztikus gépeken jelentkező eseteket is.

Itt azonban egy kisebb logikai ellentmondást találhatunk a PRAM modellben: a modell definíciójában minden memóriaművelet egységnyi, ezzel szemben a konkurens író/olvasó rutinokban egységnyi idő alatt kell több processzornak műveleteket végrehajtania. Így vagy az író művelet nem lehet egyértelmű, vagy valamilyen további szabályozásra, a modell kiegészítésére van szükség [Cormen et al. 01].

A CRCW modell az alábbi modellek szerint értelmezhető:

- *véletlenszerű CRCW* modell: amennyiben több processzor egyszerre ír egy adott területre, akkor a modell véletlenszerűen választja ki, hogy melyik processzortól jövő adatot tárolja.
- *közös CRCW*: több processzor írása esetén csak konszenzus esetén (mindegyik ugyanazt az értéket írná) történik meg a tényleges írás.
- *prioritásos CRCW*: a legkisebb sorszámú, (vagy legerősebb prioritású) processzor által számított adatot tárolja.
- *kombinált CRCW*: a processzorok által szállított értékek valamilyen függvény szerint meghatározott értéke kerül a memóriába (pl: összeg, vagy maximum).

A PRAM modellben a processzorok szigorúan szinkronizáltak, belső órájuk pontos, és nem fordulhatnak benne elő idő-anomáliák. *Brent-tétele* kimondja, hogy kombinációs áramkörökkel épített gépek szimulálhatók a PRAM modellel:

> *„Bármely d mélységű, n méretű kombinációs áramkör O(n/p+d) idő alatt szimulálható p processzoros CREW algoritmussal."* [Cormen et al. 01, p.613.]

Az algoritmusok gyakorlati tervezésénél azonban nem hagyhatjuk figyelmen kívül azt a tényt, hogy a valóságban a PRAM modell szerinti gépet még homogén rendszerekben is

nagyon nehéz előállítani. A memória elérést nem tekinthetjük egységnyi idejűnek, illetve a napjaink hardver megvalósításai már sokszor tartalmaznak alacsony szintű gyorsítási vagy párhuzamos komponenseket, amelytől azok már nem tekinthetők szigorúan homogénnek. A heterogén rendszerekben pedig a fizikai korlátok és „távolság" miatt ez a modell nagyon gyakran kerül holtponti, vagy várakozó állapotba. Összességében azért elmondható, hogy a PRAM modell, mint általános párhuzamos programozási modell kiindulásként jól használható.

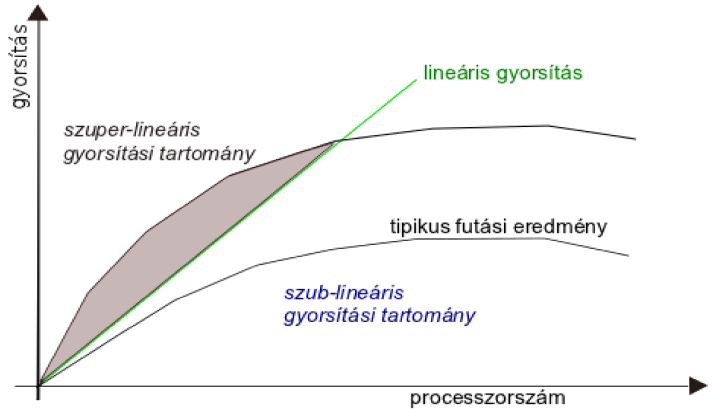
Elosztott heterogén rendszereknél a gyakorlatban a PRAM modell már csak korlátozásokkal, illetve a kommunikációs modellekkel, vagy köztes-réteg megoldásokkal kiegészítve használható. A PRAM modell hiányosságai kiegészítésére megalkották a *BSP* (Bulk-Synchronous Parallel model – szinkronizált párhuzamos modell), a *LogP* (Latency Overhead Gap Processors – ütemezett, késleltetett párhuzamos modell) és a *QSM* (Queuing Shared Memory – szinkronizált közös memóriaelérésű modell) párhuzamos algoritmusokat leíró modelleket [Iványi 05a] [Ramachandrana 03], de ezek kevésbé elterjedtek és szűkebb területeket fednek le, mint a PRAM modell. A BSP modellben az ún. *szuperlépésekkel* a folyamatok folyamatos kommunikációja és szinkronizációja is megtörténik. A QSM közös memóriájú gépekhez használható, míg a LogP a BSP modell egy továbbfejlesztett, kiegészített változata.

3.2. Párhuzamos programok teljesítményének mérése

A párhuzamos algoritmusok teljesítményének mérése meglehetősen összetett feladat, sajnos a több processzor összekapcsolása nem jelenti a számítási teljesítmény automatikus sokszorozódását. A gyakorlatban a szekvenciális programokhoz képesti *gyorsítási értékkel (speedup)*, vagy a *skálázhatósággal (scaling)* számolhatunk. A gyorsítás a párhuzamos és a szekvenciális végrehajtási idő hányadosa (*N* processzor esetén): $S_N = \frac{T_1}{T_N}$, amely ideális esetben elérheti a maximális *N* értéket, a lineáris gyorsítást. Amennyiben a gyorsítás értéke *N* feletti, akkor szuper-lineáris gyorsításról beszélünk (*lásd: 8. ábra*). Ez elérhető a párhuzamos algoritmus valamilyen speciális jellegével, vagy hardveres gyorsítótárak használatával.

Skálázódásról akkor beszélünk, ha a szekvenciális algoritmus ideje nem áll rendelkezésünkre, csak a különböző processzorszámokhoz tartozó végrehajtási idő.

A párhuzamos algoritmusok *hatékonysága* azt jellemzi, hogy a párhuzamos algoritmus mennyire használja ki a rendelkezésre álló környezetet, a számításra fordított idő milyen arányban áll a kommunikációs és szinkronizációs veszteségekkel: $E_P = \frac{S_N}{N} = \frac{T_1}{N \cdot T_N}$.



8. ábra: Gyorsítási értékek alakulása párhuzamos rendszerekben [Iványi 05a] alapján

A párhuzamos alkalmazások gyorsítására és a hatékonyságának mérésére az első alaptételt Amdhal mondta ki, amelyben az *N* processzoron elérhető maximális gyorsítási érték – az algoritmus párhuzamosítható *(P)* és a nem párhuzamosítható részeitől *(1-P)* függenek – az alábbi képlet szerint számítható [Amdahl 67]:

$$S_N = \frac{1}{(1-P) + \frac{P}{N}} \tag{1}$$

Az Amdahl tételt Gustafson módosította a '80-as években úgy, hogy előrevetítette, hogy a párhuzamosan megoldandó feladatok mérete folyamatosan nő együtt a hardverrel, így azonos idő alatt egyre nagyobb feladatok lesznek megoldhatóak [Gustafson 88]:

$$S(N) = N - \alpha(1-N), \tag{2}$$

ahol *N* a processzorok száma, α a feladat nem párhuzamosítható részeinek aránya.

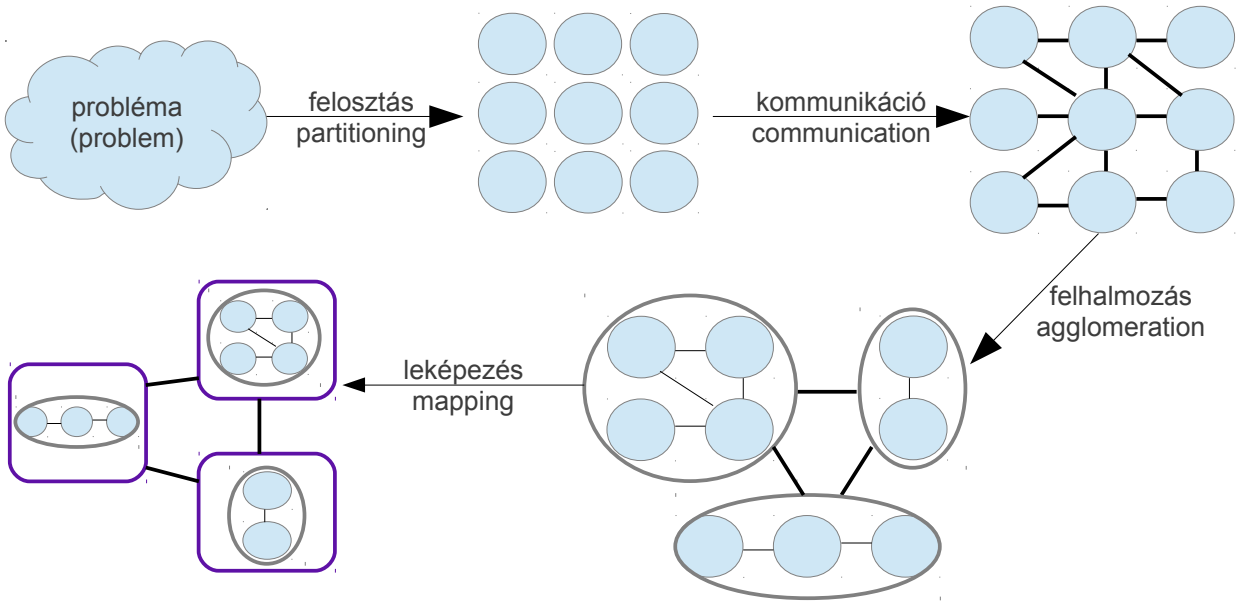
A fent említett PRAM modell nem veszi figyelembe azokat a jellegzetességeket, amelyek a mai processzorokat jellemzik. Így a feladatütemezés, taszkváltás, a különböző memória-hozzáférések ideje, a kommunikáció és a szinkronizációs folyamatok mind-mind időveszteséget okoznak, és rontják az összetett rendszerek hatékonyságát. Így a gyakorlatban, illetve az Amdahl-Gustafson törvények figyelembe vételével a processzorok számának emelése csak

egy bizonyos határig javíthatja a hatékonyságot. Ezért szükséges a hagyományos szekvenciális algoritmusok adott párhuzamos gépekhez alakítása és ezen algoritmusok viselkedésének mélyreható elemzése.

3.3. Párhuzamos algoritmusok jellemzői többprocesszoros, illetve elosztott rendszerekben

Az elosztott jellegű rendszerekben (klaszter, vagy grid-jellegű számítások) a tisztán elméleti PRAM modell helyett a folyamatok közti kommunikáción alapuló, az adott hardver kiépítést figyelembe vevő, vagy köztes-réteg megoldásokat alkalmaznak. Az elosztott rendszerek hardver- és szoftver megoldásait alaposan elemzi és mutatja be [Crichlow 03] [Tanenbaum 04] [Lynch 02] [Iványi 05b] és [Iványi 07].

A PRAM modellben használt egységnyi idejű memória-hozzáférés és a tökéletesen szinkronizált utasítás végrehajtás valós rendszerekben nem teljesül. Külön kezelendő problémát okoz az adatok memóriában tartása, továbbítása és a hozzáférések kezelése.



9. ábra: Párhuzamos algoritmusok Foster-féle tervezési lépései

Az elosztott rendszerekben elkészített algoritmusok esetén az egyes processzorok gyakran nem ismerik a teljes megoldandó „globális” problémát, csak részinformációkkal rendelkeznek. Ebben a szemléletben nagy hangsúlyt kapnak az üzenet továbbító, és szinkronizáló protokollok, illetve a hibakezelés. A párhuzamos algoritmusok a fenti elvek mentén történő tervezésének főbb lépéseit Foster fogalmazta meg (*lásd 9. ábra*): [Foster 95]

- *A probléma felosztása (partioning):* a tervezés kezdetén a feladat – a lehető legtöbb – egymástól független részre osztása. Ez lehet adat szintű (*data decomposition*), illetve a számítási tevékenység működési szintű felosztás (*functional decomposition*).
- *Kommunikáció (communication):* a darabokra bontott részfeladatok közötti kommunikációs igények megtervezése lokális-, és globális szinten. A kommunikáció szervezése lehet szinkron, vagy aszinkron.
- *Felhalmozás (agglomeration):* a részletes feladatlisták nagyobb csoportokba szervezése, melyekben több lokális szintű adat, és kevesebb kommunikációs költség szerepel. Ez a lépés elősegíti a hatékony terhelés-elosztást (*load balancing*). A feladatcsoportok számát a futtató hardver igényeihez kell igazítani (például a felosztást a rendelkezésre álló processzorszám felülről korlátozza).
- *Leképezés (mapping):* a feladatok processzorokhoz rendelése. A hozzárendelés módja függ a dekompozíció módjától, illetve szervezése lehet blokkos, ciklikus, vagy véletlenszerű.

A *Foster-féle* tervezési modellben a problémafelbontás finom és összetettebb szintjei, a kommunikációs csatornák tervezése, a költségek becslése – különös tekintettel a szűk keresztmetszetekre (*hotspot - bottleneck*) –, illetve az egyenletes terhelés elosztás (*load balancing*) célja minden esetben a futási idők minimalizálása és a hatékonyság növelése.

3.4. Elosztott környezetben használt szoftver-komponensek

Az elosztott alkalmazások fejlesztéséhez alkalmas szoftverek és szoftverrendszerek fejlesztésének igénye már az első multiprocesszoros gépek megjelenésekor megfogalmazódott. A párhuzamos algoritmusok hatékony futtatásához a klasszikus elosztott memóriás hozzáférésű szoftverrendszerek születtek. Ezek az adott hardverre készült saját (általában UNIX alapú) operációs rendszerrel rendelkeztek. A multitaszkos operációs rendszerekben már korán megjelent a folyamat-alapú párhuzamosíthatóság [Lynch 02].

Az osztott memóriás rendszerekben az SMP (*Symmetric MultiProcessing*) rendszerű alkalmazások állnak, melynek közös szabványává az OpenMP vált. Az OpenMP-t a főbb hardvergyártók ipari szabványként elfogadták. A standard leírja a főbb környezeti változókat,

fordítási direktívákat és függvénykönyvtárakat. Két támogatott programozási környezete a Fortran és a C/C++ [OPENMP].

Az egyes folyamatok (processzek, vagy taszkok) azonban az *üzenetváltásos paradigma* szerint is kommunikálhatnak. Az üzenetváltás lényege, hogy a processzek közti kommunikációban küldik át az összes szükséges adatot. Ez elsősorban az olyan rendszereknél használható, ahol közös memória nem áll rendelkezésre. Az első üzenetváltást használó szoftveralkalmazást PVM néven (*parallel virtual machine*) 1989-ben alkották meg, és ezt *de facto* szabványként évekig használták.

A PVM heterogén hardver- és szoftverkörnyezettel rendelkező gépekből egyetlen virtuális gépet épít fel, ahol a felhasználó az algoritmusokat ezen a virtuális gépen futtatja. A PVM segítségével egy UNIX processz hozzákapcsolódhat ezen virtuális géphez, illetve további folyamatokat is tud indítani. A független folyamatok képesek egymás közötti kommunikációra. A PVM rendszer implicit, vagy explicit módon a feladatokat az egyes PVM-be bevont processzoroknak kiosztja és párhuzamosan futtatja.

Az MPI (*Message Passing Interface*) egy olyan függvénykönyvtár-gyűjtemény, amely szintén az üzenetküldés paradigmát használja. Az MPIForum 1994-ben adta ki az MPI 1.0-s szabványát, majd 1997-ben az MPI 2.0-t. Az MPI megadja azokat a széles körben használt kommunikációs módszereket, melyek segítségével hatékony párhuzamos kódok írhatók. Az MPI a PVM rendszertől eltérően támogatja a szinkron – aszinkron kommunikációt, az üzenetszórásos, illetve pont-pont kommunikációt. A függvény-könyvtárakban definiált műveletek magas szintű, a heterogén rendszereken is futtathatók, könnyűszerrel hordozható kódot biztosítanak. Az MPI szabvány a Fortran, a C és C++ nyelveket támogatja [Snir et al. 95] [Dongarra 96].

A hibrid jellegű GPGPU rendszerekben a hardver architektúrához illeszkedően önálló programozási környezetet alakítottak ki a gyártók. A két legelterjedtebb módszer a hardverközeli *CUDA*, illetve az *OpenCl* programozási környezet [CUDA] [OPENCL].

A klaszterek és a grid-rendszerek üzemeltetése és menedzselése összetett mérnöki folyamat, ezért a leggyakoribb funkciókhoz (*felhasználókezelés, futtatás és feladatütemezés*) kifejlesztettek több célszoftvert is:

- *Globus:* egy olyan többrétegű szolgáltatáshalmaz, amely a fejlesztők számára nagyfokú rugalmasságot biztosít. A szolgáltatásrétegek egymásra épülnek, és jól definiált interfészeket biztosítanak. A főbb szolgáltatások a következők: *biztonság, erőforrás-keresés, erőforrás-kezelés, adatelérés, kommunikáció.*
- *Condor:* egy olyan feladatütemező rendszer, amelyet a hosszú futási idejű, nem feltétlenül dedikált klasztereken futó alkalmazások menedzselésére fejlesztettek. Elsősorban a szabad processzor kapacitásokat kihasználó feladatütemezőre épül, és hatékonyan szolgálja ki a klasztereket. A feladatkezelést hatékony ütemező, házirend és erőforrás-kezelő komponenssel biztosítják. A feladatokba ágyazott ellenőrző pontokkal az egyes, hosszabb lefutású folyamatok futási állapotait lejegyzi, így egy feladat újraindítása után (ha az adott processzor leállt, újraindult, vagy a feladatot másik processzornak kell kiosztani) a már elvégzett számításokat nem szükségszerű újrafuttatni.
- *P-GRADE* (*Parallel Grid Run-time and Application Development Environment):* a magyarországi szuperszámítógép program keretei között kifejlesztett alkalmazásfejlesztő rendszer [PGRADE], amely a Condor, majd Globus rendszerrel együttműködve hatékonyan képes kezelni a PVM és MPI alkalmazásokat nagyobb klasztereken. Az alkalmazásfejlesztést magas szintű grafikus eszköztámogatással, hibakeresési, nyomkövetési és teljesítmény monitorozási funkciókkal támogatja [Kacsuk 06].
- *PBS (Portable Batch System)*: egy multiszámítógépes környezetben általánosan használható, elosztott feladatütemező rendszer. Legfőbb feladatai a munkafolyamat kezelés, ütemezés és a monitorozás [PBS]. A PBS számos összetett rendszerben (pl. Globus), mint alrendszer megtalálható.

4. A NUMERIKUS MÓDSZEREK ÉRINTETT TÉMAKÖREI

A numerikus matematika feladata olyan – a tudományos kutatás és az ipari felhasználás gyakorlatában gyakori – problémamodellek megadása, amelyek a digitális számítógépek segítségével hatékonyan oldhatók meg. Ebben a fejezetben röviden összefoglaljuk és elemezzük azokat a témaköröket és fogalmakat, amelyeket a kutatómunka során, illetve a tézisek megfogalmazásánál a későbbiekben felhasználunk.

4.1. Számítási hibák, és a hibák tovaterjedése

A Neumann-elvű számítógépeken használt lebegőpontos aritmetikában a valós számok ábrázolása nagyon korlátozott, így már egy szám gépi ábrázolásakor előfordulnak *kerekítések*. Az ábrázolás során kiemelt szereppel bír a legkisebb ábrázolható pozitív szám: ε_1 az ún. gépi epszilon, amely meghatározza a számítást végző gép relatív pontosságát (a gépi epszilon meghatározását bővebben lásd: [Iványi 07 p. 722.]).

A számítógépes feladatmegoldásokban jelentkező hiba az *alul-* illetve *túlcsordulás* jelensége. A legtöbb gép az ábrázolási határok elérésekor legtöbbször hibajelzés nélkül „kerekít". Hasonló problémát jelent a *kiegyszerűsödés*[1] problémája [Henrici 85]. Az ilyen jellegű problémák sokszor nagyságrendekkel nagyobbak lehetnek az ε_1-nél.

A számítások során beszélhetünk *abszolút* $|\tilde{y}-y|$, illetve *relatív* $|(\tilde{y}-y)/y|$ hibáról, ahol y a pontos, és $\tilde{y}$ a számított eredmény [Stoyan 02]. A számítási hibák eredhetnek a kiinduló adatok pontatlanságából, vagy a számítás során elvégzett műveletsorozat során felhalmozódó hibákból. A hibák elemzésére használhatunk *direkt* (a kiindulási adatokból következtetve), illetve *indirekt* hibaanalízist (a végeredménynél előálló hibákból következtetünk a kiinduló adatok hibájára) [Galántai-Jeney 05].

A numerikus műveletek *numerikus stabilitásáról* akkor beszélhetünk, ha a számítás során a megoldás, vagy közelítő megoldás hibája nem romlik, azaz algoritmus nem érzékeny a kerekítési hibákra. [Blum et al. 98]. Nagy műveletigényű feladatok esetén a hibák felhalmo-

1 A kiegyszerűsödés egy példája: amennyiben egy *a* és *b* szám lebegőpontos ábrázolásakor közös *z* karakterisztika értéket vesz fel, akkor az *a-b* különbség képzésekor a mantisszában értékes jegyek veszhetnek el mind egyszeres, mind dupla pontosságú ábrázolás választásakor.

zódása *numerikus instabilitást* okoz, ekkor a számításokban a végeredmény nem minden esetben állítható elő kellő pontossággal.

4.2. Lineáris egyenletrendszerek

A lineáris egyenletrendszerek vizsgálata a numerikus matematika egyik alapfeladata, amelyek számos más tudományterületen is alkalmazhatóak (pl.: mérnöki tudományok, közgazdaságtan, föld- és anyagtudományok, adatbányászat, adatbázisok optimalizálása és a tudományos alapkutatások számos területén).

A numerikus módszerek más feladatait is gyakran vezethetjük vissza lineáris egyenletrendszerekre (lineáris egyenlőségrendszerek, differenciál- integrálegyenletek, interpoláció, optimalizáció) [Bahalov 77]. A lineáris egyenletrendszerek általános alakja így definiálható [Stoyan 02, 35.p.]:

$$A \cdot x = b,\ A = a_{i,j} \in \mathbb{R}^{nxm},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^n \tag{3}$$

A lineáris egyenletrendszereket a *kondíciószámmal*, a megoldások hibáját a *norma* értékével jellemezhetjük. Az A mátrix kondíciószáma jól jellemzi a feladat megoldhatóságát $cond(A) = \|A\| \cdot \|A^{-1}\|$. A rosszul kondicionált mátrixokkal megadott egyenletrendszerek megoldása hagyományos módszerekkel csak nagy relatív hibával lehetséges.

A lineáris egyenletrendszerek két fő megoldási lehetősége a kiküszöbölési eljárások mentén megoldó (direkt), illetve a fokozatos közelítés (iteratív) módszere.

4.2.1. Direkt és iteratív megoldó algoritmusok

Direkt módszerek

A lineáris egyenletrendszerek hagyományos direkt megoldó módszere a Gauss-féle kiküszöbölési eljárás. Segítségével a megoldásvektor, a mátrix rangja és determinánsa meghatározható. A hagyományos Gauss elimináció csak akkor végezhető el, ha a főminor értékek nem zérusak $a_{i,i} \neq 0$. (Különben a részleges főelemkiválasztás módszere alkalmazható: sorok cseréje a legnagyobb abszolút értékű együtthatójúra). Az eliminációs lépések után az utolsó egyenlet megoldása egyértelműen előáll, majd a visszahelyettesítés módszerével az összes gyök számítható [Gáspár 07]. Az eliminációs módszer műveletigénye $O(n^3)$.

A Gauss-elimináció továbbfejlesztett változata a Gauss-Jordan módszer. Ebben az algoritmusban az eliminációs lépéseket nem monoton előrehaladó lépésenként kell elvégezni, hanem minden lépésben az összes megelőző elemre is végre kell hajtani az eliminációt.

A direkt eljárások további módszerei a mátrix felbontásán alapulnak. Az LU felbontás lényege, hogy a kiindulási mátrix $A=L\cdot U$ alakúra alakítható (faktorizálható), ahol *L* alsó-, *U* a felső háromszög mátrix. A felbontás után az $L\cdot y=b$ és $U\cdot x=y$ egyenletrendszerek megoldásával $L\cdot(U\cdot x)=b$ adódik. Az LU felbontás általános esetben megadható a Gauss-féle eliminációs lépésekkel [Press et al. 07].

A módszer egyik variációja, a PLU felbontás esetén, a háromszögmátrixok mellett egy ún. permutációs mátrix segítségével a zérus főminort tartalmazó *A* mátrixok is felbonthatóak (a *P* mátrix gyakorlatilag a főelemkiválasztáshoz használható) [Stoyan 02].

Az LU felbontás alkalmazható továbbá szimmetrikus és pozitív definit mátrixoknál *LDU* alakban, ahol a *D* egy pozitív együtthatójú diagonális mátrix [Galántai 03]. Az *LU* felbontás algoritmusa főbb pontjaiban lényegében ekvivalens az Gauss-eliminációval.

Az *A* mátrix egy másik lehetséges átalakítása a Householder-féle *QR* felbontás, mellyel egy ortogonális és egy felső háromszögmátrix szorzatára bontható. A felbontáshoz a Householder transzformációk vezetnek. Az eliminációs lépések során előáll $A=QR$ mátrix, amelynél a *Q* ortogonális $(Q\cdot Q^T=I)$ és az *R (Hessenberg tulajdonságú)* felső háromszögmátrix nyerhető, amely – hasonlóan a Gauss-eliminációhoz – az egyenletrendszer megoldásait szolgáltatja [Stoyan 02].

Az *LU* felbontás egy speciális esetében – szimmetrikus pozitív definit *A* mátrixnál – a felbontás felírható $A=L\,D\,L^T$ alakban. Ez az ún. Cholesky-felbontás kevésbé költséges, mint a Gauss-elimináció $O(n^3/6)$ és numerikusan stabilabb [Stoyan 02]. A felbontás nem pozitív definit mátrixokkal is működik, de ekkor a stabilitása nem minden esetben garantált.

A különböző LU felbontáson alapuló kiküszöbölési eljárások alkalmasak a nagyméretű, de ritka mátrixok feldolgozására. Az ilyen mátrixok gyakoriak a mérnöki modellezésben, pl. differenciálegyenletek megoldásakor, de figyelembe kell venni, hogy a megoldás nagyon gyakran hibákkal terhelt [Stoyan 02] [Trefethen 97].

A lineáris egyenletrendszereket megoldhatjuk továbbá ortogonalizációs, illetve spektrálfelbontás módszerekkel. Az ortogonalizációs módszer lényege az, hogy az eredeti feladatot

olyan ortogonális bázisra alakítjuk át, amely szerint a megoldás már Fourier-sorba fejthető. A spektrálfelbontáson alapuló módszerek lényege, hogy önadjungált mátrixok esetén, ha létezik ortonormált bázis, és ismerjük a mátrix sajátértékeit, akkor $O(n^2)$ műveletigénnyel kapható meg a megoldásvektor. A két módszer hátránya, hogy maga az ortonormált bázis keresés is költséges.

A fenti megoldó algoritmusok alapvetően szekvenciálisak és megalkotásuk a klasszikus RAM modellt követi (*lásd: 3.1.fejezet*). A ritka mátrixok egy speciális osztályánál a háromátlós (tridiagonális) mátrixoknál megadható olyan algoritmus, amely a feladat némi átalakítása után párhuzamosítható formában használható [Press et al. 07 p. 57.].

Iteratív megoldó algoritmusok

A lineáris egyenletrendszerek másik nagy csoportját alkotják az iterációs módszerek, melyekkel az egyenletrendszerek valamilyen pontatlan kiindulási vektorból fokozatos közelítési lépésekkel találják meg az „elég jó” pontosságú megoldásokat [Hageman 81]. A számítógépek teljesítményének növekedésével az ilyen jellegű módszerek jelentős szerepet kaptak [Golub et al. 94]. A direkt módszerek – a köbös műveletigény miatt – a feladatok méretének növekedésével túlságosan költségesnek bizonyulnak: a probléma sokszor nem oldható meg „ésszerű” időn belül. Emellett a numerikus hibák tovaterjedésével az eredmény hibája túlságosan nagyra nő (pl.: rosszul kondicionált mátrixok esetében). Az iterációs módszerekkel az algoritmus egyes lépései során a mátrix elemei nem változnak meg, így a hibaterjedés nem olyan nagyfokú. Az iterációs lépések alapműveletei a mátrix-vektor szorzatok képzése, amely a több processzort tartalmazó gépeken jól párhuzamosítható. A leginkább elterjedt iterációs módszereket az *1. táblázat* mutatja be.

Az iterációs megoldó módszereket már számos szerző összefoglalta *(Stoyan, Trefethen, Kelley, Choi, Press, Korn, Kelley, Ciarlet, Galántai),* az alábbiakban rövid leírást adunk a legfontosabb módszerekről, és az ezekben használt fogalmakról.

Iteratív megoldó módszer	*jel*	*Szerzők, publikálás éve*
konjugált gradiens	CG	Hestens, Stiefel, 1952
minimum reziduum	MINRES	Paige, Saunders, 1975
	SYMMLQ	Paige, Saunders, 1975
	MINRES-QLP	Choi, 2006
	SYMMLQ-QLP	Choi, 2006
bikonjugált gradiens	Bi-CG	Fletcher, 1976
előretekintő-Lánczos	LA-Lanczos	Parett, Taylor, 1985
általános minimum reziduum	GMRES	Saad, Schultz, 1986
négyzetes konjugált gradiens	CGS	Sonnevald, 1989
kvázi- minimum reziduum	QMR	Freund, Nachtigal, 1991
bikonjugált-stabilizált gradiens	Bi-CGSTAB	Van der Vorst, 1992
transzponálás nélküli QMR	TFQMR	Freund, 1993
egyszerűsített QMR	SQMR	Freund, Nachtigal, 1994
ABS rangszámcsökkentő módszer	ABS	Abaffy, Broyden, Spedicato, 1981
Legkisebb négyzetek módszere	CGLS	Hestens, Stiefel, 1952
	RRLS	Chen, 1975
	LSQR	Paige, Saunders, 1982
	RRLSQR	Paige, Saunders, 1982
Gaussian Belief Propagation	Ga-BP	Weiss, Bickson, 2001

1. táblázat: A napjainkban használt iteratív megoldó módszerek
(a táblázat nem teljes, csak a legfontosabb módszereket tartalmazza)

Krylov alterek

Az iterációs módszerek alapötlete a négyzetes mátrixú $Ax=b$ kiinduló feladatból az $x_{i+1}=Bx_i+f$ alakra való áttérés. Az iteráció során képezzük az ún. Krylov-féle alteret $B=(b, Ab, A^2b, A^3b, \dots, A^{n-1}b)$ alakban, és itt az elemi iterációs lépés legyen $x_1=x_0+Bb$.

Az ilyen modell alkalmas ortogonalizációs sémák, illetve számos iterációs megoldó algoritmus alapját képezni: Arnoldi-, Gauss-Seidel-, Lánczos-iteráció, konjugált gradiens (CG), minimum reziduum (MINRES), általános minimum reziduum (GMRES), bikonjugált-stabilizált gradiens módszer (Bi-CGSTAB) [Gáspár 07][Kelley 95].

Arnoldi-iteráció

Az Arnoldi-iteráció a Gram-Schmidt-féle ortogonalizációs módszer segítségével a Krylov bázisba való leképezést valósít meg. Az altér képzése abban tér el a Krylov módszertől, hogy a $q_i = x / \|x\|_2, i = 1..\mathrm{n}$ alakú ortonormált, ún. Arnoldi-vektorokkal feszít ki alteret. Az iteráció során előálló mátrix felbontás $A = QHQ^*$ alakú, ahol *H* egy felső-Hessenberg mátrix. Az arnoldi iteráció nem minden esetben stabil: amennyiben a lépések során zérusvektor áll elő, az iteráció nem folytatható.

Prekondicionálás

A direkt módszereknél bemutatott PLU felbontás mintájára az eredeti *A* mátrix a gyors iteráció érdekében felbontható. Amennyiben a felbontás $A = P - Q$ alakú, akkor az eredeti feladat $Px = Qx + b$ alakban is felírható. Itt megadható egy $x_{j+1} = P^{-1} Q\, x_j + P^{-1} b$ alakban adott iterációs sorozat, ahol *P* jelöli az ún. prekondicionáló mátrixot. A *P* megválasztása kulcsfontosságú a ritka *A* mátrixszal megadott feladatoknál, ugyanis ennek segítségével megadható többfajta gyors és stabil konvergenciájú algoritmus.

Gauss-Seidel-iteráció

A Gauss-Seidel-iteráció azon alapul, hogy az *A* mátrix felírható $A = L + D + U$ formában. Az iterációs lépések során, egy tetszőleges x_0 vektorból kiindulva megoldást közelítő vektorok képezhetők rendre $x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$ alakban [Stoyan 02, p. 106.].

Lánczos-iteráció

Lánczos Kornél sajátérték meghatározó módszerében bemutatta, hogy egy véletlen megoldásvektorból kiindulva $x_{j+1} = A\, x_j$ formában kellően nagy *n* számra $x_n / \|x_n\|$ formula segítségével a feladat maximális sajátértéke meghatározható. A módszer lényege, hogy a szimmetrikus *A* mátrixot először olyan *T* mátrixszá alakítja, amely tridiagonális tulajdonságú. A Lánczos iteráció során a Krylov-alterekbe képezés legfeljebb *A* rangjának megfelelő

lépésből áll. A modell alkalmazható a hermitikus mátrixok iteratív megoldására, és műveletigénye kisebb, mint az Arnoldi-iterációnak [Trefethen 97] [Choi 06]. Több megoldó módszer alapul a Lánczos-iteráció elvén: konjugált gradiens (CG), SYMMLQ, minimum reziduum (MINRES) indefinit szimmetrikus rendszerekre [Paige, Saunders 75].

Gradiens módszerek

A gradiens módszerek a szimmetrikus pozitív definit mátrixú feladatok megoldására alkalmasak. A módszer azt használja ki, hogy egy iterációban felhasznált x_i megoldásvektor hibáját a $\|r_i\|_2=\|A\cdot x_i-b\|_2$ reziduum vektor euklideszi normája adja. Amennyiben ezt sikerül minimalizálni, ezzel együtt a megoldásvektor hibája is monoton csökken. A gradiens módszert használó iterációs módszerek műveletigénye – iterációs lépésenként – ritka mátrixok esetén $O(n)$. Sűrű, vagy rosszul strukturált mátrixoknál $O(n^2)$ [Trefethen 97].

Konjugált gradiens módszer (CG)

A konjugált gradiens módszer az egyik legrégebbi iterációs eljárás [Hestens, Steifel 52], szimmetrikus pozitív definit mátrixú feladatoknál alkalmazható. A módszer célja, hogy az $x=A^{-1}b$ megoldásvektort, a mátrix invertálása helyett, az $f(x)=\frac{1}{2}x^TAx-b^Tx$ függvény minimalizálásával megtalálja. Ez az $r_{i+1}=r_i-\alpha_i A x_i$ iterációsorozat segítségével érhető el. A közelítések során, a soron következő lépések reziduum vektorai ortogonálisak. Ebből az következik, hogy az algoritmus maximális iterációszáma az *A* mátrix rangja lesz [Press et al. 07]. A CG módszer összes műveletigénye sűrű mátrixoknál legrosszabb esetben $O(n^3)$.

Legkisebb négyzetek módszere

A fizikai, műszaki és egyéb általános folyamatok megfigyelésekor – amelyek mérési hibával tarkítottak – Legendre és Gauss alapján megállapítható, hogy valamely mért értékek közelíthetők a $b(t)\approx\sum_{j=1}^{m}x_j\varphi_j(t)$ formulával, ahol $\varphi_j(t)$ jelöli az alkalmazott közelítő függvényrendszert. A mérések számának növelésével a mérési hibák kiküszöbölhetők: $\sum_{i=1}^{l}\left(\sum_{j=1}^{m}x_j\varphi_j(t)-b(t)\right)^2\rightarrow\min_{x_j}$ [Stoyan 02, p. 156.].

Ezt a módszert a lineáris egyenletrendszereknél a $\|Ax-b\|_2^2 \rightarrow \min_x$ formula írja le. Vagyis az euklideszi normában elvégzett lépések négyzetesen közelítik a pontos megoldást: $Ax \approx b$. A legkisebb négyzetek módszere a $A^T \cdot Ax = A^T \cdot b$ normálegyenlet megoldását jelenti.

A képletben szereplő $A^T \cdot A$ szorzat az ilyen módszerek gyenge pontja, mert a szorzatmátrix kondíciószáma az eredeti négyzetére nő. Ezért a szorzás helyett hatékonyabban használható az a módszer, ha a feladatot x-re kifejtjük: $x = (A^T \cdot A)^{-1} A^T \cdot b$, majd megadjuk a jobboldali mátrixszorzat *QR* felbontását, amely jellemzően egy ritka (pl. bidiagonális) mátrix megoldására egyszerűsödik [Stoyan 02].

A legkisebb négyzetek módszerére épülnek az LSQR [Paige, Saunders 82], CGLS, RRLS és RRLSQR [Choi 06] algoritmusok.

MINRES

A MINRES módszer olyan Lánczos-iteráción alapuló konjugált gradiens módszer, melyben a minimalizálás euklideszi normában történik. A lineáris egyenletrendszer $(A-sI)x = b$ alakú, ahol $A-sI$ szimmetrikus mátrix, amely lehet definit, indefinit és akár szinguláris is. Ebben a módszerben az iteráció célja a $\|(A-sI)x-b\|_2$ minimalizálása.

Egy alternatíva szinguláris feladatok megoldására a MINRES-QLP [Choi 06], amely a lehetséges MINRES megoldások közül a minimális lépésszámú. A nemszinguláris feladatokra prekondicionálással megadható egy hatékony algoritmus, a SYMMLQ [Paige, Saunders 75] és a SYMMLQ-QLP [Choi 06].

GMRES

Az általános irányú minimum reziduum módszer (GMRES) módszer a iterációs lépéseiben az Arnoldi-iterációt alkalmazza [Saad, Schultz 86]. Szimmetrikus *A* mátrix esetén egyenértékű a Lánczos-iterációt használó MINRES algoritmussal, de a GMRES fő erőssége a nem szimmetrikus feladatok megoldása.

A GMRES módszer kiegészíthető prekondicionáló mátrixxal, ekkor már a $MAx = Mb$ alakú egyenletrendszert kell megoldani, ahol *M* a prekondicináló mátrix. A cél az $\|I-AM\|_2 = \rho < 1$ formula minimalizálása [Kelley 95]. Speciális feladatokban (Poisson egyenletek, multigrid, váltakozó irányok módszere) alkalmazott *M* mátrixokkal az algoritmus gyorsan konvergál.

Bikonjugált gradiens módszer (BiCG)

A Lánczos-iteráció tridiagonális mátrixot állít elő. A nem hermitikus A mátrixra azonban a $A=QHQ^*$ átalakítás nem végezhető el. Helyette az $A^*=V^{-*}T^*\left(V^{-*}\right)^{-1}$ formula alkalmazható. Az átalakítás az ún. *tridiagonális biortogonalizációs* módszerrel történik. Az eljárás általában konvergens, de itt a reziduum normával mért hiba nem monoton csökkenő, és extrém esetekben az iterációs sorozat elakadhat. A megoldás előnye az iterációnkénti alacsonyabb műveletigény és a kisebb tárhely igénye (a GMRES módszerhez képest) [Kelley 95]. Az utóbbi évtizedekben számos javítás született a BiCG algoritmusra: az LA-Lanczos (look-ahead Lanczos), CGS (conjugate gradient squarred), QMR (quasi-minimal residuals) és változatai a TFQMR, SQMR, és a stabilizált bikonjugált gradiens (Bi-CGSTAB) algoritmusok hatékonyabb és jobb konvergencia tulajdonsággal jellemezhetők [Trefethen 97].

ABS rangszámcsökkentő módszerek

Az ABS módszer, amelyet a '80-as években Abaffy József és társai alapoztak meg [Abaffy et al. 84], széles körben használható lineáris egyenletek megoldására is. A módszer a magyar tudománytörténet értékes része, és mind a mai napig aktívan kutatott témakör. A módszer központi eleme egy mátrix transzformáció, amely az Egerváry-féle rangszámcsökkentő módszer egy továbbfejlesztése. A feladat alkalmas az olyan nem-meghatározott egyenletrendszerek (*egyenletek száma ≤ ismeretlenek száma*) esetén is megoldást szolgáltatni, amelyeknél más algoritmusok nem alkalmazhatók. Az ABS módszernek létezik *direkt* megoldó verziója [Abaffy et al. 84] (implicit LU ABS: Abaffy, Jeney, Galántai), illetve *iteratív közelítő* változata [Spedicato 2000].

Az ABS konjugálás az első lépésében egy véletlen becslésből indul, majd a második egyenletre is keres olyan megoldást, amely az előzőre is teljesül. Az eljárás m lépésben ismétlődik. Az eljárás alkalmas a nem kompatibilis vagy a redundáns egyenleteket is kiszűrni a feladatból. Az ABS módszer részben párhuzamosítható [Galántai 99].

Gaussian Belief Propagation (Ga-BP)

A hagyományos iteratív módszerek aszimptotikus közelítéssel érik el a közelítő megoldást [Weiss 01]. A Ga-BP módszer a következő minimalizálási problémán alapul: $\min_x\left(\frac{1}{2}x^TAx-b^Tx\right)$. Konvergenciája a klasszikus iteratív metódusoknál jobb. A konjugált

gradiens módszernél jelentkező stabilitási probléma nem fordul elő az algoritmus futása alatt [Bickson 09]. A módszer nem használ mátrix invertálást és alkalmas az elosztott többprocesszoros rendszereken üzenetküldés alapú párhuzamos működésre is.

4.3. Ismert párhuzamosítási lehetőségek

A lineáris algebra feladatok többprocesszoros feldolgozása és a párhuzamos algoritmusok már a múlt század '70-es évétől kutatott terület volt. Az akkori munkák közül Moler, Hellner, Dongarra, Gustafson és Jordan írásai számítanak általánosan elfogadott kiindulási alapnak [Moler 72] [Hellner 78] [Dongarra et al. 84] [Jordan 86]. Ebben az időben jelentek meg az első egyedi többprocesszoros gépek, jellemzően a leggazdagabb tengerentúli kutatóintézetekben. Ebben az időszakban alkották meg a mára már standardnak számító szekvenciális BLAS (*basic linear algebra subprograms*) és LAPACK – LinPACK (*linear algebra package*) csomagokat, melyek máig az alapját képezik a hatékony hardver-szoftver megoldásoknak. (A hardver-gyártók a rendszereik teljesítményét ezen rutinok alapján összeállított feladatok megoldásával mérik [Top500].)

A '90-es évekre a többprocesszoros gépek már széles körben elérhetőek, míg az ezredfordulóra a többmagos, hibrid, valamint a sokprocesszoros grafikus kártyákkal szerelt gépek válnak elterjedtté. Napjainkra már minden gép több feldolgozó egységgel rendelkezik, és általánossá igénnyé vált a feladatok párhuzamosítása.

A lineáris egyenletrendszerben a párhuzamosítható feladatok három szintje különíthető el, [Dongarra 2000] melyek a párhuzamosítandó algoritmusok hatékonyságát döntően befolyásolják:

- *vektor-vektor műveletek* például a vektorok $y:=y+\alpha x$ összeadása, vagy $d=\langle x,y\rangle$ skaláris szorzata, jellemzően *O(n)* adat- és műveletigénnyel,
- *mátrix-vektor műveletek* például a $y=A\cdot x$ mátrix-vektor szorzatok, $O(n^2)$ adat- és műveletigénnyel,
- *mátrix-mátrix műveletek* például $C=A\cdot B$ két mátrix szorzata, $O(n^2)$ adat, és $O(n^3)$ műveletigénnyel.

Párhuzamos környezetben a fenti három művelet az adott rendszer architektúrája szerint optimalizálható. A BLAS és LAPACK rendszerek párhuzamosítására számos megoldás

született (*lásd: 2. táblázat*), de jelen pillanatban nem létezik olyan általános megoldó rendszer, amely az általánosan értelmezett (tulajdonságok, méret) lineáris egyenletrendszereket bármilyen architektúrán hatékonyan megoldja.

Párhuzamos környezet	***Megjelenés éve és továbbfejlesztési időszak***	***Támogatott nyelvek***	***Párhuzamosítási technika***
ScaLAPACK	1995-2012	Fortran, C	MPI
PLAPACK	1993-1998	Fortran, C	MPI
SuperLU	2003-2012	C	szálak
PETSc	1995-2012	C	MPI, szálak, GPU, hibrid
Matlab Parallel	2004-2012	Matlab	szálak, multicore, szerver, GPU
PLASMA	2008-2012	Fortran, C	szálak
MAGMA	2009-2012	C+CUDA, C+OpenCL	szálak + GPU
NumPy SciPy	1995-2012	Python+MPI Python+PETSc	MPI
HPJPC	2000-2011	Java	szálak, socket
UMFPACK	1994-2012	Fortran, C, Matlab	részben párhuzamos (szálak, MPI)

2. táblázat: Gyakran használt párhuzamos keretrendszerek lineáris egyenletrendszerekhez

A numerikus matematika üzleti szoftverei közül meg kell említeni a MATLAB [Kepner 09], a MAPLE [Molnárka et al. 96], illetve a Mathematica [MATH] rendszereket, melyek a legtöbb direkt és iteratív módszerre kínálnak hatékony beépített függvényeket. (Ezekben a rendszerekben a párhuzamos programozási modellek későn, csak 2005 körül kezdtek megjelenni.)

Az elosztott memóriájú többprocesszoros rendszereken használható legelső szoftver-rendszerek (ScaLAPACK, illetve PLAPACK), alkalmasak a ciklikus blokkfelbontáson és kommu-

nikáción alapuló mátrix műveletekre. A SuperLU rendszer elsősorban a ritka mátrixú direkt módszerekkel megoldható feladatokhoz használható [SuperLU]. A PETSc elsősorban a ritka mátrixú feladatok megoldásához alkalmazható, és fejlesztői kidolgozták a legtöbb direkt, illetve iteratív algoritmust. Előnye, hogy többféle, szorosan csatolt, elosztott, illetve hibrid architektúrákon is alkalmazható [PETSc].

A PLASMA és MAGMA rendszerek olyan BLAS kiterjesztések, melyek az utóbbi évtizedben elterjedt többmagos, illetve a többmagos- és grafikus kártyákkal ellátott rendszerekre általánosan használhatók. Ezen rendszerek mindegyike alkalmas Fortran, vagy C nyelven megvalósított algoritmusok futtatására [PLASMA].

Szintén a BLAS rutinokra épül az UMFPACK rendszer. Csak ritka és nemszimmetrikus lineáris egyenletrenszerek megoldására használható. A fejlesztők részben párhuzamos rutinokat is beleépítettek, de a párhuzamosítás szintje messze elmarad az előzőektől [Davis 94].

Feltétlenül meg kell említenünk a Python programnyelvre épülő NumPy és SciPy környezeteket, amelyek az MPI, illetve a PETSc rendszerekkel együttműködve hatékony lineáris egyenletmegoldó komponensekkel bírnak [NUMPY].

Egy érdekes jelenség a Java környezetben kifejlesztett HPJPC környezet [Christopher 2000], amely alkalmas a szorosan csatolt architektúrájú gépeken, objektumorientált szemléletben leírni a lineáris egyenletrendszerek megoldó módszereit, de teljesítményében és hatékonyságában ez jelen pillanatban elmarad a fenti lehetőségektől.

A felsorolt lineáris egyenletrendszer megoldó általános rendszerek mellett egyedi, egy-egy speciális jellegű feladathoz kidolgozott párhuzamos algoritmussal számos kutatócsoport foglalkozik [Basermann et al. 97] [Galántai 99] [Duff 99] [Sosonkina 02] [Eijkhout 06], [Dongarra et al. 07] [Pashos et al. 09] [Bosilca et al. 12].

5. NAGY SZÁMÍTÁSIGÉNYŰ, VALÓS-IDEJŰ IPARI ALKALMAZÁSOK

Az előző fejezetekben áttekintettük azt a három tudományterületet, amelyek megalapozzák a nagy számítási igénnyel rendelkező feladatokhoz tartozó párhuzamos algoritmusok tervezésének és végrehajtásának módszertanát. A három témakör mindegyike önálló, külön-külön is kutatható tudományterület. A hardver eszközök gyors fejlődésével azonban ezen a három témakör között véleményünk szerint a hatékony algoritmustervezés érdekében kapcsolatot kell teremteni.

Az összekapcsolás és az ipari környezetbe való átültetés azonban nem triviális feladat. A mérnöki gyakorlatban gyakran jelentkeznek valósidejű feladatok, melyekben egy-egy numerikus algoritmus eredményét azonnal fel kellene használni, mint egy komplex folyamat részletszámítását. Így az ipari környezetre a fenti modellek csak korlatozással, vagy valamely termelés-optimalizáló szoftverrendszer moduljaként alkalmazhatók.

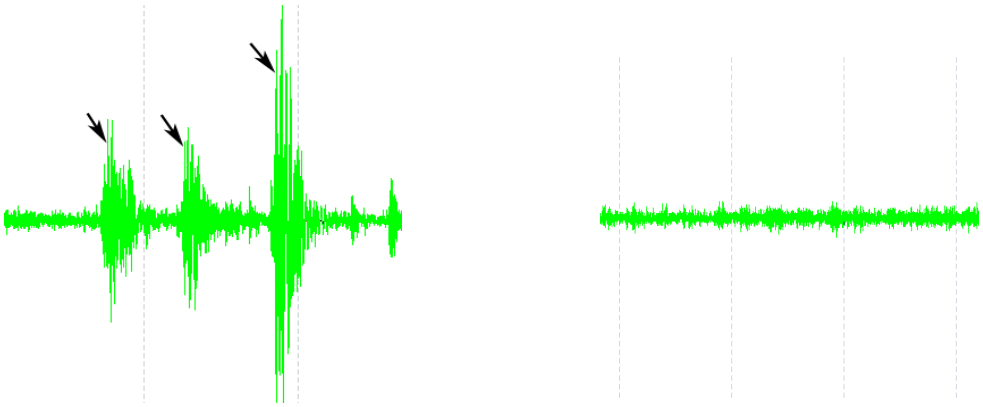
Az ipari termelési folyamatok során gyakran jelentkeznek olyan nagyméretű feladatok, amelyek klasszikus egyprocesszoros gépeken történő feldolgozása problémákba ütközik és hagyományos komputerekkel nem, vagy csak nagyon lassan oldhatók meg. Ezeket a problémákat a múltban vagy úgy küszöbölték ki, hogy azokat csak részben oldották meg (például gyakoriak a gyártósorokon a szúrópróba-szerű vizsgálatok), vagy valamely egyetem, illetve kutatóintézet bevonásával az adott problémát elkülönülten elemezték. Ezek azonban jelentős időráfordítást és költségeket jelentenek az egyes projektek alatt, vagy a cégek működésében.

Egy ilyen konkrét kutatási feladat volt egy győri székhelyű multinacionális vállalat számára készített, ipari gyártósorokon alkalmazott mérőberendezés tervezése elkészítése, tesztelése és üzembe állítása [Varjasi 08a]. A megoldandó feladat célkitűzése egy komplex mechatronikai-, informatikai mérnöki probléma modellezése és költséghatékony kifejlesztése volt. Ebben a dolgozat címével és témájával összekapcsolható fejlesztés egy olyan valós-idejű, szigorúan időkorlátos párhuzamos algoritmus tervezése, fejlesztése és legvégül validálása volt, amelyben gyártósorról érkező – nagy mennyiségű kimeneti adatot produkáló mérőberendezés – valós időben működő zaj-analizáltorként működött (*10. ábra*).

Az elkészült rendszer egy összetett, eseményvezérelt, időkritikus alkalmazás, amely alkalmas – DVD meghajtó egységek zajmintáit egyidejű, sokcsatornás feldolgozásával és összehasonlító elemzésével – az egyes munkadarabok minőségi osztályozását elvégezni.

Az alkatrészek zajvizsgálatára nagy zajterhelésű ipari gyártósoron került sor. A munkadarabok vizsgálatát egy-egy speciálisan zajvizsgálatra fejlesztett mérőeszköz végezte. A mérőeszközbe épített mérőhelyeken a munkadarabok egyes alkatrészeinek nagy pontosságú zajossága mérhető. A mérés során az operátor feladata az alkatrész azonosítójának leolvasása és az alkatrész behelyezése a mérőeszközbe. Ezután a feldolgozó automatika elvégzi a mérési folyamatot. A mérés végén az operátor eltávolítja a munkadarabot, és a meghatározott minőségi osztály szerint a munkadarabokat szétválogatja.

A feldolgozást végző berendezéseket logikailag három részre bonthatjuk: a speciális hardverre, az ezt vezérlő programra, és a zajmintákat tesztelő szoftverre. A vezérlő és a tesztelő szoftver vagy egyetlen nagyobb teljesítményű többprocesszoros, vagy két közepes teljesítményű független PC-n futhat.



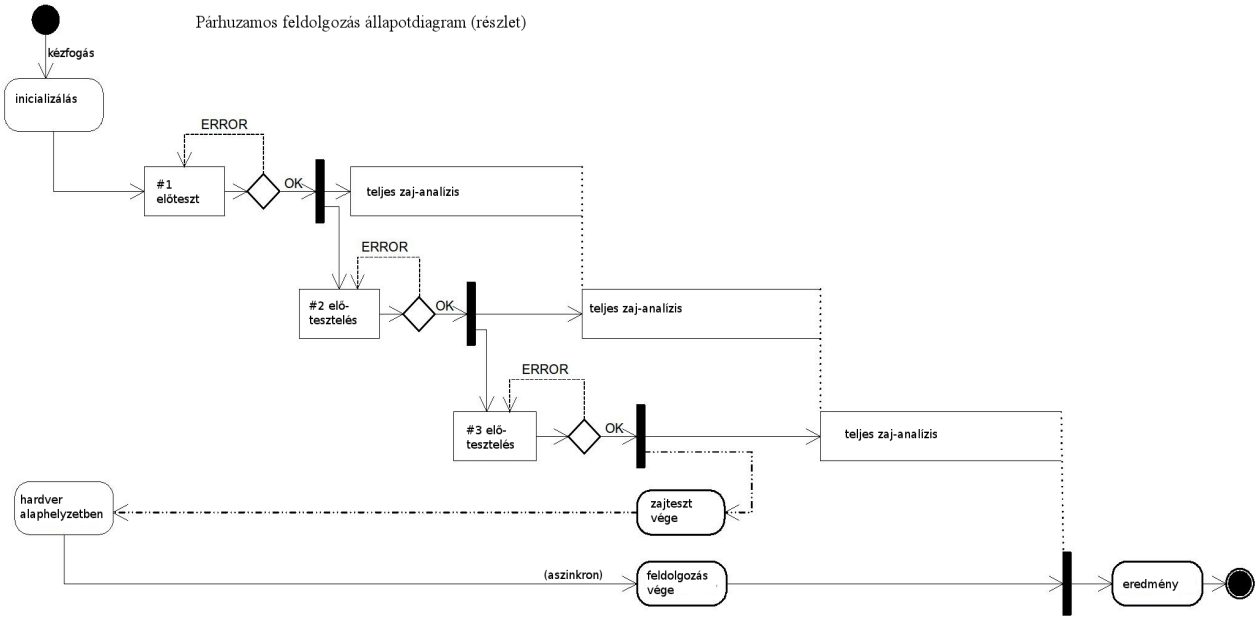
10. a, b. ábra: Rossz (a) és jó (b) minőségű munkadarabra jellemző zajminták

A speciálisan erre a feladatra tervezett gyártósori mérőberendezés elsődleges feladata az akusztikai érzékelők pneumatikus mozgatása és a munkadarabhoz illesztése volt. A munkadarabot többrétegű zajszigetelő réteg választotta el a gyártósortól, hogy az azon keletkező zajok a munkadarabon már ne legyenek mérhetők. A munkadarab főbb mechanikai alkatrészein elhelyezett gyorsulásmérők és mikrofonok szolgáltatták a feldolgozáshoz szükséges mérőjeleket. A lég- és háttérzajok szűrését egy külső mikrofon segítette a mérőrendszerben.

A hardvert vezérlő szoftver feladata a mérőállomás alacsony szintű vezérlése (mint az áram-, és levegőellátás, szenzorok állapotának vizsgálata, előerősítő kalibrálás, a zajmérő alkatrészek mozgatása stb.), továbbá a zajminták tárolása és a kommunikáció az operátorral. A zajmintákat feldolgozó szoftver elsődleges feladata a hardvervezérlő szoftver vezérlése, és a hardver által lemért hangfolyamok folyamatos, valós idejű kiértékelése. Az ipari körülmények között a magas jel-zaj arány miatt az egyes mintákat többször, ismételve kellett lemérni. A zajminták kiértékelése több lépésben folyt. A minták mérése és kiértékelése a gyártósoron a többi munkafolyamattal szinkronizáltan haladt (*11. ábra*). A kiértékeléshez a nagy pontosságú, magas mintavételszámú szabványos formátumú hangfolyamokat használtunk. (Microsoft PCM WAV, 16 bit, mono, tömörítés nélküli, 200-kHz-es mintavétellel.) A hangfolyamok kiértékeléséhez spektrális analízist [Vib 85], mintavételezést és normalizálást [Upton et al. 94], illetve gyors Fourier transzformációt használtunk [Wang et al. 07]. A hangminták magas mintavételi frekvenciája lehetővé tette az egészen apró – füllel már nem hallható – zajok kiszűrését is.

A feldolgozás a hangminták felvétele után két lépésben zajlott: egy nagyon gyors előtesztelő algoritmus megvizsgálta, hogy a munkadarab a főbb hibajelenségeket produkálta-e. Ez az előtesztelő rutin „azonnali” eredményt szolgáltat, futási ideje 10-100 ms közötti, így a mérőberendezés várakozási ideje minimalizálható. Abban az esetben, ha a mérés zajos, vagy az előtesztelés eredménye hibás volt, a munkadarabon a mérési folyamat előre meghatározott számig megismételhető.

A sikeres előtesztelés után a teljes hangmintán mélyreható elemzést végeztünk. Ez meghatározza a munkadarabok minőségi osztályát és használhatóságát. Az előtesztelés során a beérkezett hangmintákon először megvizsgáltuk, hogy a munkadarab hangmintái megkülönböztethetők-e a háttérzajtól [Pintér 08], majd csúcskereséssel és hangnyomás vizsgálattal megállapítható, hogy a hangminta értékelhető-e. A vizsgálatban először a hangminták spektrális analízise gyors Fourier transzformációval zajlik. Ezután a Zwicker-modell szerinti heurisztikus vizsgálat következik [Zwicker et al. 99]. Ezzel a módszerrel a hangminta frekvenciasávok szerinti feldolgozása során a kis jelszintű frekvenciák elhanyagolhatók. Az így kapott mintában ezután a zavaró jelek, kattogó zajok és a mozgó alkatrészek hibájára utaló hangminta információk keresésével, és ezek hangnyomásának mérésével a munkadarab zajosságának minőségi kategóriája meghatározható [Nagy, Molnárka 07].



11. ábra: A feldolgozó szoftver párhuzamos feldolgozó moduljának állapotdiagramja, háromszori előírt ismétlésszám esetén. Forrás: [Varjasi 08b]

Szekvenciális feldolgozás esetén a futási idő hosszú, így ez csupán laboratóriumi körülmények között használható, mert a mérés és a feldolgozás a gyártósor ciklusidejét sokszorosan meghaladja. A zajmérések során a mérési sorozatot többször meg kell állítani, amíg az adatfeldolgozás véget nem ér és ez folyamatosan futó gyártósoroknál termelékenységi szempontok miatt nem megengedhető. (Egy mérőfolyamat fizikai végrehajtása – szoftveres elemzés nélkül – a mérőállomáson 22 s volt, amely meghaladta a 20 másodperces gyártósori periódusidőt.) Abban az esetben, ha a feldolgozásban zajos jel miatt ismétlésére is sor kerül, a napi futásidőre vetített összesített állásidő jelentősen megnő.

Az algoritmusok fejlesztésekor a működési hatékonyságot a mérőberendezés futási idejének függvényében adtuk meg. A mérés során a futási idő három összetevőből áll. Az első szakasz a munkaállomás üresjárati ideje (initialize time), ezalatt az operátor munkadarabot cserél és elindítja ill. lezárja a mérési folyamatot. Ezalatt a mérőberendezés tétlen. A mérési folyamat alatt a munkaállomás vagy utasításra várakozik (idle time), vagy méréseket végez (work time).

A párhuzamos algoritmussal vizsgált munkaállomásokon a várakozási idő 50%-kal csökkent (17-19 s) és a hangminták párhuzamos feldolgozás a hangminták effektív feldolgozási idejét kb. 40%-kal javította (54-56 s). A párhuzamosított algoritmus már alkalmasnak ígérkezett a

gyártósori futtatásra is, hiszen a szoftver három függetlenül kiépített munkaállomás adatait dolgozta fel egyidejűleg, azaz egy időben három munkadarabot tudott megvizsgálni (így a gyártósoron rendelkezésre álló időtartam 60 s volt). A mérési folyamatok a gyártósor ciklusidejével azonos nagyságrendbe kerültek. A párhuzamos algoritmus optimalizálásával a mérési ismétlések hatékonyságán javított. A továbbfejlesztett vezérlési utasításokkal a munkadarabok zajvizsgálatánál a várakozási idő tovább csökkent (8-9 s), az előtesztelő folyamat ismétlési és vezérlési funkciókkal való ellátása pedig látványosan csökkentette a hangminták feldolgozási idejét (39-41 s) [Varjasi 08b].

A „hétköznapi" mérnöki környezetben nem állnak automatikusan rendelkezésre szuperszámítógépek, illetve nagy teljesítményű klaszter-rendszerek, így a az elkészített mérőberendezéshez a szoftverfejlesztést egy többmagos asztali számítógépre optimalizáltan végeztük 2006-2007-ben. Az elkészített mérőberendezés a gyártósorra került és folyamatosan, megbízhatóan végezte el az addig humán erőforrással végzett hallás- és zajvizsgálatokat [Varjasi 08b] [Nagy, Molnárka 07]. (A kutatás 2008-ban befejeződött[2].)

A bemutatott valós-idejű ipari alkalmazás azonban nem tárhatta fel a vizsgált tudományterületek mélyebb rétegeit. Elsősorban mérnöki feladatnak bizonyult a párhuzamos és elosztott rendszerek programozására, így kvantitatív elemzésekre kevés lehetőséget biztosított. Ezért a következőkben inkább olyan területek felé fordítottuk figyelmünket, amelyek közelebb álltak a tudományos alapkutatáshoz és új algoritmusok, modellek kidolgozását tették lehetővé.

2 A gazdasági válság, a cég tulajdonosi struktúrájának változása és költségek csökkentése miatt a gyártósort először Ukrajnába költöztették, majd profilváltás miatt végleg leszerelték.

6. A REZIDUUM MINIMALIZÁCIÓN ALAPULÓ ITERÁCIÓS ALGORITMUSOK

A dolgozat első felében tárgyalt témakörök összekapcsolása *(párhuzamos és elosztott rendszerek, párhuzamos programozás és numerikus algoritmusok)* nem triviális feladat. A szakirodalom és a gyakorlati tapasztalatok is azt mutatják, hogy jó párhuzamos algoritmusokra nem létezik univerzális modell. Helyette az adott feladattól függő, az adott számítógépre szabottan, egymással hatékonyan együtt működő komponensekből érdemes modellt, majd algoritmust alkotni.

Stoyan kijelenti, hogy: *„a párhuzamos számítógépek terjedésével a numerikus algoritmusok megítélése is változik. ...[ez] azt jelentheti, hogy a hagyományos, szekvenciális számítógép legcsiszoltabb algoritmusait fel kell adnunk... Előnyben részesülnek olyan algoritmusok, amelyek sok, egymástól függetleníthető lépésből állnak."* [Stoyan 02 p. 29.].

Ez a gyakorlatban azt jelenti, hogy a sokprocesszoros párhuzamos gépekre szánt algoritmusok nem lesznek tömörek és robusztusak. Az adatok tárolása sokszor redundáns, vagy processzorok között széttagolt. Neumanni értelemben sem mondhatók hatékonynak (a Neumann-elvek soros végrehajtású gépen értelmezett soros algoritmusokról szólnak), helyettük valamilyen blokkosított, vagy több ismétlődő és egymástól független elemből felépülő, összetett struktúrát kell megalkotnunk.

A 4.2.1. fejezetben ismertetett lineáris egyenletrendszerek megoldására szolgáló algoritmusok egyprocesszoros végrehajtás szempontjából ilyen *„hatékony és csiszolt"* modellek. A 4.3. fejezetben ezen modellek belső párhuzamosítási lehetőségeit kihasználó párhuzamos algoritmusokról is szóltunk. Ezek legtöbbje olyan lehetőségeket tartalmaz, amelyek az algoritmus egyes lépéseit párhuzamosítható részfeladatokkal helyettesíti (pl. BLAS-alapú párhuzamosítás). Így több iteratív javító algoritmus csak kis részben tudja kihasználni a teljes párhuzamos számítógépet, ezért az Amdahl-törvény értelmében jelentős gyorsulás ezekben nem érhető el. Például a gradiens módszeren alapuló algoritmusok – amely a soros végrehajtás szempontjából hatékonyak – minden egyes iterációban két előzetes lépés alapján állít elő egy újabb, a pontos megoldáshoz legmeredekebben közelítő vektort. Ez

sokprocesszoros párhuzamosítás esetén szűk keresztmetszetnek bizonyulhat a sorozatos szinkronizálási kényszer miatt.

Az alábbiakban bemutatunk egy olyan algoritmust lineáris egyenletrendszerek megoldására, amelynek alapja a véletlenszerű irányokból való közelítés. A szekvenciális algoritmust és a modell konvergenciájának bizonyítását [Molnárka, Miletics 05] közli. Ezen algoritmus, hagyományos értelemben nem hatékony (a legkisebb négyzetek elvén alapuló, lassan konvergáló iterációsorozat), párhuzamosításra azonban alkalmas. Ezért a szekvenciális modell mellett bemutatjuk ennek párhuzamosított verzióit is. (A párhuzamos algoritmusok mindegyike a szerző saját eredménye.)

6.1. Egy reziduum minimalizáción alapuló iteratív algoritmus

Legyen A egy tetszőleges mátrix. Tekintsük alapfeladatnak a (4) alakú lineáris egyenletrendszerek megoldását.

$$Ax = b \tag{4}$$

A probléma numerikus megoldására számos, az előző fejezetben ismertetett módszer ismert. Nagyméretű feladatok megoldásánál a leghatékonyabb algoritmusok az iterációs módszerek, a direkt megoldáson alapuló algoritmusok jellemzően nem stabilak numerikusan. Az iteráción alapuló módszerek nagy része csak bizonyos speciális feltételeket kielégítő A mátrixok esetén hatékonyak (pl. szimmetrikus, ritka, vagy pozitív definit mátrixok).

$$A^T Ax = A^T b = v \tag{5}$$

Ha az eredeti (4) egyenletrendszer helyett az egyenlethez tartozó (5) normálegyenletet oldjuk meg, akkor a megoldó algoritmusokból több áll rendelkezésünkre, ám óriási hátrány lesz az, hogy – általános és teljes rangú négyzetes mátrixok esetén – az új $A^T A$ mátrix szimmetrikus pozitív definit lesz, de kondíciószáma az eredeti mátrix kondíciószámának négyzetére növekszik! A kondíciószám négyzetes növekedése pedig az elérhető megoldás pontosságának csökkenését okozza (*lásd: 4.1. fejezet*). A klasszikus, szekvenciális algoritmussal működő iterációs módszerek konvergenciája ebben az esetben sokkal lassabb lesz. Ha a legkisebb négyzetek elvén alapuló iteratív minimalizáló eljárást alkalmazunk – amelyben nem szükséges előállítani a normálegyenletet –, akkor a konvergencia sebessége lassabb

lesz, de az elérhető közelítő megoldásvektor hibája kisebb marad. Ezek között van olyan is, amely sokprocesszoros realizálásra alkalmas.

A legkisebb négyzetek módszerét alkalmazó iterációs módszerek mindig valamilyen függvényminimalizáláson alapulnak [Paige, Saunders 82]. A fenti egyenletrendszer is a legkisebb négyzetek módszeréből származtatható, ahol meg kell oldani a következő minimalizálási feladatot:

$$\min_{x\in\mathbb{R}^n}\|Ax-b\|_2^2=\min_{x\in\mathbb{R}^n}\langle Ax-b,Ax-b\rangle=\min_{x\in\mathbb{R}^n}\langle r,r\rangle, \tag{6}$$

azaz az egyenletrendszer hibavektorainak euklideszi normában vett négyzetes minimalizálását, ahol $r=Ax-b$ az x-vektorhoz tartozó hibavektor, vagy más szóval reziduum.

Az eredeti algoritmus

A minimum feladat megoldására számos kutatási eredmény és hatékony módszer ismert (LSQR, CGLS, RRLS, RRLSQR, *lásd 4.2.1. fejezet*), most válasszunk egy olyan lehetséges megoldó algoritmust, amelyet a következő állításokban összefoglalt megfigyelésből az alábbiak szerint mutat be [Molnárka, Miletics 05].

Legyen $A\in\mathbb{R}^n\to\mathbb{R}^n$ teljes rangú (nem szinguláris) mátrix, legyen továbbá x^α és x^β két tetszőleges, de olyan különböző n-dimenziós vektor melyekre $A(x^\alpha-x^\beta)\neq 0$. Vezessük be a következő jelölést: $r^s=Ax^s-b,\ \ s=\alpha,\beta,$ továbbá

$$r^{\alpha\beta}=c\cdot r^\alpha+(1-c)r^\beta, \tag{7}$$

és

$$c=\frac{\langle r^\beta-r^\alpha,r^\beta\rangle}{\|r^\alpha-r^\beta\|_2^2} \tag{8}$$

Könnyen kaphatjuk ezekből, hogy $Ax^{\alpha\beta}-b=r^{\alpha\beta}$, ahol $x^{\alpha\beta}$ a következő összefüggéssel számítható:

$$x^{\alpha\beta}=c\cdot x^\alpha+(1-c)x^\beta \tag{9}$$

A fenti számításokban a következő összefüggéseket használhatjuk:

$$\|r^{\alpha\beta}\|\le\min_{\alpha,\beta}(\|r^\alpha\|,\|r^\beta\|), \tag{10}$$

azaz a képzett új közelítő vektor reziduumának euklideszi normája nem nagyobb a minimális reziduum normájánál.

Az $x^{\alpha\beta}$ vektor előállítási módja a módszert alkalmassá teszi iterációs sorozat előállítására.

Tétel 1.:

Ha adott (6) minimalizálási feladat és adott két különböző, de tetszőleges x^{α} és x^{β} vektor melyekhez az r^{α} és r^{β} – egymástól különböző – reziduum vektorok tartoznak, akkor az n dimenziós térben az x^{α} pontot az x^{β} ponttal összekötő egyenes mentén $Ax^{\alpha\beta}-b=r^{\alpha\beta}$ megoldása az az $x^{\alpha\beta}$ vektor lesz, melyet c konstans (8) értéke mellett kapunk. Az új reziduum és az új közelítő megoldásvektor a (7) és (9) összefüggésekkel számítható. [Molnárka, Miletics 05]

A tétel bizonyítása azon alapszik, hogy a minimalizálandó (6) feladat egy alulról szigorúan konvex

$$\min_c f(c)=\min_c \langle c\cdot r^{\alpha}+(1-c)r^{\beta}, c\cdot r^{\alpha}+(1-c)r^{\beta}\rangle=\min_c \|r^{\alpha\beta}(c)\|_2^2 \tag{11}$$

függvény, teljes rangú *A* mátrix esetében. A minimalizálás során felhasználható a következő tulajdonság: $\frac{\partial^2 f(c)}{\partial c^2}=2\cdot\|r^{\alpha}-r^{\beta}\|^2>0$ [Molnárka, Miletics 05, p. 3.].

Mivel *f(c)* egy pozitív főegyütthatójú másodfokú polinom, amely alulról szigorúan konvex, ebből az következik, hogy az iterációsorozat konvergens.

6.2. Véletlen irányokból való közelítés

A klasszikus iteratív módszerekkel ellentétben ennek a reziduum minimalizáló algoritmusnak az újdonsága az előre meg nem határozott irányokban történő lépések sorozata. Az első eltérés itt mutatkozik a gradiens-, vagy a bikonjugált gradiens módszerekhez [Dongarra 2000] [Eijkhout 06] képest. Továbbá az is jelentős eltérés, hogy *minden esetben véletlen* egyenesek mentén számítjuk ki a minimumokat (minden iterációs lépésben). Ezért szekvenciális esetben az algoritmus konvergenciasebessége lassabb lesz, hiszen nem törekszik a legnagyobb értékű lecsökkenésre. Legfőbb előnye viszont az, hogy az algoritmus nem akad el valamely lokális minimumnál, hanem folytatni tudjuk az algoritmust *bármilyen* függetlenül választott próbavektorral.

Ez az újdonság alkalmassá teszi az algoritmust több $x^k, k=1,2,3... \in \mathbb{N}$ független ágon futó párhuzamos számítás elvégzésére. Mindez végrehajtható úgy, hogy a későbbiekben minden párhuzamosan előállított eredmény alkalmas lehet a többi ág eredményeinek javítására.

ALGORITMUS 1.

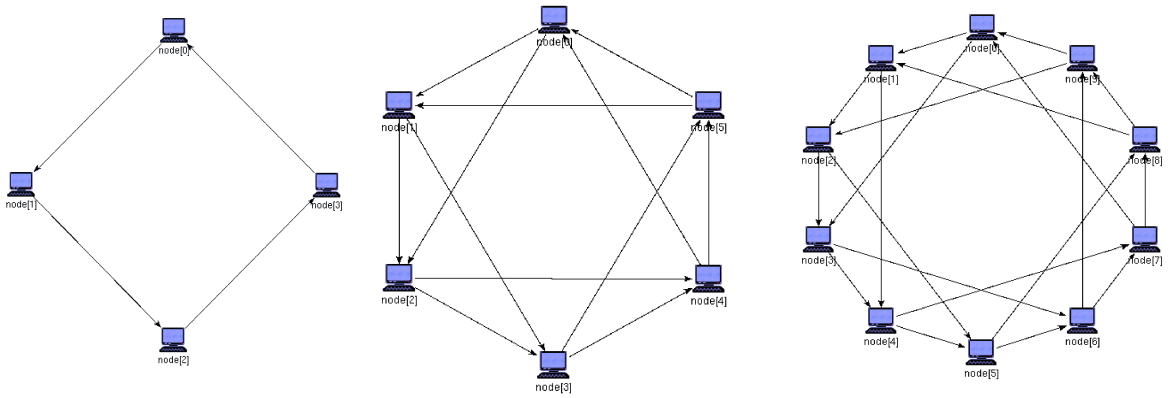
1. Legyen x^1 egy véletlen *n*-dimenziós vektor és legyen ε az elérni kívánt számítási pontosság.
2. $r^1 = Ax^1 - b$
3. Legyen x^2 olyan véletlen vektor, amelyre $r^1 - r^2 \neq 0$
4. $c^{12} := \dfrac{\langle r^2 - r^1, r^2 \rangle}{\|r^1 - r^2\|_2^2}$
5. Számítsuk ki az új véletlen vektort és reziduumát:
   $x^{12} := c^{12} \cdot x^1 + (1 - c^{12}) x^2$ és
   $r^{12} := c^{12} \cdot r^1 + (1 - c^{12}) r^2$.
6. $x^1 := x^{12}$, $r^1 := r^{12}$.
7. Ha $\|r^1\|_2 < \varepsilon$, akkor a kívánt pontosságú megoldás x^1, különben vissza a 3. lépéshez.

Ez a szekvenciális algoritmus alkalmas egy, a megoldást közelítő vektorsorozat, és reziduumsorozat előállítására [Molnárka, Varjasi 05]. (A véletlen értékek előállítására alkalmasak az egyenletes eloszlású álvéletlenszám generátorok.) A szekvenciális algoritmus validálása MATLAB környezetben történt. A tesztelés során az iterációs sorozat valóban a megoldáshoz konvergált.

6.3. Párhuzamosítási lehetőségek

A szekvenciális algoritmus kisebb átalakításokkal felkészíthető párhuzamos futásra. A párhuzamos futtatáshoz egy több gépből álló LAM/MPI alapú párhuzamos klaszterrendszert építettünk ki. A számításban részt vevő számítógépek azonos teljesítményűek voltak, a hálózat a futtatások során hibamentesen működött. A vizsgálatokat *12. ábrán* látható többféle topológián elvégeztük. A párhuzamos kódot az OMNET++ diszkrét idejű eseményvezérelt szimulátor segítségével futtattuk (az üzenetváltások vizuális nyomkövetésével) [Varjasi 06a],

majd később egy hatékonyabb és tisztán MPI alapú megoldást is kifejlesztettünk [Varjasi 06b]. (A párhuzamos algoritmusok összehasonlításához és hatékonyságának elemzéséhez lásd [Körfgen 06] munkáját.)



12. ábra: A párhuzamosítás során felhasznált gyűrű topológiák

Az egyes modellekben az irányított gráffal megadott topológia szerint minden csomópont a szomszédos csomópontnak küldhet, illetve fogadhat véletlen vektorokat (ez fontos szerepet kap a második algoritmus *vi, vii* lépésénél). Az összetett topológiát használó esetekben egy-egy távolabbi csomópont felé is küldhető, illetve attól is fogadható véletlen vektor. Az adatcseréktől az algoritmus hatékonysága javul [Molnárka, Varjasi 05], azaz a megadott pontosságú megoldás kisebb számítási idő alatt kapható meg.

ALGORITMUS 2. A PÁRHUZAMOS REZIDUUM MINIMALIZÁLÓ ALGORITMUS

1. Legyen N a csomópontok halmaza és legyen ε az elérni kívánt számítási pontosság.
2. Párhuzamosan $\forall n \in N$ csomópontra: x^1 vektor generálása.
3. Párhuzamosan $\forall n \in N$ csomópontra: *Művelet()* amíg megoldás érkezik.
4. Eredmények közlése.

Művelet()

i. x^2 véletlenvektor generálása

ii. *do*

iii. $r^1 := Ax^1 - b$ és $r^2 := Ax^2 - b$, amelyre $r^1 - r^2 \neq 0$

iv. $c^{12} := \frac{\langle r^2 - r^1, r^2 \rangle}{\|r^1 - r^2\|_2^2}$

v. $x^{12} := c^{12} \cdot x^1 + (1 - c^{12}) x^2$, $r^{12} := c^{12} \cdot r^1 + (1 - c^{12}) r^2$,

$x^1 := x^{12}$ és $r^1 := r^{12}$

vi. *ha **felt1** akkor*

x^1 küldése a szomszédos csomópontnak, x^2 fogadása

vii. *különben ha **felt2** akkor*

x^1 küldése egy távoli csomópontnak, x^2 fogadása

viii. *különben* új x^2 generálása

ix. *while* $\|r^1\|_2 < \varepsilon$

x. *return* x^1.

Látható, hogy a párhuzamosításra a *vi.* és *vii.* lépés alkalmazható. Az algoritmusban szövegesen jelzett „szomszédos csomópont” meghatározása a topológiában meghatározott élek mentén történik. Ez a szomszédosság kijelölhető egyedi gráfokkal, vagy a *12. ábrán* található irányított élű gyűrű topológiával. (Az irányított élek használata nem szűkíti az algoritmus hatáskörét.)

A *felt1* és *felt2* az adatcserék végrehajtásának leírására szolgáló feltételes kifejezés. Ez a kifejezés tulajdonképpen egy olyan optimalizálandó feladat megoldását jelenti, amely a lineáris egyenletrendszer méretétől, kondíciószámától, a választott topológiától és a rendelkezésre álló hardvertől függ. Ennek kvantitatív vizsgálatára nem vállalkoztunk, hanem helyette tapasztalati úton, a lineáris egyenletrendszer, a választott topológia és a futási eredmények alapján választottunk alkalmas kifejezéseket.

Kiindulásként azt az egyszerű lépést használtuk, hogy egy csomópont bizonyos iterációszám (a *ii-ix.* ciklus ismétléseinek száma) elérése után az addigi legjobb eredményét a következő szomszédos csomópont felé elküldi (*felt1:* $lépés = 1, 10, 100, \ldots \in \mathbb{N}$). A küldés után a megelőző szomszédos csomóponttól érkező véletlen vektorral folytatja a számítást. Mivel a feltételes kifejezés végrehajtása minden processzornál ugyanabban az iterációs lépésben történik, ezért a processzorok közötti adatcserék a kommunikációs csatornát egyszerre terhelik. Ezért a konkrét számítógépes megvalósítás során az adatcserék ütemezését úgy becsültük,

hogy sem a túl gyakori, sem túl ritka adatcsere nem hatékony. A túl gyakori adatcsere azért elkerülendő, mert nem létezik olyan hardver, amely az adatokat késleltetés nélkül vinné át a számítógépek/processzorok között. A feladat méretének – illetve a topológia összetettségének – növekedésével az átviteli csatornák telítődnek. (Megjegyzés: tulajdonképpen ezért kell bevezetni a *vi.* lépésben ezt a korlátozó feltételt, hiszen az algoritmust felírhatnánk úgy is, hogy minden egyes iterációs lépésben legyen adatcsere.)

A túl ritka adatcsere pedig azért nem szerencsés, mert ekkor az algoritmus egymástól független utakon keresné a közelítő megoldást (egymás mellé helyezett szekvenciális feladatokként). Ilyenkor azonban a több processzor használatával az eredeti Algoritmus 1.-nél csak rosszabb hatékonyságú megoldást érhetnénk el.

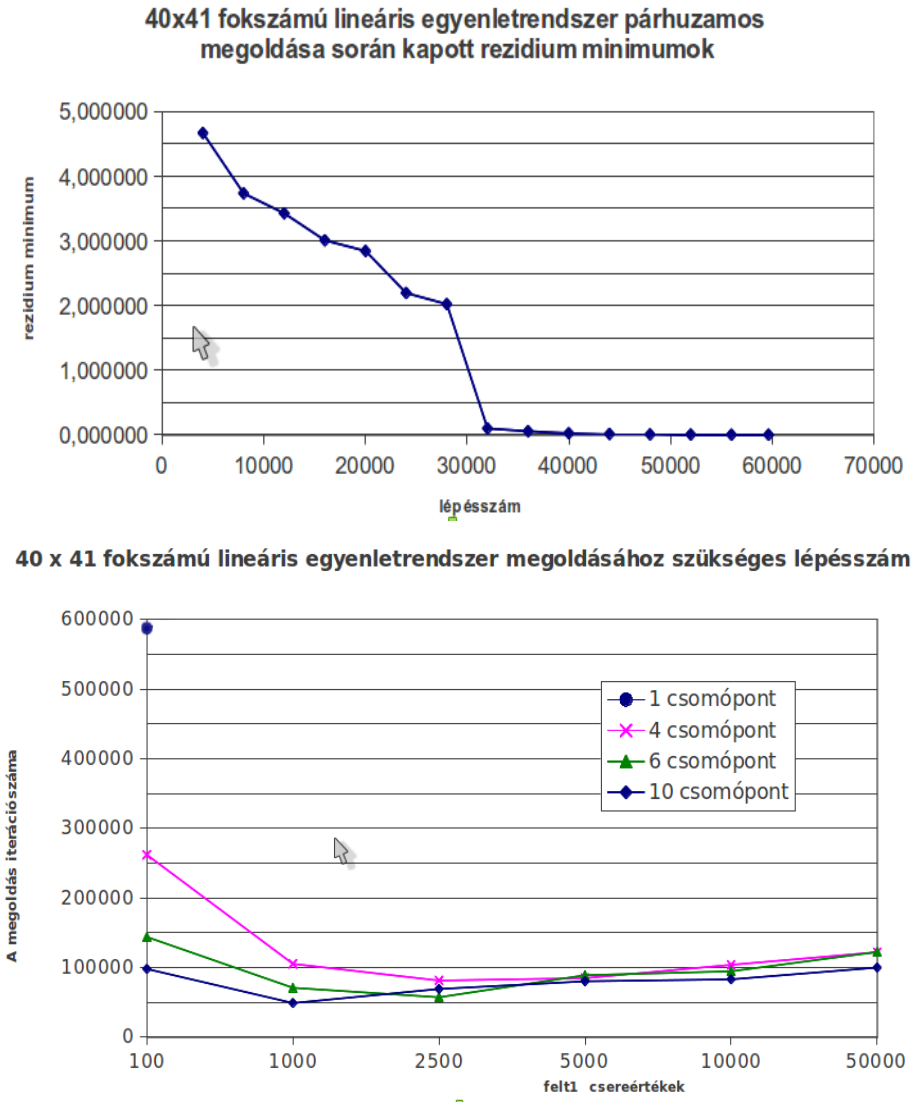
Az algoritmus *vii.* pontja az olyan topológiák esetén használt kommunikációs lépés, ahol a csomópontok között értelmezett egy második, „távolabbi" összeköttetés is. A *felt2* kifejezés a *felt1*-hez hasonlóan tapasztalati úton meghatározott értékű. A kifejezés bevezetésével azt szeretnénk elérni, hogy a szomszédos csomópontok közti kommunikáció mellett legyen egy olyan lépés, amely az eseménytér egy másik tartományából kap közelítő megoldásvektorokat. A *felt2* kifejezés megadható egy konkrét iterációszámmal, vagy *felt1*-től függő kifejezéssel. Az alábbiakban konkrét megoldások alapján elemezzük az algoritmusunkat.

6.3.1. A párhuzamos algoritmus futtatási eredményei

A párhuzamos algoritmus alapján elkészítettük a felvázolt gyűrűs topológián futásra alkalmas programot. A program tesztelésére több egyenletrendszert is megvizsgáltunk. A vizsgálat során *n=10, 20,..., 100* méretű lineáris egyenletrendszer közelítő megoldását végeztük el (minden méretből 20-20 különböző esetet vizsgáltunk). A választott számítógépes környezet egy 10 db egyprocesszoros gépből álló számítógépes klaszterrendszer volt, melyben a számítógépek közti kommunikációt 100Mbit/s sebességű hálózati kapcsolat szolgáltatta.

A *13. a. ábrán* látható *egyetlen kiválasztott* csomópont reziduum értékeinek alakulása az iterációs lépések során. A grafikonon ábrázolt eredményekhez a következő kifejezéseket használtuk: *felt1: (lépés==1000), felt2: (lépés==n/10).*

Megjegyzés: A bemutatott feladatnál a feltételek az iterációszám bizonyos értékeinél válnak igazzá. A kifejezés csak az adott számítógépes környezet lassú átviteli csatornája miatt kapott ilyen magas értéket. Mivel az *Algoritmus 1.* rosszabb konvergencia tulajdonságú, mint a konjugált gradiens módszer, ezért a futása nem ad pontos megoldást n lépés után. Ezért szerepel *felt2*-ben a lineáris egyenletrendszer méretével összefüggő kifejezés.



13. a, b. ábra: Algoritmus 2. párhuzamos futtatási eredménye (n=40, p=1..10) [Varjasi 06b]

A *13. b. ábrán* láthatjuk a különböző számú processzort felhasználó kísérletek *átlagolt* eredményeit, ahol egy $n=40$ méretű, korábban generált mátrixú egyenletrendszert mutatunk be. Az ábráról leolvasható, hogy a szekvenciális algoritmushoz képest a párhuzamos végrehajtás során csökkent a megoldáshoz szükséges iterációszám, és ezzel együtt a végrehajtás

ideje. Ez azzal magyarázható, hogy a két adatcserét leíró lépésnél az addigi legjobb közelítő megoldást nyújtó x^1 vektort küldjük át, így a csomópontok olyan információkhoz juthatnak, amelyeket független processzorok egymásnak szolgáltatnak. A mérések elemzéséből kiderül, hogy a párhuzamosított algoritmus hatékonyan, és a várt eredményeket jóval meghaladóan (6 csomópontnál tízszeres, 10 csomópontnál tizenkétszeres növekedéssel) javította a szekvenciális algoritmus működését.

Ugyan a vizsgált környezet és a feladatok osztálya szerény, az itt jelentkező eredmények mégis reményt keltőek. Az eredmények azt mutatják, hogy a párhuzamosítás hatékony lehet és nemcsak a több processzor használatából eredő javulás mérhető, hanem megfigyelhető az ún. szuper-gyorsító hatás is (ezt bővebben majd az *6.5. fejezetben* fejtjük ki).

Az egymástól független processzorok közötti adatcserék használatával az algoritmus konvergenciája megmarad. (Az algoritmus stabilitásának vizsgálatakor olyan kísérleteket is végeztünk, amelyben a *vi.* vagy *vii.* lépésben az adott csomópont legjobb eredményvektora helyett egy arra merőleges vektort – amely az iterációt „elrontja" – küldtünk tovább a következő csomópontnak. Az algoritmus konvergenciája – megnövekvő iterációszám mellett – ilyen esetben is megmaradt.)

A hatékonyság javítása

A *13. a. ábrán* továbbá megfigyelhetjük azt is, hogy bizonyos iterációszám után, – ami ezzel arányos számú adatcserét és új véletlen vektort is jelent – az algoritmusban a konvergencia ugrásszerűen felgyorsul. **Ez rávilágít a párhuzamos algoritmus egy egyedi tulajdonságára: a független komputerek közötti adatcsere során nemcsak a reziduum-norma monoton csökkenését figyelhetjük meg, hanem egy új jelenség is kialakul. Ez nem más mint a független gépek használatával elérhető, valóban véletlen vektorok kombinációjának hatása, amely az algoritmus nemdeterminisztikus működését szolgáltatja.** Ez a nemdeterminisztikus működés kérdésessé teheti a vizsgálatok teljes, minden részletében azonos reprodukálhatóságát. Vizsgálatainkban igyekeztünk olyan kiinduló feladatokat kiválasztani, amelyeknek megoldása ismert és a megoldás során előálló közelítő vektor a pontos eredménnyel összehasonlítható. Így az algoritmus által szolgáltatott reziduális hiba mellett a megoldás abszolút hibája is mérhető volt. Az *Algoritmus 2.* által

szolgáltatott megoldások azonos kiinduló feladatokból, azonos véletlenszám generátor kezdőértéket alkalmazva azonos topológián reprodukálhatók.

A megfigyelt – párhuzamos futtatásból eredő – javító lépések jelensége azt támasztja alá, hogy a bemutatott párhuzamos algoritmus evolúciós jelleget mutat.

Evolúciós jellegű algoritmusnak [Futó 03] az olyan feladatokat tekinthetjük, amely az egyedek egy kiinduló populációból, különböző operátorok hatására (új egyedek létrehozása, mutáció, keresztezés) a populáció egyedeit úgy módosítják, hogy azok valamilyen célértéket elérjenek, vagy azt megközelítsék. A genetikus operátorok végrehajtása valamilyen fitneszérték kiértékelése után történhet meg, így elérhető, hogy a jobb genetikai tulajdonságú egyedek vegyenek részt a szaporodásban.

Esetünkben a párhuzamos működésre tervezett *Algoritmus 2.* jellemezhető evolúciós jelzővel, jóllehet ez az algoritmustervezés mellékhatásként jelentkezik. Ilyen szempontból megvizsgálva a lineáris egyenletrendszer iteratív megoldása felfogható egy evolúciós optimalizáló feladatként. A futás során generált egyes közelítő vektorok (új egyedek) alkotják a közelítő megoldáshalmaz populációját. Az egyedek reziduum minimalizáló lépései felfoghatók fitnesz függvény kiértékelésként, ahol a kisebb reziduum normával rendelkező vektorok jelentik a jobb fitnesz értékeket. Az x^{12} vektor két populációbeli egyed (x^1, x^2) keresztezéséből előálló új egyed lesz. Az így előálló új egyedek a populáció részét képezik, a régi egyedek pedig kikerülnek a populációból. Az iterációs sorozatban a legjobb és a véletlen tulajdonságú egyedek sorozatos kombinációjaként előálló közelítő megoldásvektor az alapfeladat optimális megoldáshoz közelítő megoldását is jelenti.

Az *Algoritmus 2.* alkalmas lehet a fentieknél nagyobb méretű számítógépes hálózaton való párhuzamos futtatásra is. Nagyobb hálózatokon azonban fontos a hatékony erőforráskihasználás, amiből következik, hogy a *felt1* és *felt2* kifejezés nem tartalmazhat olyan részeket, amelyek függenek a csomópontokban található számítógépek egyedi teljesítményétől. Vagyis a feltételeket úgy kell megfogalmazni, hogy a különböző teljesítményű csomópontok lehetőleg kerüljék el a tétlenségi, várakozási, illetve torlódási helyzeteket. Ezt úgy lehet elérni, hogy a megoldandó feladat és a rendelkezésre álló hardver vizsgálatával a feltételes kifejezéseket optimalizáljuk.

A processzorszám növelésével további távolabbi csomópontokkal való kommunikáció érhető el (topológia kiterjesztés több belső élt tartalmazó gráfra). Ez a kommunikációs feltételek megfogalmazásánál további lehetőségeket jelent (bevezethető például csomópontonként harmadik kommunikációs kapcsolat). Ez azonban olyan mellékhatásokkal jár, mint a sorozatos várakozási helyzetek, vagy a kommunikációs csatorna betelése, amely az algoritmus összesített hatékonyságát jelentős mértékben rontja. Ezért az algoritmust más irányban érdemes tovább finomítani. Erről a következő alfejezetben írunk bővebben.

6.3.2. Realizáció homogén és heterogén számítógépes rendszeren

A párhuzamos algoritmus tesztelésekor a csomópontokat egyenrangúnak tekintjük. Ekkor minden egyes csomópont ugyanazon algoritmust futtatja. A futtatáshoz egyaránt alkalmasak a multiprocesszoros, közös memóriát használó gépek [Basermann et al. 97], mint a heterogén számítógépekből álló klaszterek. Ez utóbbi esetben a kiinduló adatok minden processzoron való inicializálása és a futás közbeni folyamatos üzenetváltás kommunikációs költségeit figyelembe kell venni, hiszen jelentősen lassítják a megoldás elérésének idejét.

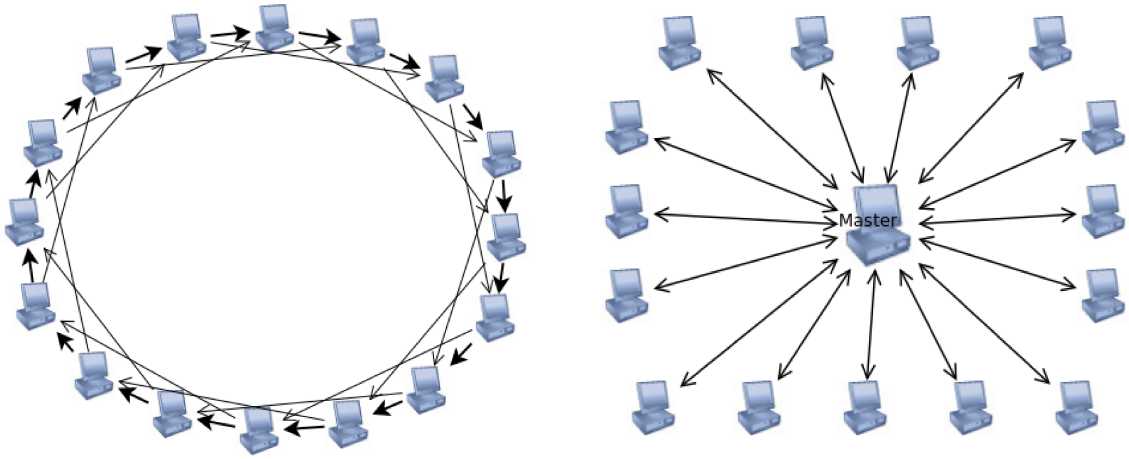
A fenti modell gyűrűs topológiára készült és alkalmas a gyűrűs, de akár a teljes gráffal leírt egyenrangú modellen való működésre. A futtatást egy 2-10 számítógépből álló heterogén klaszteren végeztük, melyet később az egyetem első *HP Blade System c3000-es* multiszámítógépén is megismételtünk [Varjasi 07].

6.4. Szinkron és aszinkron modell

A párhuzamos algoritmus futtatásának célja a hatékony és gyors feladat feldolgozás. Az egyenrangú feladatokkal végzett *Algoritmus 2.* futtatásakor megfigyelhető volt a gyűrűs topológiákon jelentkező periodikus – a kommunikáció szinkronizációjából fakadó – jelenség. Ebben az esetben az összes csomópont szinkronizáltan, azonos ütemben cserél adatokat, és azonos ütemben hajtja végre a számítási feladatokat. Ez homogén rendszerekben viszonylag jól kezelhető, hiszen a processzorok azonos teljesítményével és az „olcsó” kommunikációs költségek miatt ritkábban alakul ki üresjárat, és foglalva várakoztatás. Heterogén klasztereknél, vagy NUMA-rendszerű multiprocesszoros gépek alkotta hálózatokban viszont az algoritmus a leggyengébb processzor és a leglassabb hálózati kapcsolat

sebességére lassul. A futtatás során a gyorsabb processzorok jelentős teljesítmény-esése figyelhető meg.

Ezen jelenségek mellékhatásainak és a nagyobb feladatok hatékony megoldásának elérésére az algoritmust úgy kell átdolgozni, hogy abban a kommunikációs ütközések minimálisak legyenek. Ez valamilyen aszinkron modell segítségével írható le. Az újabb modellben a csomópontokat mester-szolga rendszerű egymélységű fa (csillag) struktúrára alakítjuk (*lásd 14. ábra*).



14. ábra: Gyűrűs (a) és hierarchikus (csillag) topológia(b) kommunikációs csatornái

A hierarchikus topológia esetén az egyik csomópont (processzor) a többitől elérő feladatot kap. Ez a végrehajtó egység a feladat részfeladatokra felosztásáért, menedzselésért és a kommunikáció vezérléséért felel. A feladat végrehajtását *p-1* szolga csomópont végzi. Az elkészült modell teljes informáltságú, minden egyes processzornál szükséges a teljes feladat tárolása. A mester-szolga kommunikáció működhet szinkron, vagy aszinkron üzenetváltással.

6.4.1. A mester-szolga rendszerű reziduum-minimalizációs algoritmus

Az algoritmus mester-szolga modellben az alábbi módosításokkal alkalmazható:

ALGORITMUS 3. PÁRHUZAMOS REZIDUUM-MINIMALIZÁLÁS, HIERARCHIKUS MODELL

1. Legyen a csomópontok halmaza *P*, a mester csomópont: p_m a szolga csomópontok $p \in P_w$, és legyen az elérni kívánt pontosság ε.
2. Legyen a feldolgozni kívánt mátrix *A*, és *b* az egyenletrendszer jobb oldala.
3. A mester csomópontban legyen x^1 egy véletlenszerűen választott vektor, *(* r^1 *a hozzá tartozó reziduum)* és küldd ki $\forall\, p \in P_w$ csomópontnak.
4. A mester csomóponton: *Vezérlés()*, amíg $\left\|r^1\right\|_2^2 < \varepsilon$
5. Párhuzamosan $\forall\, p \in P_w$ processzoron: *Művelet(*x^1*)*
6. A feladat megoldása: x^1 a mester csomóponton.
7. $\forall\, p \in P$ csomóponton: feladat vége.

Művelet(*x^1*)

i. x^2 véletlenvektor generálása

ii. $r^1 := Ax^1 - b$ és $r^2 := Ax^2 - b$, amelyre $r^1 - r^2 \neq 0$

iii. $c^{12} := \dfrac{\langle r^2 - r^1, r^2 \rangle}{\left\|r^1 - r^2\right\|_2^2}$

iv. $x^{12} := c^{12} \cdot x^1 + \left(1 - c^{12}\right) x^2$, $r^{12} := c^{12} \cdot r^1 + \left(1 - c^{12}\right) r^2$

v. $x^1 := x^{12}$ és $r^1 := r^{12}$

vi. *return* x^1, $\left\|r^1\right\|_2^2$ a mester csomópontnak.

Vezérlés()

I. Legyen x^s az eddigi legjobb megoldásvektor, r^s a hozzá tartozó reziduum.

II. *do*

III. x^i, $\left\|r^1\right\|_2^2$ vektor és norma fogadása $p_i \in P_w$ processzoroktól.

IV. *ha* $\left\|r^i\right\|_2^2 < \left\|r^s\right\|_2^2$ *akkor*

$x^s := x^i$,

új x^i vektor generálása, és küldése $p_i \in P_w$ processzornak.

V. *különben ha* $\left\|r^i\right\|_2^2=\left\|r^s\right\|_2^2$ *akkor*

új x^i vektor generálása, és küldése $p_i \in P_w$ processzornak.

VI. *különben* x^s vektor küldése $p_i \in P_w$ processzornak.

VII. *while* $\left\|r^s\right\|_2^2<\varepsilon$

Az átalakított algoritmus a reziduum minimalizáló algoritmust a *Művelet*(x^1) szubrutinba helyezi át. Az átalakítással azt érjük el, hogy *Algoritmus 2.*-hez képest elkülönülnek a hierarchiát leíró, a kommunikációs és az algoritmust vezérlő részfeladatok. A megoldással a szolga csomópontok egymástól függetlenül, csak a mesterrel kommunikálva hajthatják végre az egyes iterációs lépéseket, és az általuk számított megoldásvektorukat a mesternek továbbítják, majd új megoldandó iterációs lépésre várnak. A mester a beérkező közelítő megoldásvektorokat begyűjti, majd az addigi legkisebb hibával rendelkező megoldásvektort továbbküldi a szolga csomópontoknak.

Állítjuk, hogy a mester csomópont által vezérelt végrehajtás segítségével elérhető, hogy az algoritmus hatékonysága javuljon (*Algoritmus 2.*-hez képest)**, annak ellenére, hogy a gyűrűs topológiában eggyel több processzor vesz részt a számításban.**

A legjobb vektorok továbbküldésével az algoritmus evolúciós jellegét erősítjük: míg a gyűrűs topológiában egy jó megoldás csak lassan terjed tovább a populációban (a szomszédos csomópontok közvetítésével), addig a hierarchikus topológiában a mester használatával a soron következő adatcserével elérhető a javító hatás.

Ez a megoldás egy olyan társasjátékkal is szemléltethető, amelyben minden játékos a saját pályáján kockadobással jut előre. Azonban a lépéseket nem a saját régebbi pozíciójáról kezdi, hanem a versenyben éppen vezető játékos pozíciójáról léphet előre. A versenyt az első célba érkező nyeri úgy, hogy a pálya bejárását minden játékos dobása segíti. A játék elsődleges célja ugyanis nem egyetlen versenyző győzelme, hanem a minél gyorsabb célba érkezés.

6.5. Mérési eredmények

Az *Algoritmus 3.* számítógépes realizálását egy kisebb méretű (*p=20*) heterogén ethernet kapcsolt klaszteren és a *HP Blade System C3000* multiszámítógépen (elérhető processzorok száma: *p=88*) végeztük MPI szoftver környezetben. Az algoritmus megvalósításában a véletlen vektorok előállításához Mersenne-Twister [Matsumoto 98] álvéletlenszám generátort alkalmaztunk. Az egyes csomópontokban a véletlenszerűséget növeli a számítási csomópontok egyedi és független órája.

Feladat mérete	Kondíciószám
$n=500$	$cond=4\cdot10^{8}$
$n=1000$	$cond=2\cdot10^{10}$
$n=5000$	$cond=4\cdot10^{13}$
$n=10\,000$	$cond=7\cdot10^{12}$
$n=15\,000$	$cond=5\cdot10^{16}$

3. táblázat: A vizsgált lineáris egyenletrendszerek mérete és kondíciószámának nagyságrendje

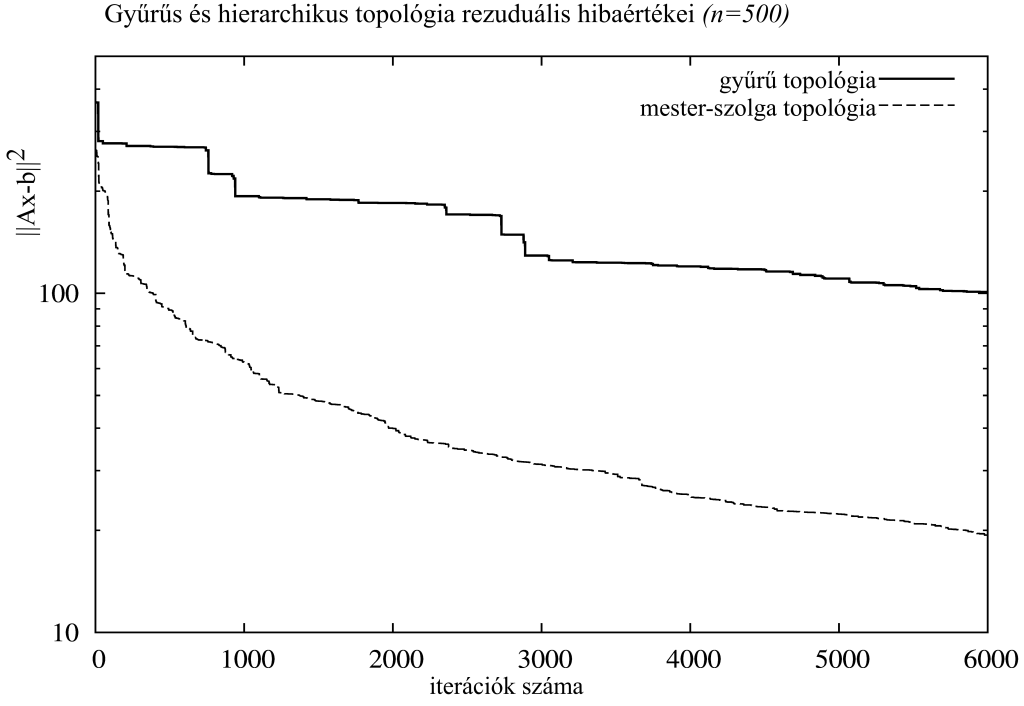
Az algoritmust a *3. táblázatban* látható feladatosztályokra teszteltük. A méréseket különböző számú processzor felhasználásával (*p=1..20,* illetve *p=1..88*), 20-20-szeres ismétléssel végeztük, majd a futási eredményeket átlagoltuk. Mivel az algoritmus véletlen működésen alapuló számításokat használ, ezért a futási eredmények ε hibaértéken belül pontosak. Továbbá itt is felhasználtuk azt az ellenőrző módszert, hogy ismert megoldású feladatokhoz kerestünk az algoritmus segítségével közelítő megoldásokat. (Megjegyzés: a bemutatott ábrák egy-egy konkrét futtatás eredményét tartalmazzák, így egyediek. Igyekeztünk a táblázat feladatosztályaiból válogatva több méreten is megmutatni az algoritmusok főbb jellegzetességeit és tendenciáit, ugyanis az elvégzett kísérletek alapján számos hasonló ábra generálható lenne.)

A feladat validálásához ezért két mérőszámot használunk [Kulish 08]. Az első a már előzőleg is használt általános reziduális hiba $err_r=\|r^1\|^2=\|Ax^1-b\|^2$. Amennyiben a feladat egzakt megoldása is ismert (*x'*), akkor meghatározható a megoldásvektor abszolút hibája is

$err_x = \|x^1 - x'\|^2$, ahol x^1 a közelítő megoldásvektor. Az ilyen méréseket ismert tesztesetek felhasználásával végeztük [MATRIXMARKET].

Az algoritmus alkalmas általános teljes rangú, nagyméretű lineáris egyenletrendszerek megoldására, melyeknek magas a kondíciószáma. A magas kondíciószám a megoldandó feladat bonyolultságát és megoldhatóságát jellemzi (*lásd 4.2. fejezet*). A számítógépes tesztekben olyan problémákat választottunk (*3. táblázat*), melyek nagyméretűek, rosszul kondicionáltak, és hagyományos direkt megoldó algoritmussal már nem [Gilbert 94], vagy nehezen oldhatók meg [Oishi et al. 08].

A *15. ábrán* megfigyelhetjük a mester-szolga viszony előnyeit. A mesternek jelentett legjobb megoldások között néha található egy-egy olyan javító lépés (lépcsőzetesség a grafikonon), amely az algoritmus evolúciós jellege miatt figyelhető meg: valamelyik szolga egy pontosabb megoldást talált. Ez a jobb megoldás ezután a többi csomóponthoz is el fog jutni, így a program a szekvenciális és a gyűrűs topológiánál bemutatottnál hatékonyabb lesz.



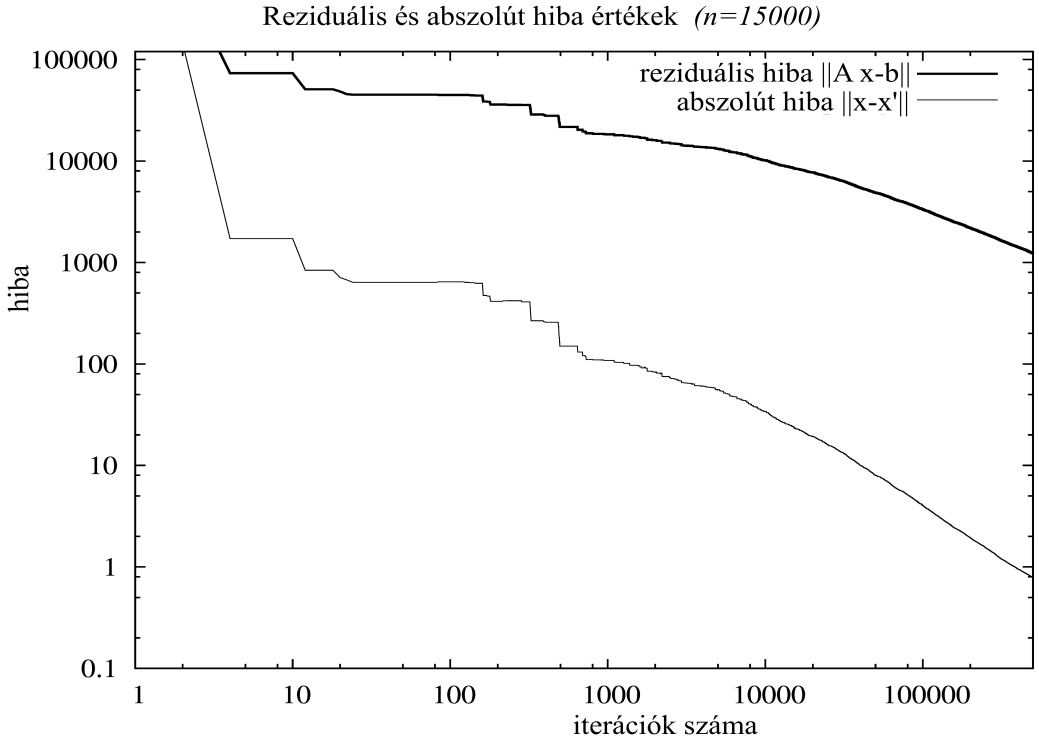
15. ábra: Algoritmus 2. és Algoritmus 3. reziduális hibájának konvergenciája gyűrűs, illetve hierarchikus topológián (n=500, p=16).

Az ilyen javító lépések nem határozhatók meg előre, előfordulásuk a nemdeterminisztikus működés következményei. Az ilyen gyorsító lépések előfordulása esetén a többi szolga

csomópont maximum egy „felesleges” iterációt hajthat végre, míg a gyűrűs modellnél akár *p-1* iteráció is szükséges lehet a javító lépés által szolgáltatott pontosabb eredményvektor elterjedésére.

A nagyméretű és rosszul kondicionált feladatoknál a reziduális hiba értéke relatív nagy lehet. Ilyenkor az egzakt megoldással összehasonlítva azonban azt tapasztaljuk, hogy az abszolút hiba már jóval pontosabb, akár a gépi számábrázolás közelébe is eshet. Az abszolút hiba nagyságrendje a következő eljárással számítható: $\|x-x'\|^2=\sum_{i=1}^{n}\|x_i-x_i'\|^2$

(Például ha *A* mátrix $cond=10^{16}$ kondíciószámú és a reziduális hiba $err_r=1000$ értékű, akkor a megoldás abszolút hiba értéke csupán $err_x=\|x_i-x_i'\|^2\approx 10^{-8}$ nagyságrendű.)



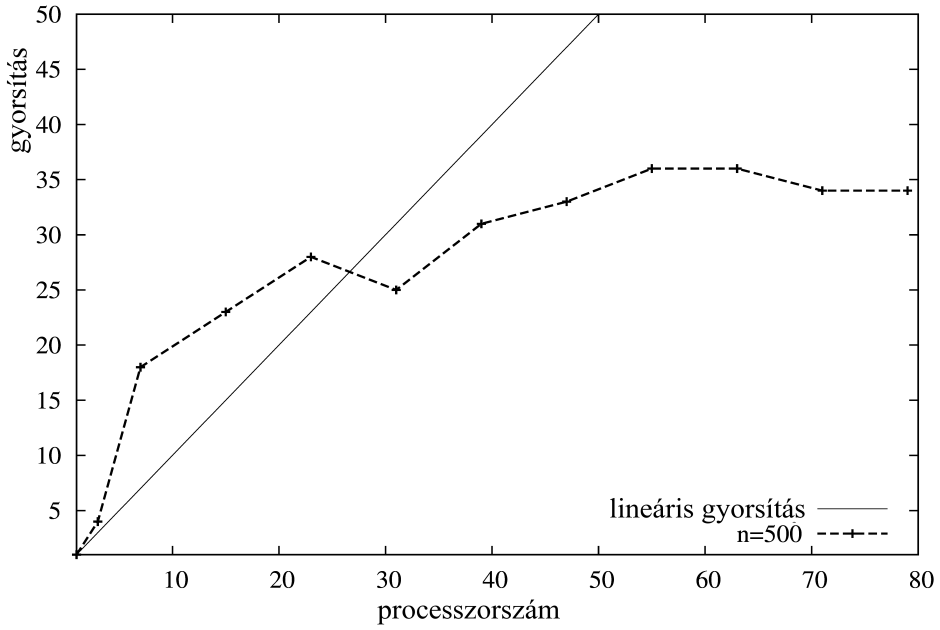
16. ábra: A megoldásvektorok reziduális és abszolút hiba értékei az iteráció során hierarchikus topológiánál (n=15 000, p=88, log-log skála)

Amennyiben a vizsgált feladat pontos megoldását összehasonlítjuk az algoritmus által az iteráció során kapott legjobb értékekkel, azt állíthatjuk, hogy a mesternél jelentkező legjobb megoldások abszolút hibaértéke mindig alacsonyabb, mint a reziduális hiba (*lásd: 16.ábra*).

Hierarchikus modellben az ilyen jó megoldások az ún. „soklépéses” iterációs jelleg miatt, a következő adatcserekor már eljutnak a többi csomóponthoz és azoknál a pontos megoldáshoz való közelítést gyorsítják.

A hierarchikus modellt használó *Algoritmus 3.*-ban több hatás is erősíti a nemdeterminisztikus jelleget: egyrészt a véletlen vektorok kombinálása, másrészt a hierarchián belüli kommunikáció aszinkron jellege. A vizsgálatok teljes, minden részletében azonos reprodukálhatósága ezért nem érhető el (még a véletlenszám generátorok rögzítésével sem). Ezért az azonos kiinduló feladatok megoldásai a választott ε pontosság mellett reprodukálhatók.

A hierarchikus algoritmus hatékony konvergenciáját a következő mérés is alátámasztja. A szekvenciális esettel összehasonlítva, a processzorszám emelésével az adatcserék száma emelkedik. Az így előálló megoldássorozat a hatékony véletlenszám-generálás miatt és az adatcserékben rejlő evolúciós jellegű algoritmus viselkedés előnyei miatt a processzorszám emelésével egyes tartományokban szuper-lineáris jelleget mutat (*17. ábra*) [Molnárka, Varjasi 11a].



17. ábra: Algoritmus 3. gyorsítási értékei (n=500, p=1..80)

Az ábrán egy konkrét feladat megoldása látható, de a többi vizsgált feladatosztályban is jelentkezett ez a hatás. Megállapítható, hogy a szuper-lineáris gyorsítás csak bizonyos esetekben, egy bizonyos processzorszámig tartható, utána a hagyományos Amdahl-féle

lefutási értéket kapjuk *(vö: 8. ábra)*. Sajnos az algoritmusnak ez egy jelentős hátránya, amely abból a hardveres megszorításból ered, hogy a kommunikációs csatorna nem bővíthető korlátlanul. Mivel a feladat teljesen informált és az adatcserék során a teljes x megoldásvektorokat p processzor között oda-vissza mozgatjuk, így a hálózati kapcsolat könnyen túlterhelhető.

Kisebb mértékben késleltethető ez a hatás azzal, ha szolga csomópontokban néhány iterációt végrehajtunk a mesternek jelentés nélkül, ekkor azonban le kell mondanunk a javító lépések azonnali tovaterjedéséről. Ezzel összekapcsolható az algoritmus evolúciós jellegének hangsúlyozása és hatékony kihasználása és a szolga oldalon az érkező megoldásvektor szórásának figyelembe vétele. Az ilyen szempontok szerinti módosítások alapján állíthatjuk elő a következő algoritmust.

ALGORITMUS 4. MÓDOSÍTOTT PÁRHUZAMOS REZIDUUM-MINIMALIZÁLÁS

1. Legyen a csomópontok halmaza *P*, a mester csomópont: p_m a szolga csomópontok $p \in P_w$, és legyen az elérni kívánt pontosság ε.
2. Legyen a feldolgozni kívánt mátrix *A*, és *b* az egyenletrendszer jobb oldala.
3. A mester csomópontban legyen $x = \{x_1^1, x_1^2, \ldots, x_1^g\}$ véletlenszerűen választott vektorok halmaza $g \in N$, $r = \{r_1^1, r_1^2, \ldots, r_1^g\}$ a reziduum vektorok halmaza. Legyen továbbá x_s a feladat kiindulási közelítő megoldása, r_s a hozzá tartozó reziduum.
4. A mester csomóponton: *Vezérlés()*, amíg $\|r_s\|_2^2 < \varepsilon$
5. Párhuzamosan $\forall p \in P_w$ processzoron: *Művelet*($\{x_1^1, x_1^2, \ldots, x_1^g\}$)
6. A feladat megoldása: x_s a mester csomóponton.
7. $\forall p \in P$ csomóponton: feladat vége.

Művelet($\{x_1^1, x_1^2, \ldots, x_1^g\}$)

i. Legyen $x_1 = x_1^1$ és $x_2 = \{x_1^2, x_1^3, \ldots, x_1^g, x_2^1, x_2^2, \ldots, x_2^g, x_2^{g+1}\}$ vektorhalmaz, melyből $\{x_2^1, x_2^2, \ldots, x_2^g, x_2^{g+1}\}$ véletlenszerűen generált vektorok.

ii. *ciklus* $i = 1$ *-től* $2g$ *-ig*

iii. $r_1 := Ax_1 - b$ és $r_2^i := Ax_2^i - b$, amelyre $r_1 - r_2^i \neq 0$

iv. $c^{12} := \dfrac{\langle r_2^i - r_1, r_2^i \rangle}{\|r_1 - r_2^i\|_2^2}$

v. $x^{12} := c^{12} \cdot x_1 + (1 - c^{12}) x_2^i$, $r^{12} := c^{12} \cdot r_1 + (1 - c^{12}) r_2^i$

vi. $x_1 := x^{12}$ és $r_1 := r^{12}$

vii. *ciklus vége*

viii. *return* x_1 és $\|r_1\|_2^2$ a mester csomópontnak.

Vezérlés()

I. Legyen x_s az eddigi legjobb megoldásvektor.

II. *do*

III. x_1 és $\|r_1\|_2^2$ vektor fogadása $p_i \in P_w$ processzoroktól.

IV. *ha* $\|r_1\|_2^2 < \|r_s\|_2^2$ *akkor* $x_s := x_1$

V. *különben* $x_1^g := x_1$, $r_1^g := r_1$

VI. $x = \{x_1^1, x_1^2, \ldots, x_1^g\}$ vektorhalmazt rendezd $\|r_1^i\|_2^2$ szerint.

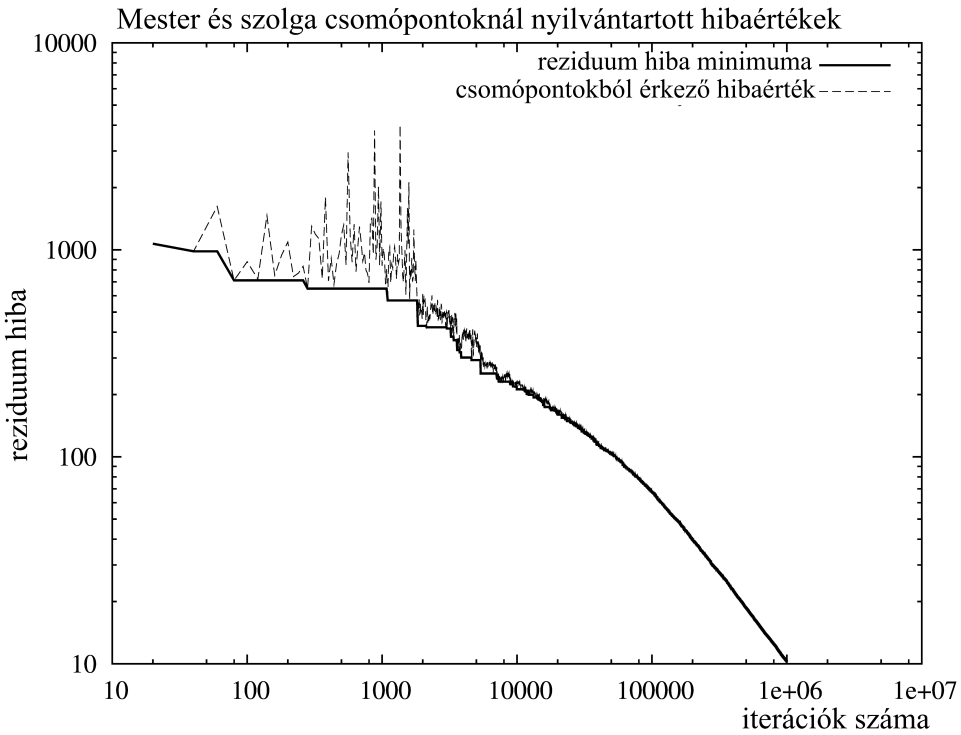
VII. $x = \{x_1^1, x_1^2, \ldots, x_1^g\}$ vektorhalmaz küldése $p_i \in P_w$ processzornak.

VIII. *while* $\|r_s\|_2^2 < \varepsilon$.

Az *Algoritmus 4.* az *Algoritmus 3.* olyan átalakításán alapul, amelyben a mester g darab – reziduális hiba szerint sorba rendezett – megoldásvektort tárol. A mester az egyes csomópontokkal való kommunikáció során ezt a vektorsereget (vektorok halmazát) továbbítja. A szolga csomópontok, miután a vektorsereget megkapták, további g darab véletlenszerű vektort generálnak, egymástól függetlenül. Az így előálló $2g$ darab vektoron minden szolga elvégez $2g$ darab iterációs lépést. Legvégül az előálló legjobb megoldásvektort, és a hozzá tartozó reziduum vektort visszaküldi a mester csomópontnak. Az g változó értéke empirikus érték, azzal a kikötéssel, hogy $g \ll p$. (A vizsgált példákban $g = 2..4$ értékeket használtunk, de kvantitatív optimalizációt erre a változóra nem végeztünk.) Ezzel a megoldással az evolú-

ciós jelleg is erősíthető, hiszen a több megoldásvektor rendezett használata a populációban *elitizmust* jelent. Túl magas *g* érték választásával pedig a véletlen vektorok által szolgáltatott javító hatást vesztjük el.

Az *Algoritmus 4.* módosításaival elérhető, hogy a futtatás során kisebb méretű adatcsere, a csomópontokban nagyobb „cpu" terhelés jelentkezzen. A feladatot az ún. soklépéses iterációval hajtjuk végre, amelyben az egyes lépéseket egymástól független irányokból közelítjük. A szolga csomópontoknál azt a megfigyelést használhatjuk ki, hogy az *Algoritmus 1.*-ben leírt szekvenciális algoritmus konvergenciája stabil, és akkor is megmarad, ha azt a megoldástól „távoli" vektorból indítjuk [Molnárka 06b]. Ilyen esetben az iterációszám növekedésével ugyan, de ugyanolyan hibaosztályú megoldás érhető el. A számítógépes tesztek ezt a viselkedést alátámasztják.



18. ábra: Egy iteráció sorozatban a legjobb megoldások (folytonos) és a szolgák által közölt megoldásvektorok (szaggatott) reziduális hibája (n=10 000, g=4, log-log skála)

A *18.* ábráról leolvashatjuk, hogy a szolga csomópontokból – szaggatott vonallal jelölt – hibaértékek érkeznek. A mester csomópontban a legjobb megoldás hibaértékét a folytonos vonal jelzi. Megfigyelhető, hogy a szolgáktól kezdetben ugyan érkeznek kevésbé jó

megoldások, ám az iteráció előrehaladtával ez a különbség minimálisra csökken, az algoritmusba épített javító hatás érvényesül.

A szolga csomópontokban továbbá a reziduum minimalizáló algoritmus előzőekben kifejtett stabil tulajdonságát más szempontból is kihasználhatjuk. Az egyes szolgáknál az érkező megoldásvektorok által kifeszített értékkészletből generálunk megoldásvektorokat. Vagyis a véletlen szám generáló szubrutint úgy módosítjuk, hogy minden adatfogadáskor végrehajtjuk a vektorok értékkészletének mérését. A generátor az előálló intervallumból szolgáltat új vektorokat. Ez ugyan minden szolga csomópontban növeli a műveletszámot, de a megfigyelések *(18. ábra)* alapján a befektetett munka gyorsabb konvergenciával megtérül. A „sejtett" intervallumból generált vektorok segítségével az iterációs lépésekben gyors közelítés érhető el. Az ilyen lépések – szolga csomópontban való – átalakítása a globális keresést nem rontja el, hiszen a mester csomópont csak a relatív hiba alapján rangsorolja a megoldásvektorokat.

Az *Algoritmus 4.* futtatásait ez előző algoritmussal megegyező feladatokon ismételten elvégeztük. A mérések során azt tapasztaltuk, hogy a feladatok futási ideje kisebb mértékben tovább javítható [Molnárka, Varjasi 10], de az *Algoritmus 2. – Algoritmus 3.– Algoritmus 4.* reziduum-minimalizáló algoritmusok jellemzően lassú konvergenciája az „elég jó" pontosságú megoldás eléréséhez továbbra is megmarad.

Összességében a reziduum-minimalizáción alapuló párhuzamos algoritmusokról elmondhatjuk, hogy a számítógépes tesztek alátámasztották az elméleti feltevésünket, és alkalmasak sokprocesszoros gépen való futtatásra. Az algoritmus finomhangolásával a futtatás az adott architektúrához igazítható, és a konvergencia sebessége javítható. A vizsgált topológiák közül a heterogén rendszerekre szabott, hierarchikus mester-szolga rendszerű algoritmusok – az egyel kevesebb végrehajtó egység ellenére – gyorsabb időbeli lefutást biztosítanak.

A véletlen irányokból közelítés módszere önmagában nem hatékony. A bemutatott algoritmusok a legkisebb négyzetek módszerén alapuló minimalizáló algoritmusnak olyan verziói, amelyek konvergencia tulajdonsága a *4.2.1. fejezetben* tárgyalt lineáris egyenletrendszert megoldó algoritmusok legrosszabbjai közé tartoznak. Ezért az ott ismertetett szekvenciális, illetve párhuzamos megoldó algoritmusokkal eredményeinket nem is vetettük össze.

Az általunk ismertetett algoritmusok egyetlen dologban azért különböznek az ismert algoritmusoktól: lehetővé teszik azt, hogy tetszőleges számú, független processzt indítsunk el úgy, hogy a részprocesszek eredményei kölcsönösen egymást javítsák. Az ismert algoritmusokon történő kis módosítás szekvenciális értelemben a konvergenciát nem javítja, de párhuzamos környezetben nagy szabadságot ad az algoritmus szempontjából hasznos műveletek elvégzésének párhuzamosítására.

7. ALTÉR DEKOMPOZÍCIÓT HASZNÁLÓ ALGORITMUSOK

Az előző fejezetben tárgyalt reziduum-minimalizáción alapuló párhuzamos algoritmusok – a feladat méretével arányosan növő kommunikációs költségek miatt – a sokprocesszoros futtatásra jó gyorsítással csak egy bizonyos processzorszámig alkalmazhatóak. Mivel az algoritmus – numerikus matematikai jellegéből fakadóan – teljesen informált rendszert feltételez, ezért a futás közbeni nagy kommunikációs igény miatt újabb algoritmusokkal kezdtünk el foglalkozni.

Az alapötletet az ún. altér dekompozíciós modell jelenti. Ebben azt a jelenséget használjuk ki, hogy a nagyméretű alapfeladatokat az alább ismertetett modell szerint *speciális alterekre*[3], azaz nagyságrendekkel kisebb méretű – így hagyományos direkt, vagy iteratív módszerekkel könnyen megoldható – problémák sorozatára bontjuk. Az így előálló iteráció egy jól párhuzamosítható, jól skálázódó algoritmus, amely alkalmas a nehezen megoldható, rosszul kondicionált feladatok megoldására. A modell használatával a részekre bontás kisebb kondíciószámú feladatok egymástól független feldolgozására vezethető vissza.

7.1. Az altér dekompozíciós modell

Az eredeti $Ax=b$ lineáris egyenletrendszer megoldásához ($n\times n$ alakú valós *A* mátrix és valós *b* vektor esetén) a legtöbb iteratív algoritmus (*lásd 4.2.1. fejezet*) az eredeti probléma megoldása helyett az alábbi normálegyenlet megoldását tűzi ki célul:

$$A^T Ax = A^T b = v \tag{12}$$

A normálegyenlet használatával egy teljes rangú szimmetrikus feladatot kapunk, ám az $A^T A$ szorzat mellékhatásaként az megoldandó feladat kondíciószáma az eredeti feladat négyzete lesz, így az iteráció sebessége lassú lesz. Ez a jelenség megfigyelhető a GMRES, BCG vagy QMR algoritmusok esetén [Ciarlet 06].

A legkisebb négyzetek módszere segítségével a normálegyenlet az alábbi formula minimalizálásával oldható meg:

3 *Megjegyzés:* Az altér dekompozíció kifejezést mi nem a klasszikus értelemben használjuk, ugyanis a különböző alterek kiválasztása algoritmusainkban a reziduum vektorok terében történik, és azt a megkötést sem használjuk, hogy az alterek direktösszege kiadja a teljes teret, azaz az alterek diszjunkt és teljes jellege nincs kikötve.

$$\min_{x\in\mathbb{R}^n}\|Ax-b\|_2^2=\min_{x\in\mathbb{R}^n}\langle Ax-b\,,\,Ax-b\rangle=\min_{x\in\mathbb{R}^n}\langle r\,,\,r\rangle \tag{13}$$

ahol $r=Ax-b$ jelöli az *x*-hez tartozó reziduum vektor.

A minimumfeladat megoldásához tekintsük az alábbi jelölésrendszert és tételt [Molnárka 06a].

Tétel 2.:

Legyen $A\in\mathbb{R}^n\to\mathbb{R}^n$, és $b\in\mathbb{R}^n$ a megoldandó lineáris egyenletrendszer mátrixa és eredményvektora. Vegyük rendre az $x^\alpha\in\mathbb{R}^n,\alpha=1,2,\ldots,k$ véletlen értékekkel rendelkező vektorok halhazát (vektorsereg), amelyekre igaz, hogy $A(x^\alpha-x^\beta)\neq 0, ha\,\alpha\neq\beta$, és ahol k egy véletlenszerűen választott egész szám $k\in\mathbb{N}$, melyre igaz, hogy $k<n$.

A vektorsereghez tartozó reziduum vektorok legyenek rendre $r^\alpha=Ax^\alpha-b,\alpha=1,2,\ldots,k$, és tekintsük az alábbi – iterációs lépésként használandó – összefüggést:

$$\tilde{x}^s:=c_1x^1+c_2x^2+\ldots c_{k-1}x^{k-1}+(1-c_1-c_2-\ldots-c_k)x^k \tag{14}$$

és a hozzá tartozó reziduumok:

$$\tilde{r}^s:=c_1r^1+c_2r^2+\ldots c_{k-1}r^{k-1}+(1-c_1-c_2-\ldots-c_k)r^k \tag{15}$$

ahol $c_\alpha\in\mathbb{R},\alpha=1,2,\ldots,k$ a felhasznált korrekciós vektor. A fentiekből könnyen belátható, hogy a reziduum $\tilde{r}^s=A\tilde{x}^s-b$. A fenti jelölésrendszer segítségével a (13) egyenlet megoldása az $x=\{x^1,x^2,\ldots,x^k\}$ vektorok által kifeszített altérben megoldható. Amennyiben a vektorsereg reziduumai $r=\{r^1,r^2,\ldots,r^k\}$ lineárisan függetlenek, akkor a megoldás létezik és egyértelmű, amely az $\tilde{x}^s(c)$ vektor alakban állítható elő.

A képletben szereplő korrekciós vektor $c=[c_1,c_2,\ldots,c_k]$, amely az alábbi k-dimenziós lineáris egyenletrendszer megoldása:

$$Bc=d \tag{16}$$

ahol a B mátrix az alábbi alakban állítható elő:

$$B=\begin{bmatrix} \|r^1-\tilde{r}^s\|^2 & \langle \tilde{r}^s-r^1,\tilde{r}^s-r^2\rangle & \dots & \langle \tilde{r}^s-r^1,\tilde{r}^s-r^k\rangle \\ \langle \tilde{r}^s-r^2,\tilde{r}^s-r^1\rangle & \|r^2-\tilde{r}^s\|^2 & \dots & \langle \tilde{r}^s-r^2,\tilde{r}^s-r^k\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \tilde{r}^s-r^k,\tilde{r}^s-r^1\rangle & \langle \tilde{r}^s-r^k,\tilde{r}^s-r^2\rangle & \dots & \|r^k-\tilde{r}^s\|^2 \end{bmatrix} \tag{17}$$

és

$$d=\left[\langle \tilde{r}^s-r^1,\tilde{r}^s\rangle,\langle \tilde{r}^s-r^2,\tilde{r}^s\rangle\dots\langle \tilde{r}^s-r^k,\tilde{r}^s\rangle\right]^T \tag{18}$$

továbbá

$$\|\tilde{r}^s\|<\min_{1,2\dots k}\{\|r^1\|,\|r^2\|,\dots,\|r^k\|\}. \tag{19}$$

Az összefüggésekben minden norma jelölés euklideszi normát jelent. A tétel bizonyítása megtalálható [Molnárka 06b]-ben.

Megjegyzés: Az előzőekben definiált $\tilde{x}^s$ jelölésben a hullám jelzés arra utal, hogy az iterációs lépésekben egyre újabb, pontosabb közelítő vektorokat generálunk.

7.2. Véletlen irányokból való közelítés

A klasszikus iteratív algoritmusokkal ellentétben – gradiens, konjugált gradiens módszerek – a bemutatott módszer a minimalizációhoz tartozó vektorokat véletlenszerűen választja. Az iteráció során minden egyes lépés folytatható egy újabb független x^β vektorsereggel. A módszer újdonsága a véletlen irányokból való közelítés. Az ilyen véletlen vektorok párhuzamos rendszerekben könnyen generálhatók. Ezért célul tűztük ki a hatékony és jól párhuzamosítható algoritmusok definiálását a fenti tétel alapján.

A megfogalmazott tétel többféle módon realizálható. Először a szekvenciális algoritmust adjuk meg. Az algoritmusban felhasználjuk, hogy a közelítő vektorok halmazát (sorozatát) $x=\{x^1,x^2,\dots,x^k\}$ jelöli, és az ezekhez tartozó reziduumok $r=\{r^1,r^2,\dots,r^k\}$, ahol $k\in\mathbb{N}, k\ll n$. A jelölésben minden egyes x^i vektor *n*-dimenziós valós vektor, ahol *n* az eredeti $Ax=b$ egyenletrendszer rangja.

ALGORITMUS 5. ALTÉR-DEKOMPOZÍCIÓS SZEKVENCIÁLIS MÓDSZER

1. Legyen ε az elérni kívánt megoldási pontosság, *A* a megoldandó egyenletrendszer mátrixa és *b* legyen az egyenletrendszer jobboldala.

2. Legyen *k* a dekompozíció altérének dimenziója, $k \in \mathbb{N}$ és $k \ll n$.
3. Legyen x^s egy véletlenszerűen választott megoldásvektor, r^s a hozzá tartozó reziduum, ahol $r^s = Ax^s - b$.
4. Legyen $x = \{x^1, x^2, \ldots, x^k\}$ véletlenszerű vektorsereg, és a hozzá tartozó reziduumok $r = \{r^1, r^2, \ldots, r^k\}$, $i = 1..\mathrm{k}$.
5. Határozzuk meg *B* mátrixot a reziduumok különbségének skalárszorzatával úgy, hogy $B^{i,j} = \langle r^s - r^i, r^s - r^j \rangle$, $i, j = 1..\mathrm{k}$, és a d vektort: $d^i = \langle r^s - r^i, r^s \rangle$, $i = 1..\mathrm{k}$ -ra.
6. Oldjuk meg $Bc = d$ feladatot tetszőleges módszerrel.
7. Amennyiben megoldása nem létezik, generáljunk új $x = \{x^1, x^2, \ldots, x^k\}$ vektorsereget és ismételjük a 4-5 lépéseket, amíg a 6. lépés megoldható lesz.
8. Számítsunk új x^s és r^s vektorokat az alábbi formulával:

   $x^s := c_1 x^1 + c_2 x^2 + \ldots c_{k-1} x^{k-1} + (1 - c_1 - c_2 - \ldots - c_k) x^k$,

   $r^s := c_1 r^1 + c_2 r^2 + \ldots c_{k-1} r^{k-1} + (1 - c_1 - c_2 - \ldots - c_k) r^k$.
9. Ha $\|r^s\|_2 > \varepsilon$ ismételd az algoritmust a 4. lépéstől, amíg a feltétel nem teljesül.
10. A feladat közelítő megoldása x^s .

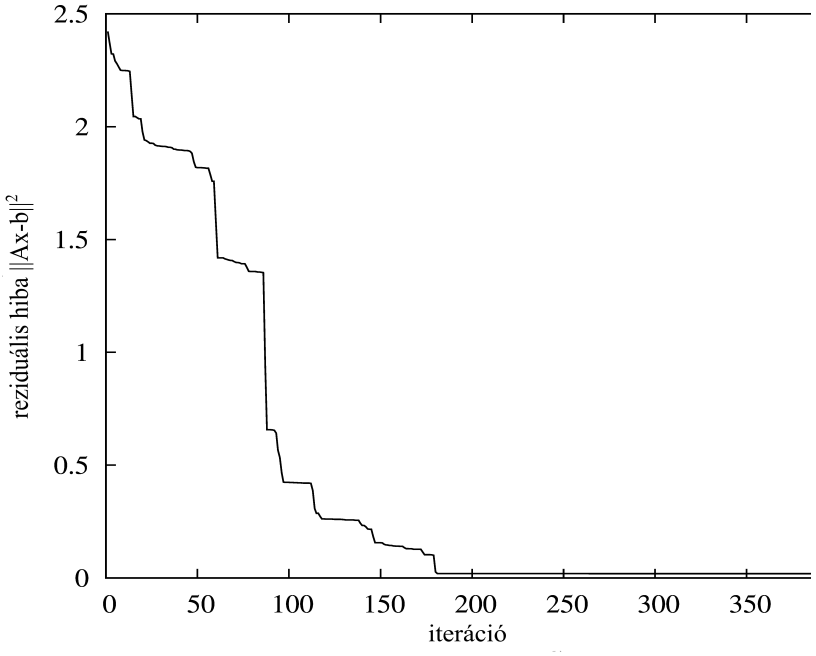
Megjegyzés:

- A *6.* lépésben bármelyik, a *4.2. fejezetben* felsorolt direkt, vagy iteratív algoritmus (LU, Gauss-Jordan, stb.) felhasználható.
- Az algoritmus segítségével a klasszikus iteratív algoritmusokhoz hasonló műveletigényű megoldás készíthető: az algoritmus nem ilyen szempontból jelentős, az újdonság és párhuzamosítás szempontjából sokkal fontosabb tulajdonság, hogy itt a a független, véletlenszerűen választott irányok módszerével dolgozik az algoritmus. Míg a klasszikus iteratív módszerekben a következő lépés minden esetben az előző lépésektől függ, ebben a módszerben ilyen függés nincs beépítve.
- Párhuzamos modellre alakítva az algoritmustól jobb és hatékonyabb konvergenciát várunk.

- A kutatások és a konferencia diszkussziók során ráirányult a figyelem a *B* mátrix kondíciószám nagyságára. Az általános értelemben vett altér felbontások során a teljes dekompozícióval a kondíciószám nem csökkenthető [Stoyan 02]. Esetünkben azonban az alterek képzése a reziduum vektorok terében történik. Így a fenti algoritmusnál *k* értékének bizonyos eseteiben *B* mátrix kondíciószám csökkenését várjuk.
- Megjegyezzük, hogy a fenti *Algoritmus 5.* $k=2$ szélsőérték melletti alakja visszavezet az *Algoritmus 1.* egyszerűsített megfogalmazására.

7.3. Mérési eredmények

A szekvenciális algoritmus konvergenciájának számítógépes ellenőrzésekor a numerikus módszerekben általánosan használt Matlab/Octave rendszert alkalmaztunk [Hanselman 01].

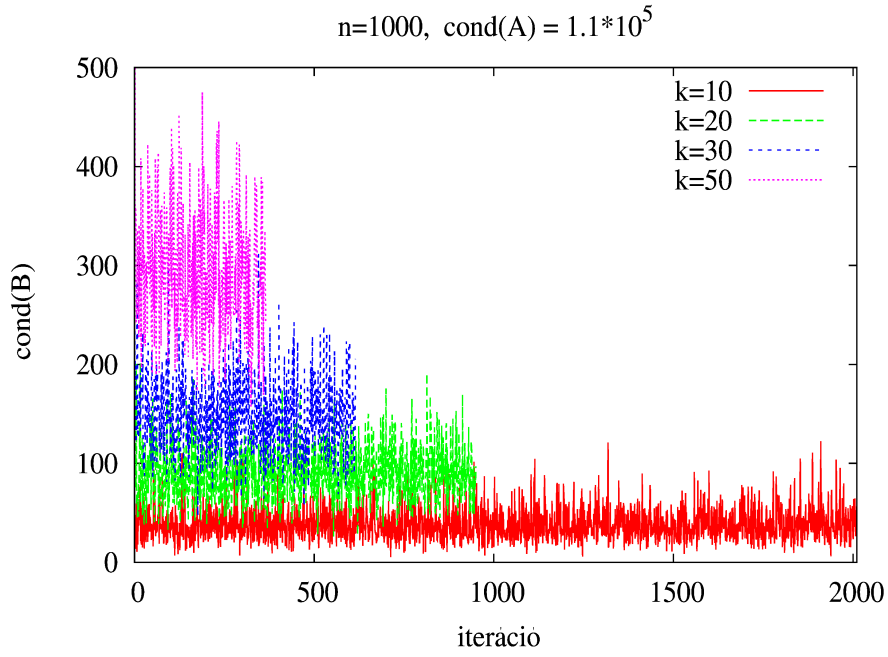


19. ábra: Az Algoritmus 5. konvergenciája szekvenciális esetben (n=100, k=10)

Az algoritmusban kétféle hibamérési módszert használtunk: az első az általános reziduális hiba, amelyet az algoritmusban is többször felhasználtunk $err_r = \|Ax^s - b\| = \|r^s\|$, ahol x^s a közelítő megoldás vektora. Amennyiben a pontos eredmény ismert *(x')*, akkor a számítás abszolút hibája is számítható : $err_x = \|x^s - x'\|$.

Az algoritmusban szereplő altér dekompozícióval előálló *k*-rangú $Bc=d$ lineáris egyenletrendszer megoldásához egy általános LU dekompozíciót számító algoritmust használtunk. Az LU módszer számítási költsége $O(k^3)$ méretű. Az *Algoritmus 5.* összesített műveletigénye így $O(\xi \cdot k^3)$ alakban határozható meg, ahol ξ az algoritmusban felhasznált iterációk számát jelöli ($\xi \sim n$).

Az *Algoritmus 5.* futtatásakor a *19. ábrán* látható monoton csökkenő hibaértékeket kaptunk eredményül (*n=100, k=10* paraméterek esetén). Az algoritmust nagyobb méretű feladatokkal is teszteltük (*n=500; 1000;* egészen a szoftver meglehetősen alacsony memóriakorlátjáig) és a futási idők a várt módon emelkedtek egyetlen processzor (processzormag) használatával.



20. ábra: B mátrix kondíciószáma az iteráció során, különböző altér méreteknél

A *B* mátrix kondíciószámának alakulása jelentős a felbontás megoldhatósága szempontjából (*lásd: 4.1. fejezet*). A *B* mátrix kondíciószámának becslésére a szekvenciális algoritmus MATLAB környezetben való ellenőrzésekor és validitásának vizsgálatakor sort kerítettünk. A numerikus kísérletek azt mutatják, hogy általános feladatokra *B* kondíciószáma nagyságrendekkel kisebb, mint az eredeti feladaté $cond(B) \ll cond(A)$. Kvantitatív elemzést ugyan nem végeztünk, de a numerikus kísérletekben azt tapasztaltuk – és hipotézisként feltehetjük –, hogy a *B* mátrix kondíciószáma az *n* értékétől nem, de az *A* kondíciószámától függ (*lásd 20. ábra*). Nagyságát elsősorban az altér mérete (*k*) befolyásolja. (Ha az altér mérete *n*, akkor

B kondíciószáma az *A* kondíciószámának megfelelő nagyságrendű.) Mérési tapasztalataink szerint *B* kondíciószáma adott *k* mellett csak kicsit változik a különböző iterációs lépésekben. Az extra nagy kondíciószámú *A* mátrixok esetén hasonló viselkedés tapasztalható.

7.4. Párhuzamosítási lehetőségek

Az altér dekompozíciós modellt és az *Algoritmus 5.* lépéseit felhasználva elkészíthetünk egy hierarchikus topológián futtatható párhuzamos algoritmust. Az *6.4. fejezetben* leírt eredmények tükrében ezt a párhuzamosítást gyűrű topológián nem végeztük el, jóllehet az algoritmus definiálása egyszerűbb lett volna, a futási eredmények azonban az ott ismertetett jelenségek hatására elmaradnának a hierarchikus modellétől [Bosilca et al. 12].

Azonos jelöléseket használva, egy p processzort használó rendszerben *p-1* szolga és egy mester csomópontra bontjuk a feladatot. A jelölésben felhívjuk a figyelmet arra, hogy a vektorseregben az x^i egy-egy, az eredeti feladat méretének megfelelő $n = rang(A)$ dimenziós vektor.

ALGORITMUS 6. PÁRHUZAMOS ALTÉR-DEKOMPOZÍCIÓS MODELL

1. Legyen ε az elérni kívánt megoldási pontosság, *A* a feladatot leíró mátrix és *b* az eredményvektor.
2. Legyen *k* az altér dekompozíció dimenziója, $k \in \mathbb{N}$ és $k < n$.
3. A mester csomópontban legyen x^s egy véletlenszerűen választott megoldásvektor, r^s a reziduum, ahol $r^s = Ax^s - b$.
4. A szolga csomópontokban $\forall p \in P_w$ ***Altér-művelet(*** x^s ***)*** párhuzamosan.
5. A mester csomópontban:

   ha $\|r^s\|_2 < \varepsilon$ *akkor* lépj a 7. pontra

   különben ha az érkező x^s pontosabb a tároltnál ($\|r^s_{érkező}\|_2 < \|r^s_{tárolt}\|_2$),

   akkor az érkező új x^s vektort tárold, és küldd szét a szolgáknak.
6. lépj a 4. lépésre és ismételj, amíg a sorozat konvergens.
7. A közelítő megoldás: x^s, algoritmus vége.

Altér-művelet(x^s ***)***

a) Legyen $x=\{x^1, x^2, \ldots, x^k\}$ véletlenszerű vektorsereg, és a hozzá tartozó reziduumok $r=\{r^1, r^2, \ldots, r^k\}$, ahol $r^i=Ax^i-b$ és $i=1..\mathrm{k}$.

b) Számítsd ki a *B* mátrixot $B^{i,j}=\langle r^s-r^i, r^s-r^j\rangle$, $i, j=1..\mathrm{k}$ és a *d* vektor értékét, amely $d^i=\langle r^s-r^i, r^s\rangle$, $i=1..\mathrm{k}$.

c) Oldjuk meg $Bc=d$ feladatot tetszőleges módszerrel.

d) Ha nem megoldható készítsünk $x=\{x^1, x^2, \ldots, x^k\}$ vektorsereget és ismételjük az *a)-c)* lépéseket, amíg a *c)* lépés megoldható lesz

e) Generáljunk új x^s és r^s vektorokat:

$x^s:=c_1x^1+c_2x^2+\ldots c_{k-1}x^{k-1}+(1-c_1-c_2-\ldots-c_k)x^k$,

$r^s:=c_1r^1+c_2r^2+\ldots c_{k-1}r^{k-1}+(1-c_1-c_2-\ldots-c_k)r^k$.

f) Küldd el a számított x^s vektort és $\|r^s\|_2$ hibát a mester csomópontnak.

A párhuzamos algoritmus alkalmas homogén és heterogén rendszereken is hatékonyan működni. A feladat végrehajtása a kezdeti inicializálás után (teljes informáltságú feladat) a mester és szolga csomópontok közötti *n*-dimenziós vektorok cseréjével működik.

A csomópontokhoz választott *k*-dimenziós alterek választásakor a feladat végrehajtható minden szolga csomópontra azonos (homogén rendszerekben), vagy különböző (heterogén rendszerekben) *k* értékekkel is.

7.5. A párhuzamos algoritmus korrekciója

Az altérben végzett művelet kisebb módosításával (keresési tér szűkítés) és a mester csomópontban a reziduális hibaérték szerint rendezett vektorseregek (halmaz) használatával az algoritmus hatékonysága tovább javítható.

ALGORITMUS 7. MÓDOSÍTOTT PÁRHUZAMOS ALTÉR-DEKOMPOZÍCIÓS MODELL

1. Legyen ε az elérni kívánt megoldási pontosság, *A* a feladatot leíró mátrix és *b* az eredményvektor.
2. Legyen *k* az altér dekompozíció dimenziója, $k\in\mathbb{N}$ és $k\ll n$.

3. A mester csomópontban legyen $x^s_{appr}=\{x^s_1, x^s_2, \dots, x^s_g\}$, $g \in \mathbb{N}$, $g \ll k$ a legjobb megoldásvektorok halmaza, x^s az érkező megoldásvektor, r^s a reziduum.
4. A szolga csomópontokban $\forall\, p \in P_w$ ***Altér-művelet(*** x^s_{appr} ***)*** párhuzamosan.
5. A mester csomópontban:
   - *ha* az érkező $\|r^s\|_2 < \varepsilon$ *akkor* tovább a *7.* lépésre
   - *különben ha* az érkező x^s jobb a tároltaknál ($\|r^s\|_2 < \|r^s_i\|_2$), *akkor* illeszd be az érkező vektort $x^s_{appr}(i), i = 1..g$ helyén.
6. Lépj a 4. lépésre és ismételj, amíg a sorozat konvergens.
7. A közelítő megoldás: x^s, algoritmus vége.

Altér-művelet(x^s_{appr} ***)***

a) Legyen $x=\{x^1, x^2, \dots, x^g, x^{g+1}, \dots, x^k\}$ egy olyan vektorsereg, amelyből *g* db paraméterként kapott, *k-g* db véletlenszerűen generált $x^i \in [min(x^s_{appr}), max(x^s_{appr})]$ és a hozzá tartozó reziduum vektorok $r=\{r^1, r^2, \dots, r^k\}$, ahol $r^i = Ax^i - b$ és $i = 1..k$.

b) Számítsd ki a *B* mátrixot $B^{i,j} = \langle r^s - r^i, r^s - r^j \rangle$, $i, j = 1..k$ és a *d* vektor értékét, amely $d^i = \langle r^s - r^i, r^s \rangle$, $i = 1..k$.

c) Oldjuk meg $Bc = d$ feladatot tetszőleges módszerrel.

d) Ha nem megoldható készítsünk $x=\{x^{g+1}, \dots, x^k\}$ véletlen vektorsereget és ismételjük az *a)-c)* lépéseket, amíg a *c)* lépés megoldható lesz

e) Generáljunk új x^s és r^s vektorokat:

$$x^s := c_1 x^1 + c_2 x^2 + \dots c_{k-1} x^{k-1} + (1 - c_1 - c_2 - \dots - c_k) x^k,$$

$$r^s := c_1 r^1 + c_2 r^2 + \dots c_{k-1} r^{k-1} + (1 - c_1 - c_2 - \dots - c_k) r^k.$$

f) Küldd el a számított x^s vektort és $\|r^s\|_2$ hibát a mester csomópontnak.

Az *Algoritmus 6.*-hoz képest két módosítás történt, a mester csomópont *5.* és az *Altér-művelet a)* pontjában. A vektorsereg használatával az algoritmus evolúciós jellegét erősíthetjük. A kommunikációs lépések során az addig *g* darab legjobb vektor továbbküldésével a szolgáknak csak *g-k* darab véletlen vektort kell generálniuk az egyes iterációs lépésekben. A *g*

meghatározásában figyelembe kell venni, hogy *g* jóval kisebb legyen *k*-nál, hiszen közel azonos értéknél elvesznének a független párhuzamosításból adódó gyorsító hatások, azaz kisebb eséllyel érhető el a szuperlineáris gyorsítás. (A vizsgálat során *g* értékét a [2, 4] intervallumból vettük, *g*-re külön nem optimalizáltuk a feladatot.)

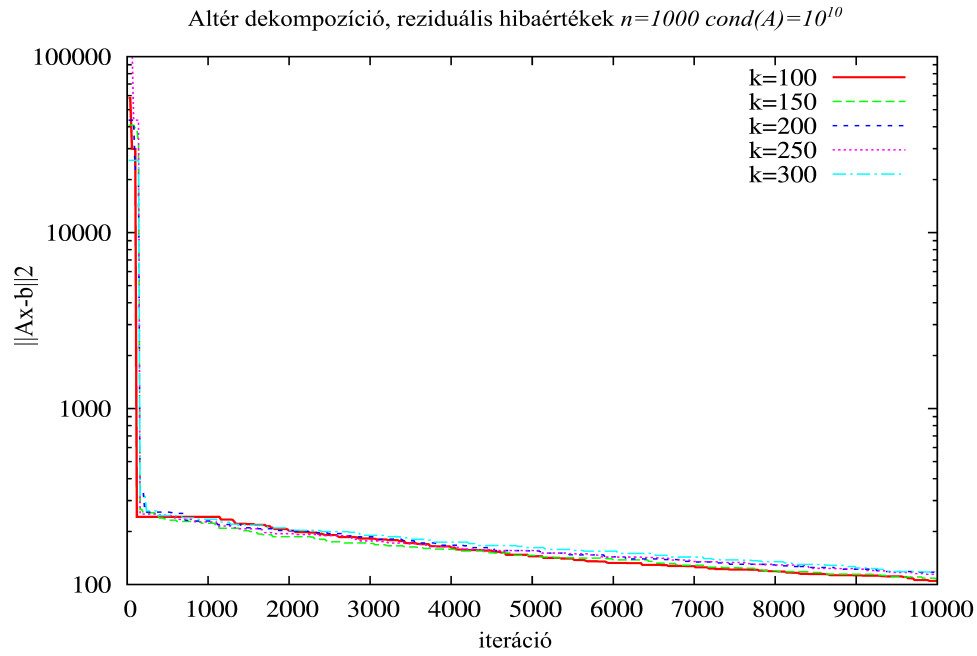
7.6. Algoritmusok futási eredményei

A bemutatott *Algoritmus 6.* – *Algoritmus 7.* párhuzamos környezetben futtatható algoritmusok alkalmasak nagyméretű lineáris egyenletrendszerek megoldására. Az algoritmus a klasszikus legkisebb négyzetek módszerével és a gradiens módszereket alkalmazó megoldások általánosításán alapul. Az általánosítás a konvergencia sebességét nem növeli, de alkalmas a párhuzamos futtatásra. A reziduum minimalizáción alapuló algoritmusok jellemzően lassú konvergenciájúak, ám általános mátrixú feladatok megoldására alkalmazhatók.

Az algoritmus alkalmas a nagyméretű, sűrű és rosszul kondicionált egyenletrendszerek megoldására is. Ezekben az esetekben a megoldás reziduális hibája nagy értékről indul és fokozatosan csökkenthető. Ahogy az előző fejezetben is megmutattuk, a $cond = 10^6$ kondíciószámú feladatban az $\|r^s\| = 0.01$ jó közelítő megoldást jelent, hiszen az abszolút hiba számításakor használt $\|x - x'\| = \left(\sum_{i=1}^{n} (x_i - x'_i)^2 \right)^{\frac{1}{2}}$ képletbe helyettesítve, a kapott megoldásvektor abszolút hibája csupán $\|x - x'\| \approx 10^{-6}$ nagyságrendű. A magas kondíciószámú feladatok a nagy komplexitású, nehezen megoldható numerikus problémák közé tartoznak. Az altér-dekompozíciós algoritmusok tesztelésekor $cond(A) \approx 10^{10} \ldots 10^{16}$ tartományú mintákat vizsgáltunk (*lásd: 3. táblázat*).

Az *Algoritmus 6.* – *Algoritmus 7.* modelleket C nyelvű, MPI üzenetküldés alapú párhuzamos szoftverkörnyezetben készítettünk el, amelyekben a véletlenszámok generálását a jól ismert Mersenne-Twister [Matsumoto 98] algoritmus állítja elő. A párhuzamos, többprocesszoros működés előnye, hogy az eltérő órák segítségével független véletlen sorozatok állíthatók elő. Az algoritmusok futtatásához egy 88 processzort használó *HP Blade systems c3000* számítógépet használtunk.

A futó algoritmus hatékonysága függ az adott gép pillanatnyi teljesítményétől, a terhelés-elosztás módszerétől. Ezért a vizsgált problémákat rendre több esetben, többfajta optimalizációs paraméter mellett mértük. A feladatban egyszerű hierarchikus szervezésű topológiát használtunk. A mester és szolga csomópontok közötti adatcsere és a feldolgozó műveletek arányát kvalitatív módszerekkel úgy optimalizáltuk, hogy a hardver erőforrásait hatékonyan használja ki.



21. ábra: Algoritmus 6. reziduális hibaértékei különböző k értékekkel (n=1000, p=24)

Mivel az algoritmus tartalmaz szekvenciális részeket a mester csomópontban, ezért az Amdahl-törvény értelmében csak egy bizonyos mértékig párhuzamosítható hatékonyan. Mindenképpen létezik olyan maximális processzorszám, ahol a futási idő már nem csökkenthető, illetve a véges hálózati kapcsolatok értelmében a kommunikáció eléri a átviteli maximumát.

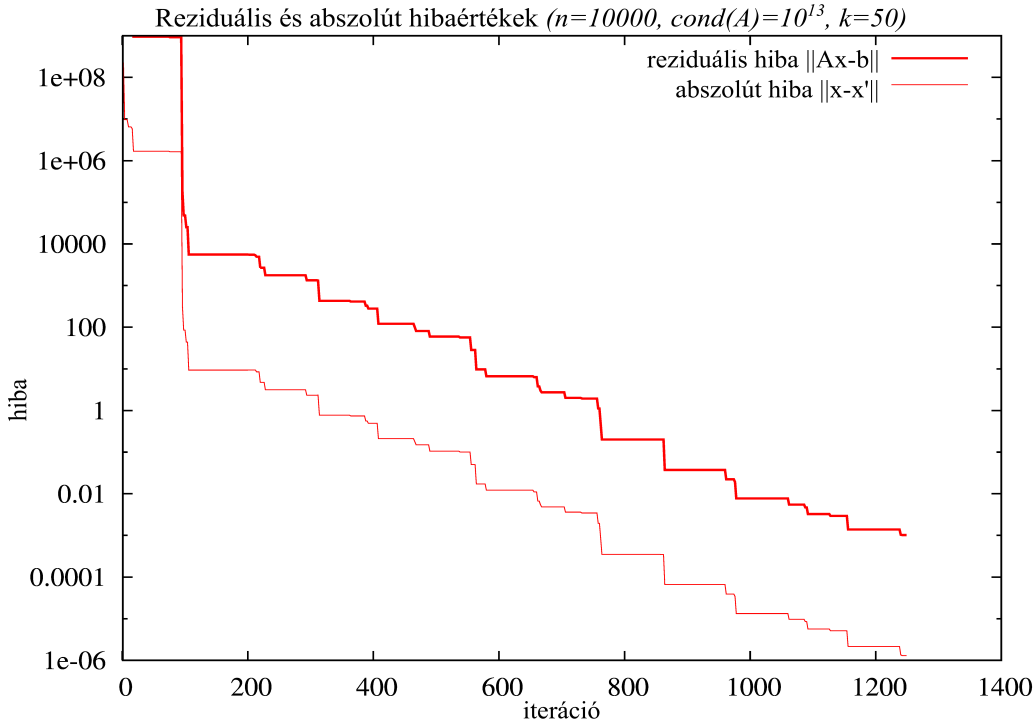
A párhuzamos futtatás során az *Algoritmus 6.* összes költsége $O(p \cdot \xi \cdot k^3)$ p processzoron, de időbeli költsége közelítőleg csak $O(\xi k^3)$.

Ahogyan a *21. ábráról* leolvasható, az altér dekompozíciós algoritmus első lépéseiben egy nagyon gyors hiba csökkenés tapasztalható. A példában *(n=1000,* $cond(A)=10^{10}$*)*, már a 150. iteráció környékén eléri az $\|r^s\| \approx 220$ hibaértéket, amely a pontos megoldáshoz viszonyított $\|x - x'\| \approx 2$ értékű abszolút hibát jelent. Ezek után a minimális reziduum algoritmu-

sokra jellemző, lassú konvergenciával halad tovább az algoritmus ($\xi \approx 10^5$) [Varjasi 07]. Az altér választásának mérete a hibaértékek lecsökkenésének szempontjából csak kisebb jelentőségű [Molnárka, Varjasi 10], szerepére a későbbiekben visszatérünk.

7.6.1. Hatékonyság javítása, optimalizáció

Az algoritmus lassú konvergenciája két hatás miatt következik be. Elsősorban azért, mert az általános lineáris egyenletrendszereknél a véletlen megoldásvektorokat a teljes valós számtartományból generáljuk. Ez abban az esetben, amíg nincs információnk a feladatról előnyös, hiszen egyszerre sok véletlen irányból közelíthetünk, azonban a helyes megoldás közelében már a túl széles intervallum a közelítést lassítja.



22. ábra: Az Algoritmus 7. reziduális és abszolút hibájának alakulása egy futtatás során ($n=10\,000$, $cond=10^{13}$, $k=50$)

A módosított *Algoritmus 7.*-ben ezért azt használjuk ki, hogy az egyes szolga csomópontokban az addigi legjobb x^i (illetve $x=\{x^1, x^2, \ldots, x^g\}$ vektorseregben) a megoldásvektor szórásának megfelelő tartományból generálunk véletlen számokat.

A módosítás hatására az algoritmus konvergenciája gyorsul, ahogy azt a *22. ábráról* leolvashatjuk. Továbbá az is megállapítható, hogy a futtatás során a módosítás hatására az

iterációszám jelentősen lecsökken $\xi = 1250$, emellett az elért abszolút pontosság – a pontos x' megoldásvektor $\sigma x' = 1$ szórása mellett – $err_r \approx 10^{-6}$.

Nagyobb szórással rendelkező megoldásvektort tartalmazó feladatoknál a konvergencia sebességének növekedésére sajnos nem tudunk ilyen „szép” eredményt produkálni, ilyenkor újra a reziduum minimalizáló algoritmusok lassú konvergenciájával találkozhatunk.

7.6.2. Idő, vagy műveletszám szerinti optimalizáció

Ha megvizsgáljuk az algoritmus konvergencia görbéit, akkor láthatjuk, hogy az altér méretének választásakor a hibaértékek lecsökkenése a kiinduló feladat kondíciószámától függ, az altér mérete csak nagyon kis mértékben befolyásolja. Azonban ha azt is megvizsgáljuk, hogy a végrehajtás ideje hogyan módosul akkor az alábbiakat állapíthatjuk meg.

A *4.* és *5.* táblázatban ábrázoltuk egy-egy feladat futási eredményeit a *k* altér mérete szerint. Egyrészt megvizsgáltuk, hogy a feladatot megoldásához hány iteráció szükséges, másrészt azt, hogy mindehhez mennyi idő (wall-clock) szükséges.

A vizsgálatban két minimális értéket találtunk. Azaz a feladat megoldását optimizálhatjuk a futási idő, illetve a legkevesebb műveletszám szerint. A jelenség párhuzamos rendszerek esetén *felülbírálja* azt a klasszikus optimalizációs elvet, hogy az alacsonyabb műveletszám, azaz a gyorsabb konvergencia rövidebb műveleti időt jelent. **Ugyanis állítjuk, hogy a vizsgált *Algoritmus 7.* futása során a legjobb megoldáshoz elvezető műveletsorozat műveletigénye és az ehhez tartozó végrehajtási idő nem minden esetben ugyanolyan paraméterezést kíván. Az *Algoritmus 7.* különböző realizációinak futtatásakor az adatcserékkel működő párhuzamos környezetben az optimalitás minősítésére csak a feladat megoldásának ideje használható egyértelműen.**

A hatás abból adódik, hogy az algoritmus alterében végrehajtott művelet költsége $O(k^3)$. Kisebb *k* érték esetén az altér művelet az egyes egyedi processzorokon gyorsabban megoldható, és az ezzel előállított megoldásvektor a többi csomóponttal kicserélhető, de egy lépésben csak kisebb javító hatást tudunk elérni. Nagyobb *k* érték esetén az algoritmus meredekebben közelíti a pontos értéket, de az egyes lépések hosszabb időt vesznek igénybe.

k	12	15	***20***	30	40	50	60	***70***	80	90	100
idő (s)	346,4	256,4	***233,8***	259,8	328,6	391,4	497,5	509,2	651,7	801,6	763,4
iteráció	4531	2778	1842	1347	1365	1265	1409	***1242***	1373	1529	1313

4. táblázat: Az Algoritmus 7. végrehajtási ideje és a végrehajtott iterációszám kapcsolata (n=5000, p=48)

k	10	***15***	20	30	40	***50***	60	70	80	90	100
idő (s)	4886	***1253,3***	1286	1804	2373	3163	4028	4229	4843	5627	6731
iteráció	10 046	1718	1344	1289	1365	***1285***	1482	1387	1336	1386	1617

5. táblázat: Az Algoritmus 7. végrehajtási ideje és a végrehajtott iterációszám kapcsolata (n=10 000, p=24)

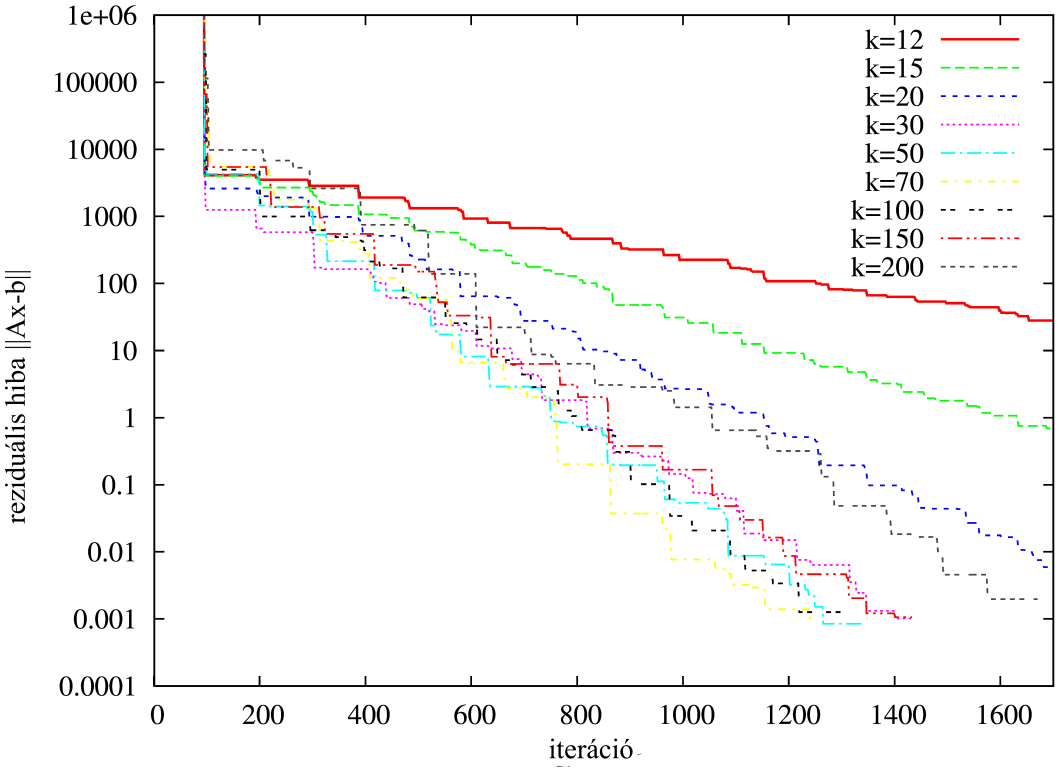
A táblázatokat megvizsgálva azt az eredményt kapjuk, hogy legrövidebb idő alatt $k=20$ ($n=5000$), és $k=15$ ($n=10\,000$) altér méret esetén hajtható végre az algoritmus.

Ha az iterációszám szerint keressük a minimális költséget, akkor ezt $k=70$ ($n=5000$), illetve $k=50$ ($n=10\,000$) altér méret esetén kapjuk meg.

A vizsgálat során azt is tapasztaltuk, hogy extrém kicsi ($k<10$), illetve nagy ($k>200$) esetén az algoritmus konvergenciája jelentősen leromlik, hiszen a kis alterek választása során a gyors számítási fázisokat jelentős kommunikációs kapcsolat terheli, míg a nagy alterek végrehajtása esetén a nagyobb részfeladatok között ritkák az adatcserék. (Megjegyzés: az alterekben direkt LU megoldó algoritmusokat használtunk, de iteratív algoritmusok használatánál is hasonló jelenség várható.)

A *23. ábrán* látható grafikon konvergencia görbéi alapján első ránézésre azt várnánk, hogy a legmeredekebb gradiensű altér méretet célszerű választani, azonban a mérési eredmények azt mutatják, hogy a futtatáshoz használt számítógépen a megoldandó feladat rangjánál nagyságrendekkel kisebb méretű ($k=15,\ 20$) altér választás volt időben a leggyorsabb (a vizsgált feladat mérete $n=5000$ és $10\,000$ volt). (Ez a társasjátékos példával illusztrálva azt jelenti, hogy nem a ritkán, de a legnagyobb lépésekkel haladó játékosok, hanem a sűrűn közepes lépésekkel haladók biztosítják a gyors célba érkezést.)

A kis altér méretek sikerét támasztja alá az a megfigyelés is, amely azt mutatja, hogy az altér-dekompozíciós algoritmus az adatcseréknek köszönhetően evolúciós jellegű.

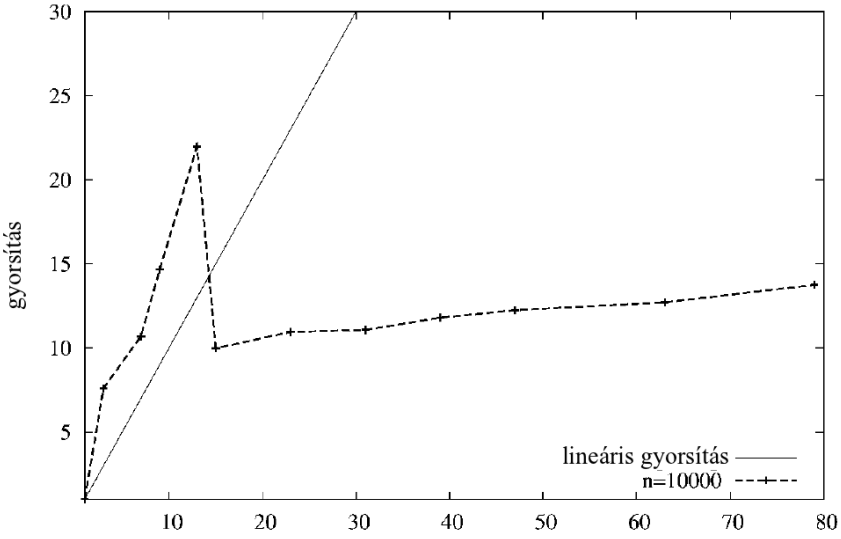


23. ábra: Algoritmus 7. altér méret szerinti konvergencia görbéi

Ezt megfigyelhetjük a *24. ábra* gyorsítási diagramján, amely egy konkrét feladatosztály vizsgálatával állt elő.

A kisméretű alterekben végzett megoldások hatékony cseréjével a független processzorok által számított értékek javító hatása érvényesül. Így a futtatáskor 12 processzor esetén 24-szeres gyorsítás (200%-os hatékonyság) is elérhetővé vált (*n=10 000, k=15*).

Az ábráról leolvasható ezután egy erőteljes visszaesés és gyorsítás romlás: az egyszerű, mester-szolga topológia nagy hátránya, hogy egy idő után a kommunikációs csatorna feltelik, és a mester csomópontban kommunikációs torlódások keletkeznek. Ezek az algoritmus futtatásakor hirtelen teljesítményromlással járnak. (Az ábrán csak azért jelöltünk egyetlen gyorsítási görbét, mivel a különböző kondíciószámú és különböző méretű feladatoknál más-más processzorszámnál jelentkeztek a javulások és visszaesések, ez így a leolvasást nagyban nehezítené.)



24. ábra: Altér-dekompozíciós algoritmus gyorsítási értékei (n=10 000, k=15)

A probléma feloldásához egyrészt a terhelés elosztás vizsgálatával az átadott vektorok számának és a szolga csomópontokon elvégzett közbenső iterációk számának alakításával lehet javításokat elvégezni. Másrészt pedig az algoritmust fel kell bontani egy két- vagy többszintű hierarchikus topológiára (a mester-szolga szerepek közé segédmester csomópontokat kell beiktatni, úgy hogy egy csomópont csak relatív kevés csomóponttal kommunikáljon). Az altér dekompozíciós algoritmusok fejlesztésében és a különböző topológiák vizsgálatával a további kutatásokban fogunk foglalkozni.

Összefoglalás

Az előzőekben nagyméretű és rosszul kondicionált lineáris egyenletrendszerek megoldására szolgáló párhuzamos algoritmusokat vizsgáltunk. A fenti algoritmusok a reziduum minimalizációs technikára épülnek, amelyeket altér-dekompozíciós lépéssel kiegészítve hierarchikus topológiájú sokprocesszoros számítógépen futtattunk. Az algoritmusra megfigyelhető az evolúciós algoritmusokra jellemző javító hatás, amely adatcserék mellékhatásaként az algoritmust alkalmassá teszi a hatékony párhuzamos futtatásra.

A számítógépes tesztek alátámasztották az algoritmus elméleti síkon megfogalmazott előnyeit. A szekvenciális algoritmusnál hatékonyabb párhuzamos futási eredményeket tapasztaltunk: alkalmasak nagyméretű párhuzamos gépeken, vagy elosztott klaszter-rendszereken való futtatásra.

Az altér-dekompozíciós párhuzamos algoritmus alkalmas olyan valós problémák hatékony megoldására, amelyek a mérnöki gyakorlatban gyakran előfordulnak: optimalizáció, véges-elemes rendszerek, szállítási-, vezérlési-, vagy szimulációs feladatok, amelyeknél problémát okoz a sűrűn kitöltött, rosszul kondicionált egyenletrendszerek megoldása. Alkalmazásával ezek a nehezen megoldható problémák is rövid idő alatt közelítő eredményt adhatnak, ugyanis visszavezethetők az alterekben levő kisméretű feladatok párhuzamos megoldására. Az altérben képződő lineáris egyenletrendszerek számos, a *4.2.1. fejezetben* ismertetett módszerrel oldhatóak meg. A kutatásban mi nem vállalkoztunk minden egyes szóba jöhető módszer kvantitatív elemzésére és összehasonlítására, meglátásunk szerint a feladat összetettsége túlmutatna ezen értekezés keretein.

Lehetséges jövőbeli kutatási irányok

A fenti algoritmusokban minden esetben előre megválasztott k értékekkel oldottuk meg a lineáris egyenletrendszereket. Ezt a korlátozást azért használtuk, hogy a különböző párhuzamos topológiákra megfogalmazott algoritmusok definíciója egyszerűbb legyen. Általános esetben egyetlen k érték helyett megadhatnánk egy $k=\{k_1,k_2,\ldots,k_u\}, u\in\mathbb{N}$ értékhalmazt, amely azt jelenti, hogy az algoritmusban az egyes szolga csomópontok ezen változó $k_i, i=1\ldots u$ értékek mellett eltérő dimenziójú alterekben dolgoznak. Ez a tulajdonság heterogén rendszerekben jelent előnyt, de ilyen kísérleteket az algoritmus tesztelése során nem végeztünk.

A előzőekben ismertetett párhuzamos algoritmusok hatékonyságának javítása, azok konvergenciájának gyorsítása természetes jövőbeli kutatási irány lehet. A dolgozatban kifejtett párhuzamos algoritmusokat különböző architektúrához illesztésekhez – a dolgozatban egyelőre nem teljes mértékben használt – vezérlő paraméterek szerinti optimizációs lehetőségeket tartalmaz. Ezen paraméterek megfelelő megválasztása lehetővé teszi az algoritmusok illesztését a különböző heterogén, vagy hibrid párhuzamos hardver környezetekhez is.

8. PÁRHUZAMOS ALGORITMUSOK HATÉKONYSÁGA ÉS A SOKPROCESSZOROS RENDSZEREK

A párhuzamos programozás céljának a nagy feladatok rész-problémákra való felosztását, és ezen rész-problémák egyidejű hatékony megoldását tekinthetjük. A párhuzamos feladatmegoldásban ez a hatékonyság két fő – egymástól független – problémakörre bontható. Az első a feladatok szekvenciális programozásban jól ismert *aritmetikai költsége*, a második pedig a párhuzamos elosztott rendszerekre jellemző *kommunikációs költség* az egyes végrehajtó egységek (processzorok) között. Ezt a felosztást mind a szorosan-, mind a lazán csatolt rendszerekben megtehetjük, jóllehet a két költség komponens más-más súllyal jelentkezik. Az *6.* és *7. fejezetben* ismertetett párhuzamos algoritmusok futási eredményeinek előzetes becslésekor, illetve ezeknek az adott hardver-architektúrán realizált eredményeinek vizsgálatakor eltéréseket tapasztaltunk. Ezekből a tapasztalatból kiindulva kezdtünk a téma alaposabb tanulmányozásához. A vizsgálatok eredményeként az alábbiakban megmutatjuk, hogy egy adott algoritmus csak akkor lehet hatékony, ha e két szempont között az arany középutat megtaláljuk. Ehhez a rendelkezésre álló számítógépes rendszerről, a használt topológiáról valamilyen mérhető értékre van szükségünk. Az optimális futási eredményekhez a rendszer beállításait és pillanatnyi állapotait mérni kell, és ehhez a megfelelő topológia és a megfelelő párhuzamosító eljárás alkalmazható, illeszthető.

A párhuzamos feladatok végrehajtásának ezen vizsgálatát a szakirodalom feladat elosztásnak – *load-balancing* nevezi. A felsorolt szakirodalomban ezt a problémát már számos nézőpontból megvizsgálták, de a kutatás napjainkban is folyamatos ezen a téren.

Sleator [Tarjan, Sleator 85] és Tarján [Tarjan 85] klasszikus gráfelméleti és algoritmizálási tanulmányaikban, bináris fák elemzésekor megállapították az azok kiegyensúlyozásához szükséges feltételeket. Tarján javaslatot tett egy ún. potenciálfüggvényre, amely a bináris fák kiegyensúlyozásánál az adott struktúrát jellemzi. Mi ezen fogalomból kiindulva fogjuk a hálózati topológiákon mérhető potenciált értelmezni.

A lineáris egyenletrendszerek párhuzamos megoldása is régóta foglalkoztatja a kutatókat: Duff részletes elemzést készített a lineáris egyenletrendszerek párhuzamosítható megoldó

algoritmusairól [Duff 99]. Munkájában a hangsúlyt a hálózati topológiáktól függő optimalizált megoldásokra helyezte.

Becciani definiált egy speciális fa-struktúrát, melyet az N-test szimulációban használt párhuzamos algoritmusok leírásához és elemzéséhez. Definiálta az erőforrás elosztás és a hálózati optimalizáció fogalmán alapuló optimizációs módszereket [Becciani et al. 97]. Lin egy olyan időben elcsúsztatott eseményeket leíró kommunikációs modellt készített a klaszter rendszerekhez, amely a leggyakoribb hálózati konfliktus jelenségeket jellemzi. Például a sokszor pulzáló hálózati működésekre, az egyidejű kommunikációs konfliktusokra jó leírást ad [Lin 2000].

Yero már a nagyméretű, mester-szolga architektúrákon jelentkező terhelés kiegyenlítés modelljeivel foglalkozik [Yero 07]. Napjainkban Schliephake dolgozott ki újabb, jellemzően a nagyméretű klaszter rendszerekre alkalmazható hálózati- és rendszerleíró modelleket [Schliephake 11].

8.1. Topológiák összehasonlítása

A párhuzamos programozási környezetben az egyes számítási csomópontokat többféle szereppel felruházhatjuk. A legegyszerűbb esetben nincsenek kitüntetett végrehajtó egységek, minden csomópont egyenrangú, azonos feladatot lát el. Az ilyen feladatokhoz számos topológiát találhatunk: gyűrű, kocka, hiperkocka, pillangó, stb. ahogy azt a *2.2.1. fejezetben* bemutattuk.

Az ilyen rendszerekben a párhuzamos algoritmusok „egyszerűbbek”, hiszen a programok végrehajtásában tulajdonképpen csak két feladat ismétlődik: az algoritmus egyes részfeladatainak végrehajtása és a csomópontok közötti kommunikáció. Az optimális megoldást ott érhetjük el, ahol ez a két szempont maximálisan kihasználja az adott rendszer lehetőségeit. (Minden processzor feladatot hajt végre, és a kommunikációs csatornák jól működnek.) Tulajdonképpen – a korábban ismertetett – PRAM modell is ezt az egyszerűsítést alkalmazza.

A szimmetrikus topológiák előnyei:

- egyszerűbb kódalkotás,
- rögzített processzorszám,

- fix méretű hálózatok,

Hátrányok:

- csak homogén rendszerekben hatékony,
- gyakran jelentkezik „szívdobogás” (*heartbreath*) effektus (a végrehajtás nem folyamatos, hanem a processzorok sebesség-különbsége, vagy a kommunikációs késések miatt periodikus),
- a processzorszám emelésével a topológia elbonyolódik,
- a kommunikációs költségek exponenciálisan nőnek,
- nehezen alakítható át a leírt kód: kisebb változtatásokra is jelentősen megváltozhat a rendszer viselkedése.

8.1.1. Egyszerű hierarchikus modell

Abban az esetben, ha a hardverkörnyezet inhomogén, vagy a feladat elosztott jellegű, akkor hierarchikus modellt definiálunk. A hierarchikus modell legegyszerűbben a mester-szolga modellel írható le. Ekkor az elvégzett feladatok elkülönülnek, a mester a megoldandó probléma darabolásáért, összesítéséért, és a részfeladatok kiosztásáért felel; ezen részfeladatokat pedig a szolga csomópontok oldják meg (*lásd: 25. ábra*).

hálózati topológia séma *csomóponti vektor*

25. ábra: Fa (csillag) topológia és vektoros reprezentációja

Előnyei:

- egyszerű topológia,
- folyamatos, vagy aszinkron kommunikáció,
- az egyes szolgák egymástól függetlenek, csak a mesterrel kommunikálnak,
- bővíthető más méretű rendszerekre.

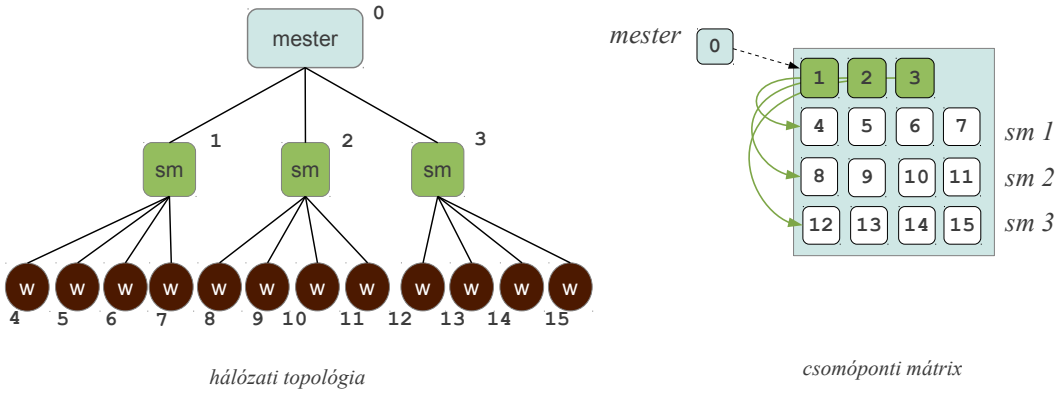
Hátrányai:

- a feladat hatékony darabolására van szükség,
- a mesternek nyilván kell tartania az egyes részfeladatok állapotát,
- érzékeny a hálózati csatornára,
- a mester jelenléte rontja az összesített hatásfokot [Yero 07], [Lin 2000].

Ez a hierarchia egy, egy mélységű fával írható le, ahol *p* processzor esetén a 0. processzort mester, az *1, 2, ... , p-1* processzorokat szolga szerepben használjuk. A csomópontok struktúrája a legegyszerűbben egy egydimenziós tömbbel implementálható [Király 11].

8.1.2. Összetett hierarchia

A megfigyelések és a párhuzamosítási alapelvek értelmében azonban a hálózatot itt sem bővíthetjük korlátlanul [Amdahl 67] [Gustafson 88]. A processzorszám növelésével egy feladat végrehajtási ideje egy bizonyos szinten túl nem csökkenthető hatékonyan ezzel a modellel. Míg a mester jelenlétével megjelenő hatékonyság romlás – eggyel kevesebb processzor végzi el a tényleges munkát – a szolgák növelésével csökken, addig a kommunikációs csatornák végessége, illetve a késleltetési idők a szolgáknál éheztetést, a mesternél pedig folytonos túlterhelést jelent [Chen 10]. Az ilyen egy mélységű fa struktúrát használó mester-szolga rendszerek csak addig hatékonyak, míg az algoritmikus végrehajtási idő nagyobb, mint a processzorok közötti kommunikáció ideje.



26. ábra: Többágú fa topológia és a reprezentáló mátrix struktúra

Amennyiben a kommunikációs idő kritikus, a hierarchikus topológia bővíthető három- illetve többszintű rendszerekre, melyeket a *26. ábra* mintájára n-ágú fákkal írhatunk le. Ebben a modellben a mester és szolga szintek közé beiktathatunk egy, illetve több segédmes-

ter réteget. Ekkor a feladatok felosztásánál és az algoritmusok tervezésénél az alábbi elveket kell figyelembe venni:

- a mester szerepe megmarad, de csak a segédmester csomópontokkal kommunikál,
- a segéd csomópontok a részfeladatokat tovább darabolják, ezeket kiosztják a szolgáknak, majd az eredményeket összesítik és a mesternek jelentik a feladat részeredményeit,
- a szolgák elemi részfeladatokat oldanak meg.

A többszintű hierarchia előnyei:

- a kommunikációs költségek a struktúra összetettsége ellenére csökkenthetők, [Duff 99],
- egyenletesebb terhelés elosztás, jobb skálázódás [Yero 07].

Hátrányok:

- a feladat további feldarabolása nem mindig triviális, hiszen a feladatok felbontásához az algoritmusok újratervezése szükséges (Foster-módszer),
- növekvő nyilvántartási „bürokrácia": csökken a tulajdonképpeni feladatot végrehajtó processzorok száma, és emelkedik a csupán menedzselő, vagy kommunikációs feladatokra használt processzorszám,
- csak nagyobb számú processzort tartalmazó rendszereknél használható hatékonyan ez a módszer.

A hierarchia leírható többágú fa hierarchiával, vagy a csomópontokat tartalmazó mátrixszal (*lásd: 26. ábra*).

Hierarchikus rendszerek jellemzése

A magasabb fokú hierarchikus topológiájú rendszereknél a hierarchia, és így a kommunikációs feladatok leírása többféle elv szerint megadható:

- előre rögzített modellel,
- kiegyensúlyozottság mértéke alapján,
- véletlenszerűen, vagy
- a következő szabad segédmester elve alapján (ekkor a szolgák nincsenek rögzítve egy segédmesterhez, hanem a feladatot az éppen szabadnak jelentik).

8.2. Hierarchikus rendszereket jellemző modell és mérőszámok homogén rendszerekben

Az előző fejezetekben ismertetett algoritmusok párhuzamos végrehajtásakor és futási eredményeinek elemzésekor megfigyeltük, hogy a futási eredmény valamilyen módon összefügg a topológia szerkezetével. *Erős*ebbnek, vagy *kiegyensúlyozott*abbnak nevezhetünk egy olyan topológiát, amelyen ugyanazt a feladatot végrehajtva, az adott hálózaton kisebb költséggel oldja meg.

A szakirodalom és a felhasznált kutatási eredmények tükrében a hierarchia leírásához a bináris keresőfák rendszerét vettük alapul. Ezekben egy algoritmus végrehajtása során a struktúra akkor nevezhető optimálisnak (legkisebb költségű műveletek), ha a kiegyensúlyozott fa rendszereket használunk [Cormen et al. 01] [Tarjan 85].

A bináris fák használatánál az optimális struktúrához többféle módszerrel is rendelhető mérőszám, ezek közül az ún. *potenciál módszert* alapul véve megadható egy olyan amortizációs függvény, amely a bináris fa kiegyensúlyozottságát írja le [Cormen et al. 01, 322p.].

Az alábbiakban ezen mérőszámokon alapulva a következő hipotézist állítjuk: a potenciálmódszerhez hasonlóan definiálható egy olyan átmenetfüggvény, amellyel megadható egy topológiát jellemző mérőszám. A fentiekben leírt hierarchikus topológiákra ez a potenciál jellegű függvény kiterjeszthető oly módon, hogy a potenciálfüggvény a topológia erősségét jelezze [Molnárka, Varjasi 11b].

8.2.1. Potenciálfüggvények

Célunk egy olyan speciális mérőszámot találni, amely a – homogénnek és egységes költségűnek tekintett – hálózati kapcsolatok rendszerét, a különböző feladatot végző processzorok számától függően egy [0,1] valós intervallumbeli Φ értékkel jellemzi: $\Phi : T \rightarrow [0,1] \in \mathbb{R}$, ahol T egy hálózati topológia. Vizsgálataink során T azon részhalmazával foglalkoztunk, amelyek összefüggő körmentes gráf struktúrával leírhatók. Így a feladat tulajdonképpen a következőre egyszerűsödik: fához valós érték rendelése. Mivel az általános, többágú fa struktúrák csak sok paraméterrel írhatók le, így a kvalitatív elemzéssel megalkotott modell egyszerűsítésére törekszünk. (Tulajdonképpen egy gráf geometriai reprezentációját kell algebrai alakban megadni. A problémát az jelenti, hogy adott számú

csomóponttal és adott számú éllel egyszerre több struktúra is leírható.) A hozzárendelést indukciós úton definiáljuk oly módon, hogy kezdetben a topológiát leíró adatok meghatározását egy- majd többparaméteres leíró modellel közelítjük. A definiált modellek validációját számítógépes futtatásokkal végezzük.

Kiindulásként tekintsük az alábbi egydimenziós potenciálfüggvényt, amely a fa struktúrát egyetlen pozitív egész paraméterrel, a processzorok számával jellemzi:

$$\Phi_p = \frac{p-1}{p} \quad ; \quad \Phi_p : N \rightarrow [0,1], \tag{20}$$

ahol p a rendszerben használt processzorok száma. Ez az egyszerű potenciálfüggvény csak annyit árul el a rendszerről, hogy egy mester (elsősorban „menedzser" funkciókkal rendelkező) csomópont kiemelésével az egész struktúra maximális számítási teljesítménye mennyiben változik. Megfigyelhető, hogy a processzorszám növelésével Φ_p monoton közelíti az 1 határértéket.

A hozzárendelés elméletileg jól jellemzi a hierarchikus struktúrákat, azonban a monoton növekedés ellentmondásban van Amdahl törvényével [Amdahl 67], amelyet a gyakorlati tapasztalatok is alátámasztanak. Ezért ez az egyparaméteres modell (20) a gyakorlatban jelentkező problémákra még nem ad kielégítő választ. Például nem tartalmaz információt a topológiáról, vagy a csomópontok egymáshoz kapcsolásának módjáról, így a hipotézist nem elégíti ki.

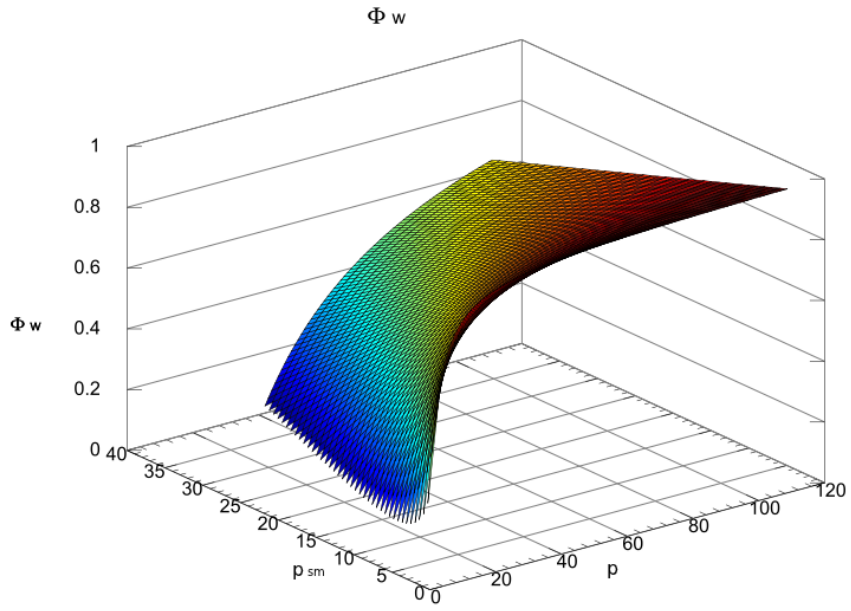
Ezért a függvény kiterjesztésére van szükség, amelyben figyelembe vesszük az összetettebb, többszintű hierarchiát, illetve a segédcsomópontok szerepét is. Legyen Φ_W kétparaméteres – a szolgákat jellemző – potenciálfüggvény:

$$\Phi_W = \frac{p_w}{p}; \quad \Phi_W(p, p_w) : N \times N \rightarrow [0,1], \tag{21}$$

ahol p az összes processzorok száma, $p_w = p - p_{sm} - 1$ a szolga processzorok száma (a fa leveleinek száma) és p_{sm} a köztes szinteken található segédmester processzorok száma (a fa belső csúcsainak száma). Mivel a fa minden esetben összefüggő, és a segédprocesszorokhoz legalább egy szolga processzor kapcsolódik, ezért a processzorok számára teljesülnie kell a $p > p_w$ és $p_w \geq p_{sm}$ feltételeknek is.

A modellben a két paraméter segítségével jellemezzük a struktúrát. Megjegyzés: a (21) formula csillag topológiánál (20) alakúra egyszerűsödik ($p_{sm}=0$).

A Φ_W értékeit ábrázolhatjuk a processzorszám, és a segédmester processzorok számának függvényében (*lásd a 27. ábrán*), amelyről leolvasható, hogy a processzorszám növeléssel a potenciálfüggvény szigorúan monoton növekvő értéket alkot.



27. ábra: Szolga csomópontok alapján meghatározható potenciál függvény: $\Phi_W(p, p_w)$

Ez a modell még mindig nem elégíti ki a hipotézist, azonban már figyelembe veszi a processzorok feladatok szerinti elosztását, de továbbra sem ad leírást a kommunikációs jellemzőkről. Nem veszi figyelembe azt, hogy egy segédmester csomópontnak hány kapcsolódó szolgája lehet, és azt sem, hogy a párhuzamos kommunikáció során kiegyensúlyozott, vagy kiegyensúlyozatlan rendszert vizsgálunk. (Adott p, p_w mellett több különböző topológia is leírható.)

A modellünket további paraméterekkel kell bővítenünk, hogy figyelembe vehessük a topológia jellegzetességeit. Az általános fa struktúrát jól jellemzi a csúcsokból a gyerekcsomópontok felé induló élek száma[4]. Ez általános fa esetén változó értékű, kvalitatív elemzés-

4 A továbbiakban fokszám. „Gyökeres fában … a fokszám a gyerekek száma, tehát a szülő nem számít bele a fokszámba." [Cormen et al. 01, p 77.]

sel vizsgálva jól jellemzi a kiegyensúlyozottságot. A kvantitatív modell felállításához, a részletesebb leírás megkönnyítésére vezessük be a következő Δx korrekciós értéket, amely megadja a fa belső csúcsaihoz rendelt fokszámainak legnagyobb különbségét. (A levél csúcsok fokszáma 0, ezeket figyelmen kívül hagyjuk.)

Legyen

$$\Delta x = \max_{i \in \{0,1 \ldots p_{sm}\}} (d(i)) - \min_{i \in \{0,1 \ldots p_{sm}\}} (d(i)), \tag{22}$$

ahol $d(i)$ az *i*-edik csomópont fokszáma (közvetlen kiterjesztésének száma), továbbá p_{sm} a fa belső csúcsainak száma (a köztes szinteken található segédmester processzorok száma).

Vezessük be a következő jelölést a fokszámok átlagára:

$$p_{avg} = \frac{\sum_{i \in \{0,1 \ldots p_{sm}\}} d(i)}{p_{sm}} \tag{23}$$

Megjegyzés: Mivel a fa struktúrákkal többprocesszoros rendszereket írunk le, ezért olyan eset nem fordul elő, hogy egy segédmesternek ne legyen szolga csomópontja, így $d(i) \geq 1$, továbbá a formulát csak többszintű hierarchiában értelmezzük, ezért $p_{sm} > 0$.

A fokszámok átlaga megadja a teljes struktúra átlagos elágazásainak számát.

Felhasználva a (22) és (23) összefüggéseket definiálhatunk egy új potenciálfüggvényt, amely már a fa szerkezetének kiegyensúlyozásáról ad információt:

$$\Phi_L = \frac{p_{sm} \cdot p_{avg} - \Delta x}{p_w}; \quad \Phi_L(p, p_{sm}, p_{avg}, \Delta x) : N^3 \times R \rightarrow [0,1], \tag{24}$$

A Φ_L négy változó alapján jellemez egy topológiát: a csomópontok, segédcsomópontok száma mellett a fokszámok közötti legnagyobb eltérés és a fokszámok átlaga is befolyásolja a függvény értékét. A leképezés olyan topológiákra ad $[0,1]$ intervallumbeli értékeket, amelyek gyökeres, összefüggő fával leírhatók, és teljesül rájuk az a kikötés is, hogy egy segédmester csomóponthoz legalább egy levél kiterjesztés (szolga) kapcsolódik. Az így leírható topológiák száma elsősorban *p*, és p_{sm}-től függ. Emellett nagyon fontos még megjegyeznünk, hogy a topológiák csak diszkrét összeállításokkal írhatók le.

A képletből leolvasható, hogy a potenciálfüggvény alacsony Δx értékekhez rendel magasabb Φ_L potenciál értéket, továbbá a potenciál érték akkor a legmagasabb, ha a struktúrában a fa minden csomópontjának azonos számú leszármazottja van (kiinduló élek száma azonos), amely a következő összefüggések mellett belátható:

$$\frac{p_{sm} \cdot p_{avg} - \Delta x}{p_w} = \frac{\sum_{i \in \{0,1 \ldots p_{sm}\}} d(i) - \max_{i \in \{0,1 \ldots p_{sm}\}} (d(i)) + \min_{i \in \{0,1 \ldots p_{sm}\}} (d(i))}{p_w} \geq \frac{\min_{i \in \{0,1 \ldots p_{sm}\}} (d(i))}{p_w} \geq 0 \qquad (25)$$

A bemutatott Φ_W és Φ_L potenciálfüggvények értékei két külön aspektusból jellemzik a topológiát: egyrészt a processzorok száma és szerepei, illetve a fa szerkezete (elágazásai) szempontjából. A két érték lineáris kombinációjából a teljes struktúrát jellemző potenciálértéket kaphatunk.

8.2.2. Potenciálfüggvény homogén hálózatok leírására

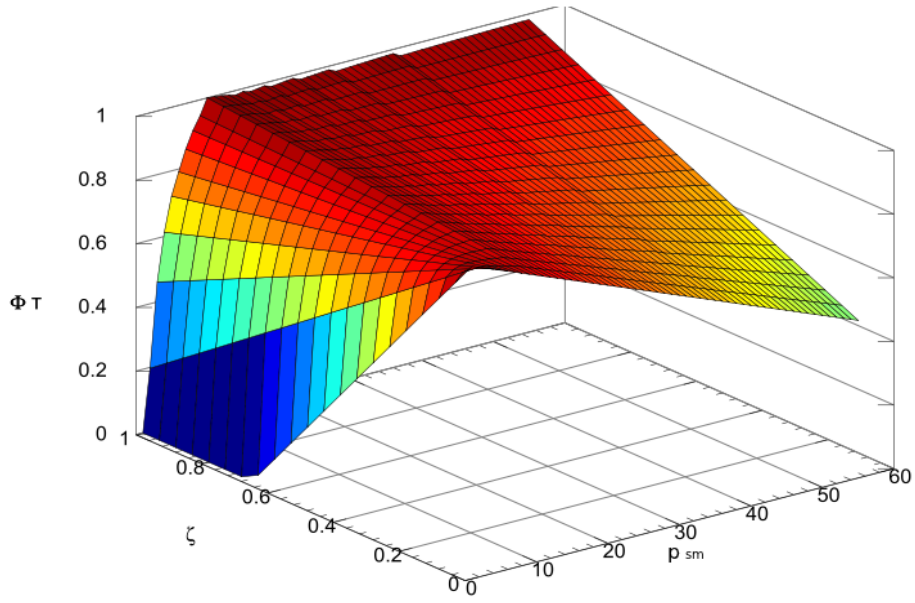
A fenti Φ_W és Φ_L potenciálfüggvények alapján felírható a következő összefüggés:

$$\Phi_T = \zeta \Phi_L + (1-\zeta)\Phi_W; \quad \Phi_T(p, p_{sm}, p_{avg}, \Delta x, \zeta) N^3 \times R^2 \rightarrow [0,1], \qquad (26)$$

ahol Φ_W (21) és Φ_L (24) alapján határozható meg, míg $\zeta \in [0,1]$ egy empirikus korrekciós érték.

A kifejezés a már ismertetett paraméterek mellett tartalmazza azt ζ értéket, amellyel a két potenciálfüggvény súlyozható. A korrekció $\zeta = 0$ értékeknél csak a topológia, $\zeta = 1$ értéknél csak a processzorok jellege alapján jellemzi a struktúrát. (Megjegyzés: a súlyozás gyakorlati szempontok miatt azért lesz fontos, hogy a valós architektúrákhoz egyetlen paraméter szerint legyen optimalizálható az összefüggés.)

Állítjuk, hogy a (26)**-ben felírt potenciálfüggvény alkalmas a homogén – egységes teljesítményű processzorokkal bíró és egységes költségű kommunikációs csatornát használó – hierarchiák terhelés elosztásának leírására.**

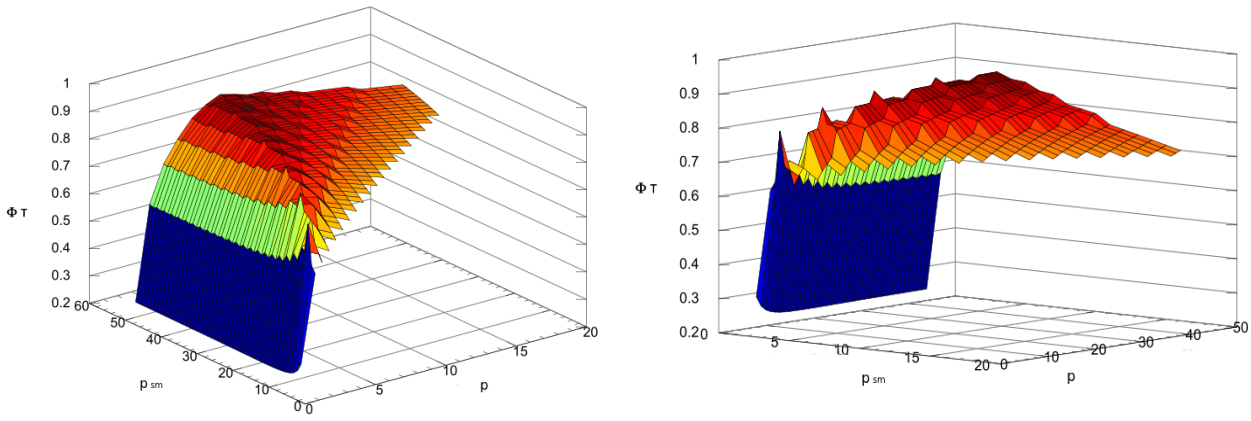


28. ábra: A potenciálfüggvény maximális értékei $\Phi_T(\zeta, p_{sm})$ dimenziójában, rögzített processzorszámnál (p=111)

A Φ_T potenciálfüggvény grafikusan is ábrázolható oly módon (*lásd: 28-29. ábrák*), hogy egyes paramétereket valamilyen szabály szerint rögzítünk. Az ábrákkal a potenciálfüggvény olyan tulajdonságait szeretnénk reprezentálni, amely megmutatja, hogy az egyes topológiákhoz (a rögzített tulajdonságok mellett), két-két változó paraméter esetén hol jellemzi valamilyen maximum érték.

A *28. ábrán* a Φ_T potenciálfüggvény értékeit ábrázoltuk a segédmester csomópontok és a ζ korrekciós érték összefüggésében, rögzített *p* érték mellett. Az ábrán látható felületet azokra a maximális értékekre feszítettük ki, amelyek a *p*, p_{sm} érték mellett minimális Δx értékeket vett fel. (Tulajdonképpen az ábrán egy sűrű, diszkrét ponthalmazt kellett volna ábrázolni, az összes lehetséges esettel, de számunkra a függvény maximális értékei az érdekesek.)

Az ábrán megfigyelhető, hogy egy adott processzorszámhoz – a segédmester csomópontok számának és a korrekciós érték változtatásával – a maximális értékek különböző kezdőértékeknél is találhatóak. Ezért vesz fel a függvénymaximumok burkolófelülete ilyen legyezőszerű alakot. (A gyakorlathoz ez úgy kapcsolódik, hogy a szorosan csatolt gépeknél alacsonyabb, lazán csatolt gépeknél magasabb ζ értéknél kapunk a futási eredményekhez közelítő eredményeket.)



29. a, b. ábra: A potenciálfüggvény maximális értékei $\Phi_T(p, p_{sm})$ *dimenziójában, rögzített* $\zeta = 0.5$ *esetén*

A *29. ábrán* a processzorszám és a segédprocesszorszám dimenziójában látjuk a Φ_T függvény értékeit. Az ábrázolhatóság miatt a korrekciós értéket $\zeta = 0.5$ középértéknél rögzítettük. Mivel adott p, p_{sm} értékekkel több topológiát is ábrázolhatunk, ezért itt is az adott processzorszámoknál minimális Δx értékkel rendelkező eseteket vettük figyelembe. Az ábrán az így kapott a maximális értékekre feszített burkolófelületet láthatjuk. (Amennyiben az összes szóba jöhető esetet ábrázolnánk háromdimenziós pontsorozatként, a maximális értékek megállapítása nehezebb volna.)

A fát numerikusan reprezentáló csomóponti mátrix megadásnál a potenciálfüggvény maximuma a kitöltött értékek „négyzetes mátrix" alakjánál adódik (*vö. 30. és 31. ábra*). (A csomóponti mátrixok ilyen ábrázolásával a területmaximumokkal kapcsolatos analógia is felfedezhető.)

A fenti potenciálfüggvény a homogén rendszerekben használható hatékonyan, hiszen az egyes csomópontok egyenrangúak, és az egyes éleknél sem használtunk irányítottságot, vagy költségeket. A későbbiekben a modell kiegészítendő azzal a két gyakorlati ténnyel, hogy egyrészt a processzorok teljesítménye különböző (a csomópontok nem azonos értékűek), másrészt a processzorok közötti adatcsere sohasem azonnali, minden estben számolni kell késleltetéssel, vagy a kommunikációs csatornák sebesség különbségével (élek súlyozása).

Következmény: **a potenciálfüggvény** (26) **segítségével az egyes hierarchikus topológiák elemezhetők, a hosszabb futásidejű feladatok végrehajtása előtt egy „gyors"**

tesztfeladattal az adott hardverre optimalizált topológia generálható (a processzorszám és a hardverre jellemző korrekciós érték segítségével) [Molnárka, Varjasi 11b]. (Ilyen jellegű gyorsteszteket az ipari gyakorlatban is használnak, például a nagyméretű kommunikációs hálózatok forgalmának elemzésére és szimulációjára [Lencse 01] [Muka, Lencse 10].)

Eredmények:

Az elosztott és párhuzamos rendszerekben a hierarchikus szervezésű topológiákon futtatott algoritmusok különböző terhelés eloszlás értékeknél különböző futási időket produkálnak. A futási idő változásait a topológia szerkezete is befolyásolja.

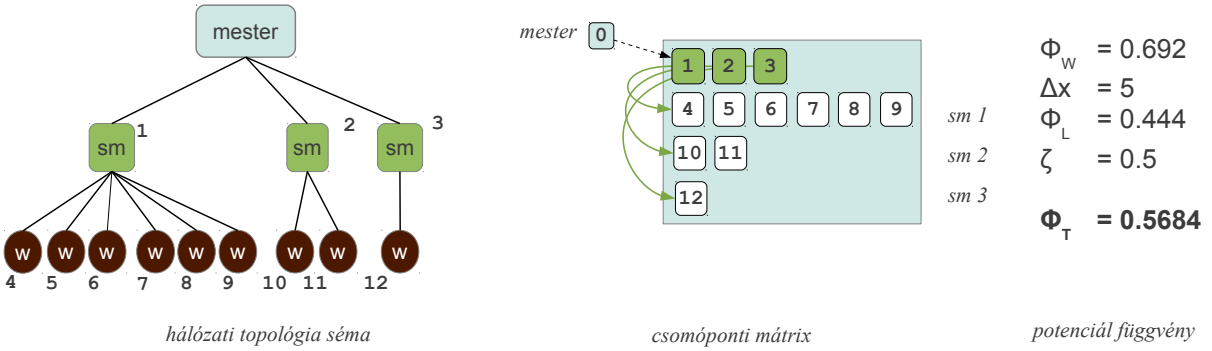
A fenti módszerrel megadott és általunk javasolt Φ_T potenciálfüggvénnyel jellemezhető hierarchia futási ideje – azonos csomópontszámnál – Φ_T maximális értékénél a legkisebb.

A számítógépes tesztek futási idejének elemzésével a fenti állításokat empirikusan alátámasztottuk [Molnárka, Varjasi 11b].

8.3. A potenciálfüggvény alkalmazása

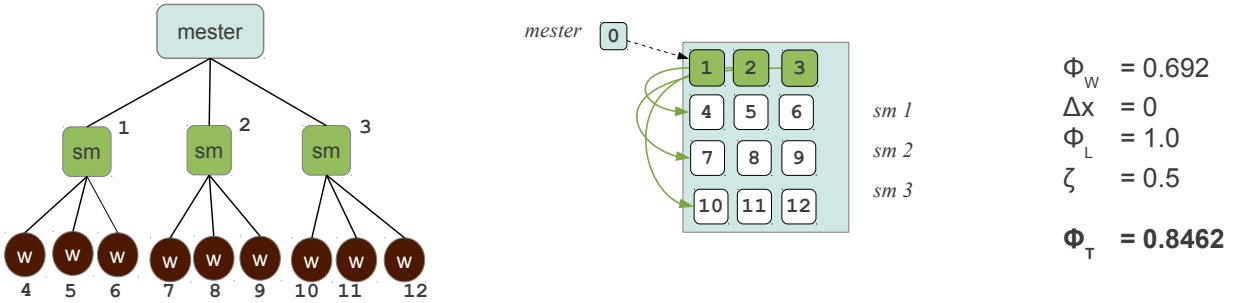
Tekintsünk át néhány konkrét, gyakorlati esetet. Egy sokprocesszoros számítógépen, vagy klaszterben egy hierarchiát számos módon összeállíthatunk. A megoldandó algoritmus Foster-féle particionálásakor egy adott gép adott processzorszámához alakítjuk a részfeladatok felosztását és a kommunikációs struktúrát (*lásd: 3.1. fejezet*). Választott p, p_{sm}, p_w processzorszámnál, a fa struktúra többféleképpen építhető fel. A *30-33. ábrákon* ábrázolt topológiákban $p=13,16,17$ számú processzor használata esetén, a struktúrát különböző kommunikációs kapcsolati struktúrával írhatjuk le. A kiegyensúlyozatlan (a szolga csomópont kiterjesztések száma eltérő) és a kiegyensúlyozott fa struktúrákban a Φ_W és Φ_L potenciál értékeket – az összehasonlíthatóság szempontjából – azonos $\zeta=0.5$ súllyal kombináltuk. A potenciálfüggvény így egyenlő mértékben veszi figyelembe a processzorok száma szerinti Φ_W és a topológiára jellemző Φ_L komponenst. (A konstans korrekciós értéket a függvény többparaméteres jellege is indokolja.)

A *30-31. ábrákról* leolvasható, hogy a struktúrákat $p=13$ processzor használatával, $p_{sm}=3$ segédmester alkalmazásával kiegyensúlyozatlan, illetve kiegyensúlyozott fával alakítottuk ki. A kiegyensúlyozatlan topológia „közepes” $\Phi_T=0.568$ értékkel jelzi, hogy ez a fajta kialakítás nem optimális, míg a kiegyensúlyozott esetben a $\Phi_T=0.846$ „jobb” struktúrát jelez.



hálózati topológia séma *csomóponti mátrix* *potenciál függvény*

30. ábra: Kiegyensúlyozatlan topológia $p=13, p_{sm}=3$

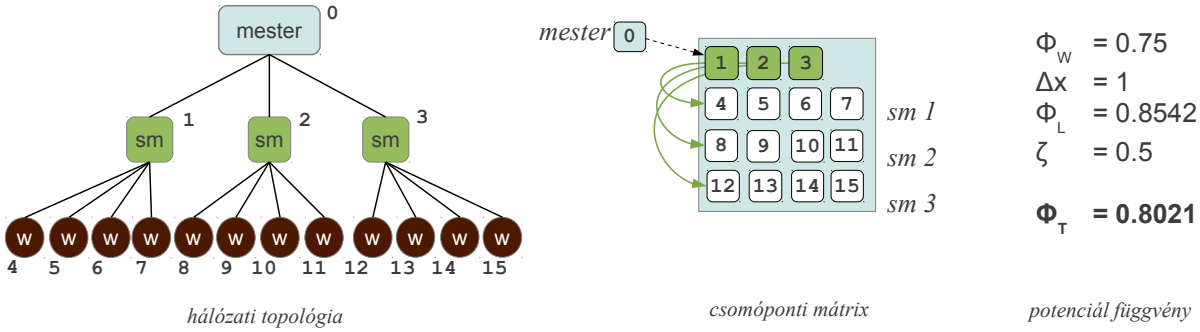


hálózati topológia séma *csomóponti mátrix* *potenciál függvény*

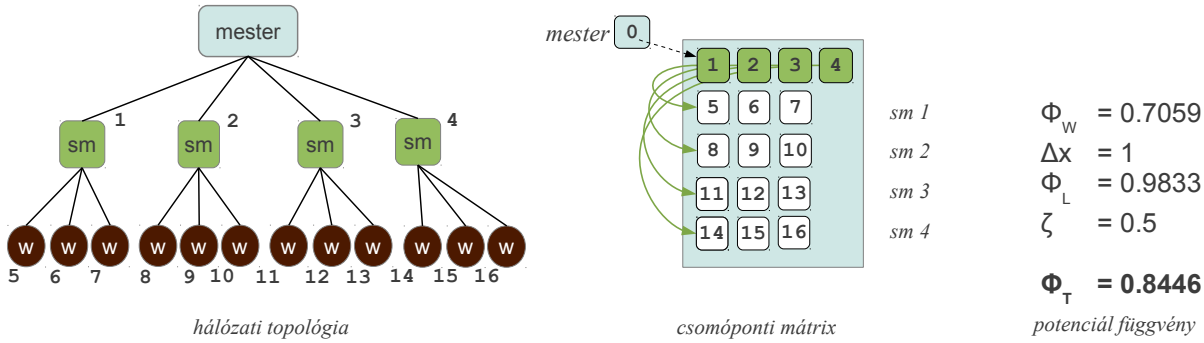
31. ábra: Kiegyensúlyozott topológia $p=13, p_{sm}=3$

A *32* és *33. ábrán* két újabb kiegyensúlyozott struktúrát láthatunk. A $p=16;17$ processzorszámú struktúrát 3, illetve 4 segédmester csomóponttal szimmetrikus rendszerbe szerveztük. A szimmetrikus rendszerek $\Phi_T=0{,}8021;0{,}8446$ potenciál értékei szintén magasabbak értékeket szolgáltatnak. Látható, hogy a processzorszám és segédmester csomópontok számának kisebb megváltoztatása is befolyásolja a potenciál értékeket.

A fenti gyakorlati példákkal egy-egy kiragadott, egyszerű modellt jellemeztünk. A bemutatott feladatokban több paramétert is változtattunk, így a végeredmények összehasonlítása még nem igazolja a potenciálfüggvények használhatóságát. (Megfigyelhető, hogy a kapott $p, p_{sm}, \Phi_W, \Delta x$ értékek eltérnek.) A potenciálfüggvényekkel leírt modell igazolásához, illetve a mérnöki gyakorlatban való hasznosításához szükséges a szélesebb körű vizsgálat és a számítógépes tesztesetek elemzése is.



32. ábra: Kiegyensúlyozott topológia $p=16, p_{sm}=3$

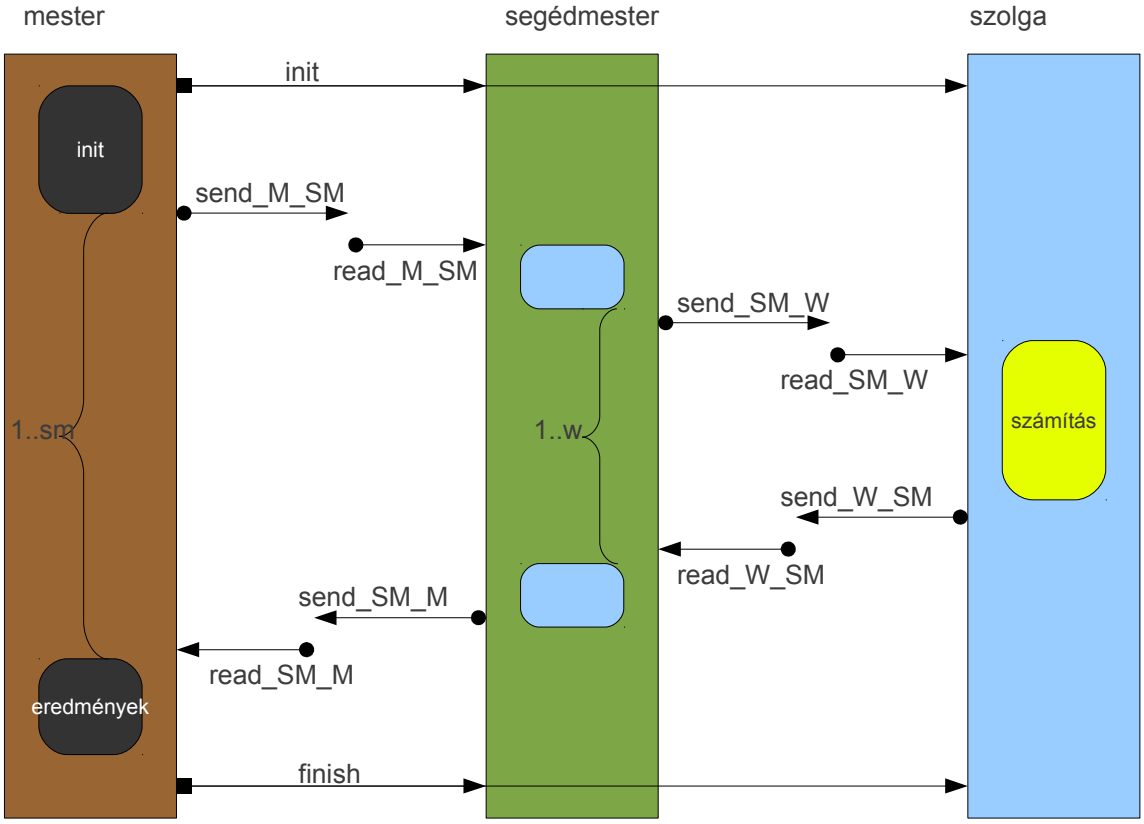


33. ábra: Kiegyensúlyozott topológia $p=17, p_{sm}=4$

A potenciálfüggvények használhatóságának alátámasztására olyan problémát választottunk, amelyek feldolgozása jól párhuzamosítható. A mester és a segédmester csomópontok a feladat darabolásában és menedzselésében, a szolga csomópontok pedig egységes műveletigényű, „atomi” feladatok végrehajtásában vesznek részt. (Horváth, a párhuzamos algoritmusok elemzésével kapcsolatban azt állapítja meg, hogy a választott feladatnak olyannak kell lennie, amely bizonyítottan hibamentes, és helyes komponenseket tartalmaz [Horváth 05]. A topológiától függő futási jellemzők a korábban bemutatott *Algoritmus 2.-Algoritmus 4.* és *Algoritmus 6.-Algoritmus 7.*-nél is megfigyelhetők, de azok összetettsége és nemdeterminisztikus jellege miatt a potenciál módszer igazolására nem alkalmazható.)

Választásunk a számelméletben jól ismert, és a kriptográfiában gyakran alkalmazott Miller–Rabin, illetve Solovay–Strassen prímtesztelő algoritmusok futtatására esett. Az algoritmusok jól definiáltak [Rabin 80] [Goldwasser, Bellare 08], ezek bemutatására külön nem térünk ki. Az algoritmusok „atomi” feladatokként alkalmasak elosztott, illetve párhuzamos környezetben való futtatásra (maguk a prímszámtesztelő algoritmusok szekvenciális részfeladatokként futnak, ezeket tovább nem párhuzamosítjuk).

A hierarchiában a mester csomópont feladata a feladat inicializálása, a csomópontok nyilvántartása és a beérkező eredmények összesítése. A segédmester csomópontok az előre tárolt tesztelés alá vont egész számokat és az ezeken végzendő feladatokat tartják nyilván, illetve a szolga (worker) csomópontokat felkérik a tesztek elvégzésére és az érkező válaszokat (prím / nemprím) tárolják.



34. ábra: Prímtesztelő elosztott párhuzamos algoritmus idődiagram, MPI modell

A *34. ábrán* a feladat párhuzamos megvalósításának idődiagramján a folyamat nyomon követhető. A hierarchia és a párhuzamos program ilyen kialakításával biztosítható, hogy a rendszerben a kommunikációs költségek nagyságrendekkel kisebbek legyenek, mint az szolgáknál futó algoritmus költségei. Így a futtatáskor egy valódi rendszer homogenitása kihasználható, melyen a speciális modellként leírt potenciálfüggvény értelmezhető, és futási idő tanulmányozható.

A példában leírt kis processzorszámú modellek kiterjeszthetők a nagyobb méretű rendszerekre is.

8.3.1. Számítógépes realizáció

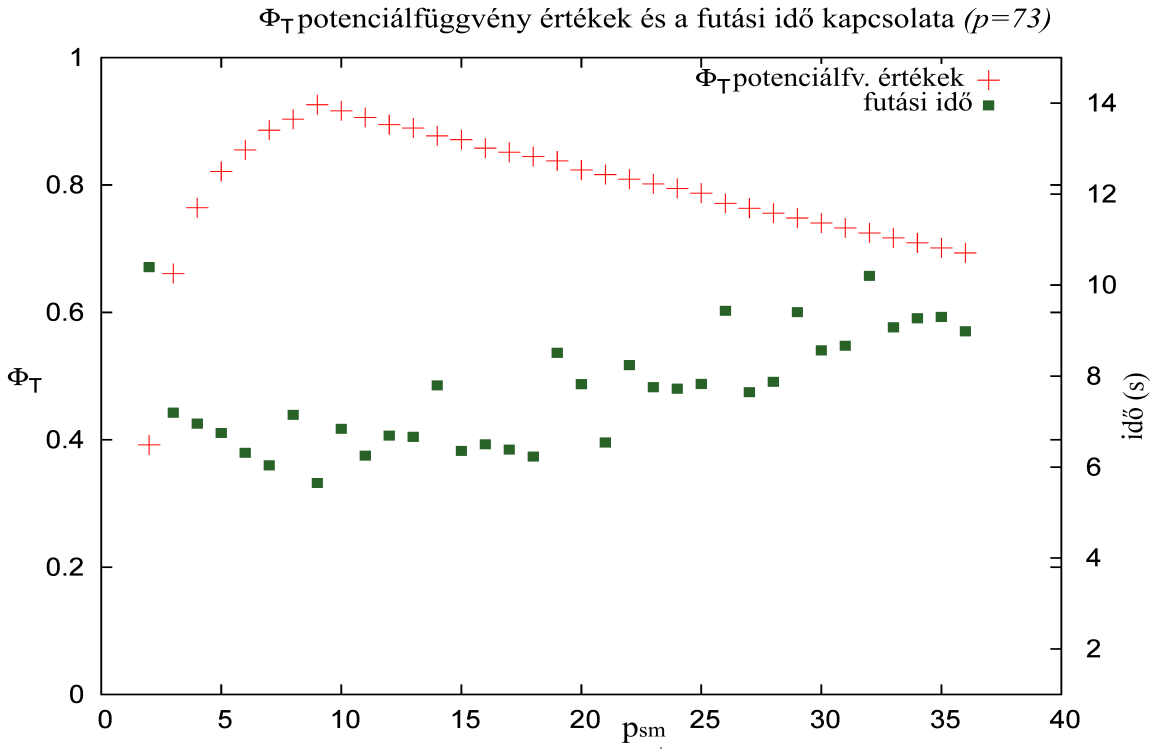
A futtatásokat egy *HP Blade c3000*-es rendszeren végeztük. A gépen a tesztelés során 73 processzort használtunk, melyek között *InfiniBand* hálózati csatolók biztosították a gyors és hibamentes kommunikációt. A hierarchiát leíró fa egy mester, 8 segédmester és az ezekhez tartozó 8x8 szolga processzor esetén a legkiegyensúlyozottabb, bármilyen más felosztásban kiegyensúlyozatlanság, vagy asszimmetria jelentkezik.

A számítógépes teszteknél a 73 processzorból számos topológiát alkothatunk úgy, hogy a segédmester processzorok számát 2 és 36 között változtatjuk, ezekhez pedig lehetőleg egyenletesen elosztva szolga csomópontokat rendelni. Az egyenletes elosztást úgy végeztük el, hogy minden segédmester csomóponthoz közel azonos számú szolga processzort választottunk. Ez több esetben eredményezett kiegyensúlyozott topológiát (pl. $p_{sm}=4,6,8,9,12,\ldots$ esetén), míg más esetekben a segédmesterekhez nem tudtunk azonos számú szolgát rendelni. Az egyenletes elosztással igyekeztünk a Δx értékét minimális értéken tartani, hogy a potenciálfüggvény maximum értékei közelében végezhessük az elemzést. (Mivel a *31. ábrán* jelölthöz hasonló kiegyensúlyozatlan topológiák nagyon eltérően terhelnék a kommunikációs csatornákat, és a segédmester csomópontokra is eltérő nagyságú munkát jelentenének, ezért az ilyen rendszereket nem vettük figyelembe. A potenciálfüggvénynél kikötött homogenitás még szorosan csatolt gépeknél sem lenne elérhető, és már kisebb feladatok esetén is jelentkezik a heterogén rendszerekre jellemző késleltetés, a *„busy-waiting”*).

A *35. ábráról* leolvasható a Miller–Rabin-féle prímtesztelő algoritmus 73 processzoron történő futtatásának időbeli eredménye 3000 prímszám kiértékeléssel. A mérésekkor minden esetben ugyanazt a számsorozatot kellett értékelni, melyek 20 futtatás utáni eredményeit átlagoltuk. Mivel a kiértékelési feladatok egységes műveletigényűek, továbbá a számítógépes rendszer a tesztek alatt végig hibamentes volt, ezért állíthatjuk, hogy a futási idők különbségei a mester és a segédmester processzorok futási idejének és a fa struktúra élei mentén értelmezett kommunikáció költségeinek tudhatók be.

Az ábrán piros kereszt jelzéssel a potenciálfüggvénnyel meghatározott mérőszámot, zöld négyzettel a futási időt ábrázoltuk. A futási idő a teljesen szimmetrikus topológiát használó

esetben volt minimális. Ezen kívül megfigyelhető az a trendszerűség is, hogy a rosszabb potenciál értékekhez általában rosszabb futási idők tartoznak.



35. ábra: Topológiához rendelt Φ_T értékek és számítógépes tesztekben egy algoritmus futási idejének kapcsolata $(p=73,\ p_{sm}=2...36,\ \zeta=0{,}5)$ értékénél

Megjegyezzük, hogy ez a teszt kvantitatív értelemben még nem támasztja alá azt az általánosan várt feltevést, hogy a potenciálfüggvény segítségével meghatározható legyen bármely rendszeren a minimális futási idő, mert a potenciálfüggvény fenti megfogalmazása a gyakorlatban tapasztalható kísérleti eredményeken alapuló empirikus formula. Annyit sikerült elérnünk a modell bevezetésével, hogy egy összetett, valós rendszert néhány egyszerű paraméter segítségével közelítő formulával írtunk le.

A bemutatott potenciálfüggvénnyel egy topológia erőssége jól becsülhető. A potenciálfüggvény értékei az elméleti homogén rendszereket írják le. A valós esetekben mindig jelentkezik némi inhomogenitás, kommunikációs késleltetés, vagy valamilyen gyorsítótározásból érkező mellékhatás. Ez a számítógépes alkalmazáskor a futási eredmények ingadozásán megfigyelhető, illetve az ábráról is leolvasható (futási idők ugrásszerű változása). Ez azzal

magyarázható, hogy a fenti feladathoz használt számítógép nem tekinthető homogén rendszernek. A gép fizikailag 12 db független alaplapból, alaplaponként 2 db 4-4 magos processzorral rendelkezik. A „közel" levő csomópontok közötti adatcsere nagyságrendekkel gyorsabb, mint a „távoli" csomópontok közötti. A párhuzamos program tervezésekor és a vizsgálat során igyekeztünk kizárni az ebből eredeztethető különbségeket, de sajnos nem sikerült teljes mértékben „homogénné" tenni a kísérletet. A megkonstruált potenciálfüggvénytől nem várhatjuk el, hogy egy bonyolult, sok paramétertől függő valós hálózat minősítését ebben a formában egy skalár értékkel pontosan megoldja. Az elvégzett kísérletek azt vetítik előre, hogy a korrelációt a potenciálfüggvény inhomogén rendszerekre történő továbbfejlesztése tudhatná biztosítani.

Célul tűztük ki a későbbi kutatásokban a potenciálfüggvény kiegészítését inhomogén rendszerekre is. Ebben az esetben a topológia minden csomópontját jellemezni kell két további értékkel: a numerikus teljesítménnyel, és a processzorok közötti kommunikációs csatorna sebességével. E két további jellemző segítségével meghatározható az egyes konkrét elosztott feladathoz tartozó optimális topológia. Az új, általános modell figyelembe veheti a segédmester csomópontok teljesítményét és kommunikációs igényét.

8.3.2. A potenciálfüggvény gyakorlati használhatósága

Az elosztott és párhuzamos programozási rendszerekben gyakoriak a hosszú, akár többórás, vagy többnapos sokprocesszoros futtatások [Szebenyi et al. 11]. Napjainkban egyre gyakoribb, hogy a homogén rendszereket más rendszerekkel összekapcsolják és metaszámítógépként, vagy elosztott grid-rendszerként használják. Máskor a homogén rendszerek időben eltérő terhelést és egy időben több feladatot kapnak. Az egyes hardver elemek meghibásodhatnak, csomópontok eshetnek ki. Ezért egy rendszert időben eltérő teljesítményűnek is megfigyelhetünk, amely a laboratóriumi, vagy ipari méréseket befolyásolhatják.

Az általunk leírt potenciálfüggvény jellemzői:

- homogén (egységes költségű kommunikációt használó rendszerben) egy adott processzorszámhoz tartozó hierarchikus topológia erősségét meghatározza,
- segítségével számítógépes gyorstesztek végezhetők: egy adott gép „erősségeit" megadni, pillanatnyi állapotáról információt tud adni.

Jóllehet a topológia megválasztásával csak kisebb mértékű gyorsítást lehet elérni, de segítségével a hosszabb futási idejű (pl. optimalizációs) feladatoknál már jelentős gyorsulás érhető el. Ehhez a fent kifejtett Φ_T potenciálfüggvény „azonnali" becslést tud nyújtani.

Heterogén rendszereket jellemző potenciálok modellje

A fentiekben megfogalmazott potenciálfüggvény csak homogén rendszerekre alkalmazható, így a valós számítógépes rendszerek speciális részhalmazán értelmezett. További kutatási irányt jelent a heterogén sokprocesszoros rendszerekre történő továbbfejlesztése. Ebben a topológiák általános leírására alkalmas modellt szükséges meghatározni, amely alkalmas a heterogén, hibrid, vagy grid-rendszerek számítási teljesítményét és a kommunikációs csatornák sebességét mérni és osztályozni. Ezzel a modellel a párhuzamos számításokra alkalmas statikus (időben nem változó) és dinamikus (időben és térben változó) multiszámítógépes rendszerekhez alkalmazkodó struktúrák potenciálja is mérhetővé válhat.

9. AZ ÉRTEKEZÉS TÉZISEINEK ÖSSZEFOGLALÁSA

A nagy számításigényű feladatokhoz megalkotható párhuzamos algoritmusok vizsgálata a XXI. század egyik jelentős kutatási irányzata. Az egyetemi környezetben elérhető nagy teljesítményű sokprocesszoros számítógépek elsősorban az alapkutatások területén jelentősek. Ezek közül a klasszikus numerikus módszerek széles körben kutatott területnek számítanak. A dolgozat első három fejezetében igyekeztünk a párhuzamos és elosztott rendszerek kialakulását áttekinteni, az algoritmusfejlesztés főbb irányelveinek és legvégül az érintett numerikus matematikai fejezetek tömör és lényegre törő összefoglalását megadni. Ezután az elmúlt években elvégzett kutatómunka eredményeit négy önálló fejezetben fejtettük ki. Az *5. fejezetben* egy konkrét, kutatómunkán alapuló valós-idejű ipari alkalmazást is bemutattunk, ahol a nagy számításigényű feladat párhuzamos és elosztott megoldást kívánt. Az ezután kifejtett témakörök az informatikai alapkutatás témakörébe tartoznak, de az itt leírt eredmények alkalmasak lehetnek a későbbiekben az ipari, vagy gyakorlati felhasználásra.

A *6.* és *7. fejezetben* érintett témakörben, az általános lineáris egyenletrendszerek megoldására jelenleg számos szekvenciális és párhuzamos környezetre készült algoritmus ismert (Gauss, LU, QR, konjugált gradiens, Lánczos, GMRES, MINRES). A jelenleg ismert párhuzamos kódok (BLAS, LAPACK, ScaLAPACK, PETSc) nagy részben a klasszikus szekvenciális algoritmusokban eleve meglévő, párhuzamosítási lehetőségeket használják ki (vektor-skalár, vektor-vektor, mátrix-vektor szorzatok). Néhány párhuzamos környezetre konstruált algoritmus a feladat speciális tulajdonságait használja fel a párhuzamosításhoz, mint pl. a tartomány dekompozíciós módszerek, prekondícionálás. Ahogyan azt a legújabb kutatások is alátámasztják a klasszikus szekvenciális algoritmusok kötött struktúrája miatt a hatékonyság növelése nem oldható meg minden esetben. Ezért sokszor új utakat kell felfedezni, régebben „gyengének” mondott módszereket kell tovább fejleszteni. Máskor a párhuzamos környezetet kell a feladathoz alakítani (processzorszám megválasztása, kommunikációs struktúrák). Kevés olyan algoritmust ismerünk, amely tetszőleges, nagy méretű lineáris egyenletrendszer esetén jól adaptálható az éppen rendelkezésre álló párhuzamos környezetre. Az első és második tézisben ilyen numerikus algoritmusokat mutatunk be, és elvégezzük ezek vizsgálatát és rendre bemutatjuk az algoritmus számítógépes tesztjeinek eredményeit.

A harmadik tézisben egy olyan mérőszám (amortizációs potenciálfüggvény) meghatározására teszünk javaslatot, amely a homogén rendszerekben a hierarchikus szervezésű topológiáját a hatékony teljesítmény elosztás szempontjából jellemzi.

1. tézis:

A rosszul kondicionált általános lineáris egyenletrendszerekre kifejlesztett reziduum-minimalizálási technikán alapuló párhuzamos algoritmusok a szekvenciális algoritmusnál hatékonyabbak. Ezt a vizsgált környezetben kapott futási eredmények alátámasztják.

Nagyméretű általános lineáris egyenletrendszerek megoldására a korábban ismert algoritmusokkal szemben olyanokat is érdemes felhasználni, amelyek szekvenciális környezetben nem hatékonyak, de párhuzamosítással a teljesítményük javítható:

1.1. Megadtam egy általános lineáris egyenletrendszereknél használható és reziduum-minimalizálási technikán alapuló párhuzamos algoritmus leírását, amely alkalmas a sokprocesszoros környezetben való futtatásra *(Algoritmus 2.)* Az algoritmus a sokprocesszoros környezet tulajdonságaihoz igazítható. Az algoritmus az eredeti reziduum minimalizációs algoritmus olyan módosítását tartalmazza, amely lehetővé teszi a független, véletlenszerű irányokból való közelítést, ezzel könnyen adaptálható a különböző számú processzort tartalmazó rendszerekre.

1.2. Számítógépes tesztekkel alátámasztottam, hogy a párhuzamosítással a szekvenciális esethez képest javítható az algoritmus teljesítménye *(Algoritmus 2., Algoritmus 3., Algoritmus 4.)*, mivel párhuzamos környezetben az iteráció speciális, soklépéses iteráció lesz.

1.3. Numerikus kísérletekkel alátámasztottam, hogy az algoritmus nagy kondíciószámú feladatok esetén is konvergens marad *(Algoritmus 3., Algoritmus 4.).*

1.4. Bemutattam olyan numerikus kísérleteket, amelyek alátámasztják, hogy az algoritmusok evolúciós jellegeket mutatnak: a processzorok közötti adatcserék az eredeti szekvenciális algoritmusnál gyorsabb konvergenciához vezetnek. Ezt az mutatja, hogy a párhuzamos végrehajtás során adatcserékkel egyes esetekben elérhető szuper-lineáris gyorsítás is *(Algoritmus 2., Algoritmus 3., Algoritmus 4.).*

Az 1. tézist alátámasztó hivatkozások: [Molnárka, Varjasi 05][Varjasi 06a][Varjasi 06b].

2. tézis

A dolgozatban megadott és vizsgált speciális altér dekompozíciót használó párhuzamos algoritmusok a rosszul kondicionált általános lineáris egyenletrendszerek hatékony megoldó módszere.

Az *1. tézisben* megfogalmazott párhuzamos algoritmuscsalád és modell tovább fejleszthető, hatékonysága javítható – az eredeti lineáris egyenletrendszerekben történő *speciális alterek* definiálásával – kisebb feladatokra bontással. *(Az altér dekompozíció kifejezést mi speciális értelemben használjuk, ugyanis a különböző alterek kiválasztása a reziduum vektorok terében történik. Azt a megkötést sem használjuk, hogy az alterek direktösszege kiadja a teljes teret, azaz az alterek diszjunkt és teljes jellege nincs kikötve.)* Ezen speciális alterekben részfeladatok a sokprocesszoros környezetben párhuzamosan oldhatók meg. Ez az algoritmus osztály alkalmas általános, jellemzően nagy kondíciószámú problémák megoldására is:

2.1. Az *1. tézisben* szereplő algoritmust általánosítottuk úgy, hogy az a reziduum vektorok terében értelmezett altér-dekompozíciókkal lett kiegészítve. Ennek realizálásához párhuzamos algoritmust definiáltam, amely a gyorsabb konvergencia mellett még azzal a tulajdonsággal is rendelkezik, hogy sokprocesszoros gépeken jól skálázható *(Algoritmus 6.)*.

2.2. Numerikus kísérletekkel alátámasztottam, hogy a definiált párhuzamos algoritmus hatékonysága javítható oly módon is, hogy ha az egyes processzorok minden egyes iterációs lépésben a véletlen vektorokat az addigi legjobb megoldás közeléből veszik, a konvergencia gyorsul *(Algoritmus 7.)*.

2.3. A speciális altér dekompozíciós módszerrel az eredeti – akár magas kondíciószámú – feladat olyan részfeladatokra bomlik, amelyek kondíciószáma alacsony, így az ismert megoldó algoritmusokkal (szekvenciális, és/vagy párhuzamos algoritmusok) a részfeladatok hatékonyan számíthatók *(Algoritmus 6., Algoritmus 7.)*.

2.4. A speciális altér dekompozíciót használó párhuzamos algoritmusok az *1.4.* tézisponthoz hasonlóan evolúciós jellegűek, a párhuzamos végrehajtás során adatcserékkel egyes esetekben szuper-lineáris gyorsítás érhető el.

A 2. tézist alátámasztó hivatkozások: [Molnárka, Varjasi 10][Molnárka, Varjasi 11a].

3. tézis

Homogén és inhomogén rendszerekre is megadható olyan számítási modell, amely egy mérőszám alapján a hierarchikus topológiák hatékonyságát jellemzi. Homogén rendszerekre ezt számítógépes tesztek is alátámasztják.

A számítást végző hardver eszközökön alkalmazott topológia nagyban befolyásolja a megoldás futtatási hatékonyságát. Az egyes számítási csomópontok üresjárat-, vagy túlterhelés vizsgálatával, illetve a kommunikációs csatornák elemzésével meghatározhatók olyan modellek, amelyek az adott hardver kiépítését jellemzik. Az egyes számítási rendszereket leíró topológiák vizsgálatával és sokprocesszoros rendszerek működésének elemzésével az alábbiakat mutattam be:

3.1. Számítógépes tesztekkel bemutattam, hogy az *1. tézisben* vizsgált *Algoritmus 2.* végrehajtása egyenrangú csomóponti topológiával (gyűrű, hálózat) rendelkező heterogén hálózatokon nem hatékony. A hierarchikus topológiákkal leírt kommunikációs modellben (csillag, n-ágú fa) *Algoritmus 4.* skálázható.

3.2. Az *Algoritmus 7.* különböző realizációinak futtatásával megmutattam, hogy az adatcserékkel működő párhuzamos környezetben az optimalitás minősítésére csak a feladat megoldásának ideje használható. (A reziduum hiba szerint gyorsabb konvergenciájú – iterációk száma szerinti konvergencia sebesség – algoritmus hosszabb idő alatt szolgáltat eredményt. Lásd: *4.* és *5. táblázat*, szuper-lineáris gyorsítás).

3.3. Definiáltam egy olyan Φ potenciál függvényt (ún. amortizációs potenciál), amely általános fa struktúrával leírható hierarchikus topológiákon a terhelés elosztást jellemzi és méri. A potenciál függvénnyel egy-egy adott sokprocesszoros rendszer jellemezhető. Számításokkal előre jelezhető, hogy egy adott feladathoz milyen hierarchia szolgáltathatja a leghatékonyabb futást.

A 3. tézist alátámasztó hivatkozások: [Varjasi 06b][Varjasi 07][Molnárka, Varjasi 10][Molnárka, Varjasi 11b]

10. IRODALOMJEGYZÉK

[Abaffy et al. 84] J. Abaffy, Ch. Broyden, E. Spedicato: *A Class of Direct Methods for Linear Systems.* Numerische Mathematik 45. 1984. pp. 361–376.

[Amdahl 67] Amdahl, Gene : *Validity of the Single Processor Approach to Achieving Large-Scale* Computing Capabilities. AFIPS Conference Proceedings, 1967. pp. 483–485.

[Bahalov 77] Bahalov, N. Sz.: *A gépi matematika numerikus módszerei.* Műszaki Kiadó, Budapest, 1977. p. 551. ISBN 963 10 1753 2

[Basermann et al. 97] A. Basermann, B. Reichel, C. Schelthoff: *Preconditioned CG methods for sparse matrices on massively parallel machines.* Parallel Computing 23. 1997. pp. 381 – 398.

[Becciani et al. 97] U. Becciani, R. Ansaloni, V. Antonuccio-Delogu, G. Erbacci, M. Gambera, A. Pagliaro: A *parallel tree code for large N-body simulation: dynamic load balance and data distribution on a CRAY T3D system.* In Computer Physics Communications 106, 1997. pp. 105–113.

[Berman et al. 03] *Grid computing.* (Ed: Fran Berman, G. Fox, T. Hey) : Wiley & Sons, 2003. ISBN 0 470-85 319-0

[Bickson 09] D. Bickson: *Gaussian Belief Propagation: Theory and Application.* Ph.D. Thesis. Hebrew University of Jerusalem, 2009. http://arxiv.org/abs/0811.2518v3

[Blum et al. 98] L. Blum, F. Cucker, M. Shub, S. Smale: *Complexity and real computation.* Springer, New York, 1998. ISBN 0-378-98 281-7

[Bosilca et al. 12] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Herault, Piotr Luszczek, and Jack Dongarra: *Dense Linear Algebra on Distributed Heterogeneous Hardware with a Symbolic DAG Approach.* In: Scalable Computing and Communications, Theory and Practice, Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya, to appear John Wiley & Sons, 2012. http://www.netlib.org/utk/people/JackDongarra/PAPERS/scal_comp.pdf

[Boysen 08] Nils Boysen, Malte Fliedner: *A versatile algorithm for assembly line balancing.* European Journal of Operational Research, Volume 184, Issue 1. 2008. pp. 39-56.

[Ciarlet 06] Philippe G. Ciarlet: *Introduction à l'analyse numérique matricielle et à l'optimisation*, MASSON, Paris, (nouvelle présentation) 2006. p. 259. ISBN 2 10 050 808 3.

[Chen 09] Y. Chen, Y. Deng: *A detailed analysis of communication load balance on BlueGene supercomputer.* Computer Physics Communications, 180. 2009. pp. 1251 – 1258.

[Chen 10] X. H. Sun, Y. Chen: *Reevaluating Amdahl's law in the multicore era.* J. Parallel Distributed Computations 70. 2010. pp. 183–188.

[Choi 06] Sou-Cheng (Terrya) Choi: *Iterative methods for singular linear equations and least-squares problems.* PhD. Dissertation, Stanford University, 2006. p. 116. http://www.stanford.edu/group/SOL/dissertations/sou-cheng-choi-thesis.pdf

[Christopher 2000] T. W. Christopher, G.K. Thiruvathukal *High-Performance Java Platform Computing: Multithreaded and Networked Programming.* Prentice Hall PTR, 2000. ISBN 978-0 130 161 642

[Cormen et al. 01] Thomas H. Cormen, Charles Leiserson, Roland Rivest: *Algoritmusok.* Műszaki Könyvkiadó, Budapest, 2001. p. 884. ISBN 963-16-3029-3

[Crichlow 03] Joel M. Crichlow: *Elosztott rendszerek.* Kiskapu Kft. Budapest 2003. p. 290. ISBN 9 639 301 647

[Davis 94] T.A. Davis, I. S. Duff: *An unsymmetric-pattern multifrontal package (umfpack).* Tech. Report TR-95-004, Computer and Information Sciences, Department, University of Florida, Gainesville, FL, 1995. http://www.cise.ufl.edu/tr/DOC/REP-1994-159.ps

[Dongarra et al. 84] J. J. Dongarra, F. G. Gustavson and A. Karp: *Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine.* SIAM Review Vol. 26, No. 1 (jan., 1984), pp. 91 – 112.

[Dongarra 96] J. Dongarra and D. Walker: *MPI: A Standard Message Passing Interface.* Supercomputer 12(1). 1996. pp. 56 – 68. ISSN 0168-7875.

[Dongarra et al. 07] Jack Dongarra, J. Pineau, Y. Robert, Z. Shi, F. Vivien: *Revisiting Matrix Product on Master-Worker Platforms.* Parallel and Distributed Processing Symposium 2007. IPDPS 2007, IEEE International. pp. 1 – 8. DOI: 10.1109/IPDPS.2007.370466.

[Dongarra 2000] J.J. Dongarra, V. Eijkhout: *Numerical linear algebra algorithms and software.* Journal of Computational and Applied Mathematics 123. 2000. pp. 489 – 514.

[Duff 99] I. S. Duff, H. A. van der Vorst: *Developments and trends in the parallel solution of linear systems.* Parallel Computing 25. 1999. pp. 1931 – 1970.

[Eijkhout 06] Victor Eijkhout, Julien Langou, and Jack Dongarra: *Parallel Linear Algebra Software.* In: Parallel Processing for Scientific Computing, SIAM 2006, ISBN 9 780 898 716 191 DOI: http://dx.doi.org/10.1137/1.9780 898 718 133.ch13

[Foster 95] Ian Foster: *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison Wesley 1995. ISBN: 0 201 575 949 http://www.mcs.anl.-gov/~itf/dbpp/text/book.html

[Futó 03] *Mesterséges intelligencia.* Futó Iván (szerk.) Aula Kiadó, Budapest, 2003. p .1007. ISBN: 9 789 639 078 994

[Gustafson 88] J. L. Gustafson: *Reevaluating Amdahl's law*, Communications of ACM, 1988. Doi:

10.1145/42411.42415.

[Hageman 81] Louis A. Hageman, Davis M. Joung: *Applied Iterative Methods.* Computer Science and Applied Mathematics, Academic Press, 1981. ISBN 0-486-43477-X

[Hanselman 01] Hanselman, D.C., Littlefield B.: *Mastering Matlab 6: a comprehensive tutorial and reference.* Prentice Hall New Jersey 2001. ISBN 0130194689

[Henrici 85] Peter Henrici: *Numerikus analízis.* Műszaki Könyvkiadó Budapest, 1985. p. 370. ISBN 963 10 6419 0

[Herlihy 08] Maurice Herlihy, Nir Shavit: *The Art of Multiprocessor Programming.* Elsevier, 2008. p. 529. ISBN 978-0-12-370591-4

[Hord 93] Hord, R. Michael: *Parallel Supercomputing in MIMD Architectures.* CRC Press, 1993. p. ISBN 0-8493-4417-4

[Gáspár 07] Gáspár Csaba: *Numerikus módszerek.* Széchenyi István Egyetem Győr, 2007. (oktatási segédlet), http://rs1.szif.hu/~gasparcs/numo2.hlp

[Gilbert 94] J. Gilbert, D. Kershaw, *Large-Scale Matrix Problems and the Numerical Solution of Partial Differential Equations,* Advances in Numerical Analysis, Vol. III. Clarendon Press, Oxford, 1994.

[Hellner 78] Don Heller: *A Survey of Parallel Algorithms in Numerical Linear Algebra.* SIAM Review Vol. 20, No. 4. (Oct., 1978), pp. 740 – 777.

[Hestens, Steifel 52] M.R. Hestens, E. Steifel: *Methods os Conjugate Gradients for Solving Linear Systems.* Journal of Research of National Bureau for Standards, Vol 49. No. 6, 1952.

[Horváth 05] Horváth Zoltán: *Párhuzamos és elosztott programozás.* In: Bevezetés a programozásba. egyetemi jegyzet ELTE Informatikai Kar, 2005. http://www.inf.elte.hu/karunkrol/digitkonyv/Jegyzetek2004/ParhProg.pdf

[Iványi 03] Iványi Antal: *Párhuzamos algoritmusok.* ELTE Eötvös Kiadó, Budapest, 2003. p. 335. ISBN 963 463 590 3

[Iványi 05a] Iványi Antal és Claudia Leopold: Párhuzamos számítások. In: Informatikai algoritmusok I. Iványi Antal (szerk.) ELTE Eötvös Kiadó, Budapest, 2004. pp. 222–267. ISBN 963 463 664 0. http://compalg.inf.elte.hu/~tony/Elektronikus/Informatikai/Infalg1E.pdf

[Iványi 05b] Iványi Antal (szerk.): *Informatikai algoritmusok II.* ELTE Eötvös Kiadó, Budapest, 2005. p. 1580. ISBN 963 463 775 2

[Iványi 07] Iványi Antal (szerk.): *Algorithms of Informatics vol2.* MondAt Kiadó, Budapest, 2007. p. 1147. ISBN 963 463 87596 2-7

[Jordan 86] M. I. Jordan : *An introduction to linear algebra in parallel distributed processing.* In: D.E. Rumelhart and J.L. McClelland (Eds.): Parallel distributed processing: explorations in the microstructure of cognition, Vol. 1. MIT Press Cambridge, MA, USA, 1986. ISBN:0-262-68 053-X

[Galántai 99] Galántai, A.: *Parallel ABS Projection Methods for Linear and Nonlinear Systems with Block Arrowhead Structure.* Computers and Mathematics with Applications 38. 1999. pp. 11 – 17.

[Galántai 03] Galántai Aurél: *Projection Methods for Linear and Nonlinear Equations.* MTA doktori disszertáció. p. 176. http://www.uni-obuda.hu/users/galantai/mtadokt3j.pdf

[Galántai-Jeney 05] Galántay Aurél és Jeney András: *Tudományos számítások.* In: Informatikai algoritmusok I. Iványi Antal (szerk.) ELTE Eötvös Kiadó, Budapest, 2004. pp. 717–773. ISBN 963 463 664 0. http://compalg.inf.elte.hu/~tony/Elektronikus/Informatikai/I17E.pdf

[Golub et al. 94] G. Golub, A. Greenbaum, M. Luskin, eds., *Recent Advances is Iterative Methods*, The IMA Volumes in Math. and its Applications Vol.60., Springer Verlag, 1994.

[Goldwasser, Bellare 08] Shafi Goldwasser, Mihir Bellare: *Lecture Notes on Cryptography.* MIT Computer Science and Artifcial Intelligence Laboratory 2008. p. 289. *http://cseweb.ucsd.edu/~mihir/papers/gb.pdf*

[Kacsuk 02] Kacsuk Péter: *Magyar Szuperszámítógép GRID projekt.* Networkshop konferencia 2002, Eger.

[Kacsuk 06] Peter Kacsuk, Gergely Sipos: *Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal.* Journal of Grid Computing 2006. Vol. 3. pp. 221 – 238. DOI: 10.1007/s10 723-005-9012-6

[Kallós 04] G. Kallós: *Párhuzamos programozás*: elektronikus egyetemi jegyzet : Széchenyi István Egyetem, Győr. 2004. http://rs1.szif.hu/~kallos/pp/pp.pdf

[Kanti 11] G. Kanti: *Hierarchikus szervezésű párhuzamos programozási modellek MPI rendszerben*, Széchenyi István Egyetem Győr, 2011. p. 63.

[Kelley 95] C.T. Kelley: *Iterative Methods for Linear and Nonlinear Equations.* North Carolina State University, 1995. http://www.siam.org/books/textbooks/fr16_book.pdf

[Kepner 09] Jeremy Kepner: *Parallel MATLAB for Multicore and Multinode Computers.* SIAM 2009. p. 253. ISBN 978-0-898 716-73-3

[Khanna 10] Gaurav Khanna, Justin McKennon: *Numerical modeling of gravitational wave sources accelerated by OpenCL.* Computer Physics Communications, Volume 181, Issue 9, September 2010, pp. 1605-1611.

[Király 11] Z. Király: *Adatstruktúrák*, ELTE, 2011. p.92; http://www.cs.elte.hu/~kiraly/Adatstrukt-

urak.pdf

[Korn 75] Granino A. Korn, T. M. Korn: *Matematikai kézikönyv műszakiaknak.* Műszaki Könyvkiadó Budapest, 1975. p. 995. ISBN 963 10 0673 5

[Körfgen 06] Bernd Körfgen and Inge Gutheil: *Parallel Linear Algebra Methods.* Computational Nanoscience: J. Grotendorst, S. Blugel, D. Marx (Eds.), John von Neumann Institute for Computing, Julich, NIC Series, 2006. Vol. 31, pp. 507–522. ISBN 3-00-017350-1

[Környei et al. 10] L. Környei, G. Kallós, D. Fülep: *Parallel Computations on the Blade Server at Széchenyi University – Classic Problems in Practice.* Acta Technica Jaurinensis Vol. 3. No. 1. 2010. pp. 111–125.

[Kulish 08] Ulrich Kulisch: *Computer Arithmetic and Validity.* Walter de Gruyter, Berlin, 2008. ISBN 978-3-11-020318-9

[Lencse 01] Lencse, G: *Traffic-Flow Analysis for Fast Performance Estimation of Communication Systems.* Journal of Computing and Information Technology, Vol. 9, No. 1, March 2001. pp. 15-27.

[Lin 2000] W.-M. Lin, W. Xie: *Load-skewing task assignment to minimize communication conflicts on network of workstations.* Parallel Computing 26, 2000. pp. 179 – 197.

[Lynch 02] Nancy Ann Lynch: *Osztott algoritmusok.* Kiskapu, Budapest, 2002. p. 781. ISBN 963 9301 03 5

[Moler 72] Cleve B. Moler: *Matrix Computations with Fortran and Paging.* Communications of the ACM, Vol. 15, April 1972, pp. 268 – 270.

[Morrison 03] Richard S. Morrison: *Cluster Computing: Architectures, Operating Systems, Parallel Processing & Programming Languages.* 2003. p. 150. http://www.scribd.com/doc/16632867/Cluster-Computing-Architectures-Operating-Systems-Parallel-Processing-Programming-Languages-v24-Apr-2003-by-Laxxuss;

[Morrison 03b] Richard S. Morrison: *Cluster Computing.* http://www.ace.ual.es/~jaberme/docsppal/cluster/CLSTR_CMPTNG_THEORY.pdf

[Matsumoto 98] M. Matsumoto and T. Nishimura: *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.* ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January, 1998. pp. 3 – 30.

[Molnárka et al. 96] Molnárka Győző, Gergó Lajos, Wettl Ferenc, Horváth András, Kallós Gábor: *A Maple V és alkalmazásai.* Springer, Budapest, 1996. ISBN 9638455896

[Molnárka, Miletics 05] G Molnárka, E Miletics: A *Genetic Algorithm for Solving General System of Equations.* In: Proceedings of the Third Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence. Herlany, Szlovákia, 2005. pp. 467 – 473. ISBN 963 7154 35

3

[Molnárka 06a] G. Molnárka: *A scalable parallel algorithm for solving general linear system equations*, 77th GAMM annual meeting 2006, Berlin, 2006. pp.441.

[Molnárka 06b] G. Molnárka, *A Scalable Parallel Genetic Algorithm for Solving Linear Systems.* In: B.H.V. Topping, G. Montero, R. Montenegro (editors), Proceedings of the Fifth International Conference on Engineering Computational Technology, Civil-Comp Press, Stirlingshire, UK, Paper 95, 2006. Doi:10.4203/ccp.84.95

[Muka, Lencse 10] Muka, L; G. Lencse: *Improving the Spatial Distribution Algorithm of the Traffic-Flow Analysis.* Acta Technica Jaurinensis Vol. 3. No. 2. (2010) pp. 161-173.

[Nagy, Molnárka 07] Attila Nagy, Győző Molnárka: *Informatics and measurement technical research in the KKK* at *Tech4Auto 2007: „AUTOPOLIS Economy – Co-operation – Development" Regional Economic Development Conference Győr, Hungary, 20th September 2007.*

[Oishi et al. 08] Shin'ichi Oishi, Takeshi Ogita, Siegfried M. Rump: *Iterative Refinement for Ill-conditioned Linear Equations.* 2008 International Symposium on Nonlinear Theory and its Applications NOLTA'08, Budapest, Hungary, September 7-10, 2008. http://www.oishi.info.waseda.ac.jp/~oishi/papers/OiOgRu08.pdf

[Paige, Saunders 75] C. C. Paige , M. A. Saunders: *Solution of Sparse Indefinite Systems of Linear Equations.* SIAM Journal on Numerical Analysis 12.(4). 1975. pp. 617–629.

[Paige, Saunders 82] C. C. Paige , M. A. Saunders: *LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares.* ACM Transactions on Mathematical Software, Vol. 8. No. 1, 1982. pp. 43 – 71.

[Pashos et al. 09] G. Pashos, M. Kavousanakis, A. N. Spyropoulos, J. A. Palyvos, A. G. Boudouvis: *Simoultanous solution of large-scale linear systems and eigenvalue problems with parallel GMRES method.* Journal of Comput. and Applied Math. Vol. 227. 2009. pp. 196 – 205.

[Pintér 08] István Pintér: *Noise suppression using non-linear speech model. P*ollack Periodica, Volume 2, Supplement 1, 29. 2008. pp. 121-133. ISSN: 1788-3911

[Preis 99] Tobias Preis, Peter Virnau, Wolfgang Paul and Johannes J Schneider: *Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets.* 2009. New J. Phys. p. 21.

[Press et al. 07] William H. Press, S. Teukolsky, W. Vettering, B. Flannery: *Numerical Recepies: The art of scientific programming.* Cambridge University Press, Third edition 2007. p. 1235. ISBN 978-0-521–88 068-8

[Rabin 80] Michael O. Rabin: "*Probabilistic algorithm for testing primality*", Journal of Number Theory 12 (1) 1980. pp. 128 – 138. Doi:10.1016/0022-314X(80)90 084-0

[Ramachandrana 03] Vijaya Ramachandrana, Brian Graysonb, Michael Dahlin: *Emulations between QSM, BSP and LogP: a framework for general-purpose parallel algorithm design.* Journal of Parallel and Distributed Computing Vol.63, Issue 12, December 2003, pp. 1175 – 1192. Doi: 10.1016/j.jpdc.2003.04.001

[Riedel et. al 09] Riedel, M., Laure, E., Soddemann, T., Field, L., Navarro, J., et al., *Interoperation of world-wide production e-Science infrastructures.* Concurrency and Computation: Pract. Exper., 21, 2009. pp.961–990. Doi: 10.1002/cpe.1402

[Saad, Schultz 86] Y. Saad, M. Schultz: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM J. ScI. STAT. COMPUT. vol. 7, no. 3, July 1986

[Snir et al. 95] Marc Snir, Steve W. Otto, David W. Walker, Jack Dongarra, Steven Huss-Lederman: *MPI: The Complete Reference.* MIT Press, Cambridge, USA, 1995. ISBN: 0 262 691 841

[Sosonkina 02] Maria Sosonkina, D. C. S. Allison and L. T. Watson: *Scalability analysis of parallel GMRES implementations.* Parallel Algorithms and Applications Volume 17, Issue 4, 2002. pp. 263-284 Doi:10.1080/0 149 573 020 8 941 444

[Spedicato 2000] Emilio Spedicato, Zunquan Xia , Liwei Zhang: A*BS algorithms for linear equations and optimization.* Journal of Computational and Applied Mathematics 124. 2000. pp. 155 – 170.

[Schliephake 11] M. Schliephake, X. Aguilar, E. Laure: *Design and Implementation of a Runtime System for Parallel Numerical Simulations on Large-Scale Clusters.* In: Procedia Computer Science 4., 2011. pp. 2105–2114.

[Stoyan 02] Stoyan Gisbert, Takó Galina: *Numerikus módszerek I.* Typotex, Budapest, 2002. p. 460. ISBN 963 932 620 8

[Szebenyi et al. 11] Szebenyi Zoltán, Felix Wolf, Wylie, Brian J. N.: *Performance Analysis of Long-running Applications.* In: Proc. of the 25th IEEE Int'l Parallel Distributed Processing Symposium (IPDPS) PhD Forum, Anchorage, AK, USA, 2011. pp. 2100 – 2103. ISBN 978-0-7695-4385-7

[Szeberényi 06] Szeberényi Imre: *A GRID technológia és kutatás hazai és nemzetközi eredményei.* IX. Országos Neumann Kongresszus, Győr, 2006.

[Strohmaier et al. 05] Strohmaier E., Dongarra J.J., Meuer H.W., Simon H.D.: *Recent trends in the marketplace of high performance computing*. Parallel Computing, Vol. 31. (3-4), 2005. pp. 261-273. Doi: 10.1016/j.parco.2005.02.001

[Tanenbaum 01] Andrew S. Tanenbaum: *Számítógép architektúrák.* Panem, Budapest 2001. p. 816. ISBN 9 789 635 454 570

[Tanenbaum 04] Andrew S. Tanenbaum, Maarten van Steen: *Elosztott rendszerek: alapelvek és paradig-*

mák. Panem, Budapest, 2004. p. 872. ISBN 963 545 387 6.

[Tarjan, Sleator 85] D.D.Sleator, R. E. Tarjan: *Self-Adjusting Binary Search Trees.* in Journal of the Association for Computing Machinery. Vol. 32, No. 3, 1985. pp. 652 – 686.

[Tarjan 85] R. E. Tarjan: *Amortized Computational Complexity.* in SIAM Journal on Algebraic and Discrete Methods, Vol. 6. No. 2. 1985. pp. 306 – 317.

[Thompson 02] Chris J. Thompson, Sahngyun Hahn, and Mark Oskin: U*sing Modern Graphics Architectures for General-Purpose Computing: A Framework and Analysis.* International Symposium on Microarchitecture (MICRO), Turkey, Nov. 2002

[Tomov et al. 10] Stanimire Tomov, Rajib Nath, Hatem Ltaief, and Jack Dongarra: *Dense Linear Algebra Solvers for Multicore with GPU Accelerators.* Proceedings of IPDPS 2010: 24th IEEE I nternational Parallel and Distributed Processing Symposium, Atlanta, GA, April 2010.

[Trefethen 97] Trefethen, Lloyd N.; Bau, D.: *Numerical Lienar Algebra.* SIAM 1997. p. 361. ISBN 0-89 871-361-7.

[Upton et al. 94] Roger Upton, Bruel&Kjaer, Seishiro Daimon: *Use of real-time Zwicker loudness in automotive noise measurements.* Matsushita Inter-techno Co., Ltd., JSAE Review, Volume 15, Issue 3, July 1994, pp 188–191.

[Van de Velde 94] Eric Van de Velde: *Concurrent scientific computing.* Springer, New York, 1994. ISBN 0-378-94 195-9

[Vib 85] *Vibration analyser uses spectral analysis.* International Journal of Fatigue, Volume 7, Issue 4, October 1985, Page 228. Bruel and Kjaer (UK) Ltd. http://dx.doi.org/10.1016/0142-1123(85)90 059-3

[Wang et al. 07] Y.S. Wang, C.-M. Lee, D.-G. Kim and Y. Xu *: Sound-quality prediction for nonstationary vehicle interior noise based on wavelet pre-processing neural network model.* Journal of Sound and Vibration, Volume 299, Issues 4-5, 6. 2007, pp. 933-947.

[Weiss 01] Weiss Y, Freeman WT.: *Correctness of belief propagation in Gaussian graphical models of arbitrary topology.* Neural Computation, 2001. Vol. 13. No.10. pp. 2173-2200. http://dx.doi.org/10.1162/0 899 766 017 50 541 769

[Yasar 01] O. Yasar: *New trends in high performance computing.* Parallel Computing 27. 2001. pp.1 – 2.

[Yero 07] E. J. H. Yero, M. A. A Henriques: *Speedup and scalability analysis of Master-Slave applications on large heterogeneous clusters.* Journal of Parallel and Distributed Computing 67. 2007. pp. 1155 – 1167.

[Zwicker et al. 99]Hugo Fastl, Eberhard Zwicker: *Psychoacoustics: Facts and Models.* 2nd Edition, Springer-Verlag Berlin. 1999. pp 111-149. ISBN-10 3-540-65 063-6

Internetes hivatkozások

[BEO] Beowulf cluster project: http://www.beowulf.org/overview/history.html

[BOINC] Berkeley Open Infrastructure for Network Computing (desktop grid): http://boinc.berkeley.edu

[BOINCST] BOINC statistics: http://www.allprojectstats.com/po.php?projekt=0

[CUDA] CUDA parallel programming platform: http://www.nvidia.com/object/cuda_home_new.html

[EARTH] Earth Simulator Center – Japan Agency for Marine-Earth Science and Technology. http://www.es.jamstec.go.jp/esc/eng/

[EUDG] European DataGrid Projekt: http://eu-datagrid.web.cern.ch/eu-datagrid/

[GTRN] Global Terrabit Research Network. http://www.indiana.edu/~gtrn/

[IWAY] Information-Wide-Area-Year (I-WAY). http://www.nitrd.gov/pubs/bluebooks/1997/i-way.html

[IBMp690] IBM eServer, (pSeries 690) technical details http://www-03.ibm.com/servers/eserver/pseries/hardware/highend/p690.html

[MATH] Wolfram MATHEMATICA: http://www.wolfram.com/mathematica/

[MATRIXMARKET] Matrix Market: *A visual repository of test data for use in comparative studies of algorithms for numerical linear algebra:* http://math.nist.gov/MatrixMarket/

[MSZGRID] Magyar szuperszámítógép grid. http://mszgrid.iif.hu/

[NUMPY] NumPy, fundamental package for scientific computing in Python: http://docs.scipy.org/doc/numpy/numpy-user.pdf

[OPENCL] OpenCl, open standard for heterogenous programming system: http://www.khronos.org/opencl/

[OPENMP] OpenMP specification: http://openmp.org/wp/openmp-specifications/

[PBS] PBS, Portable Batch System: http://www.pbsworks.com/documentation/support/PBSProUserGuide11.3.pdf

[PGRADE] P-GRADE: Parallel Grid Run-time and Application Development Environment http://www.lpds.sztaki.hu/pgrade/

[PETSc] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik and Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith and Hong Zhang: *PETSc web page*, http://www.mcs.anl.gov/petsc, 2011.

[PLASMA] Parallel Linear Algebra Software for Multicore Architectures

http://icl.cs.utk.edu/projectsfiles/plasma/pdf/users_guide.pdf

[SuperLU] SuperLU: general direct solution of large, sparse, nonsymmetric systems of linear equations. http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

[Top500] Top500 Supercomputing sites: http://www.top500.org; http://www.top500.org/static/lists/2011/11/TOP500_201111.xls; http://www.top500.org/static/lists/2011/11/TOP500_201111_Poster.pdf

[ZOLTAN] Zoltan: Parallel Partitioning, Load Balancing and Data-Management Services http://www.cs.sandia.gov/Zoltan/

A jelölt disszertációban hivatkozott munkái

[Molnárka, Varjasi 05] Molnárka Győző, Varjasi Norbert: *Egy párhuzamos algoritmus általános lineáris egyenletrendszerek megoldására.* In: Pethő A, Herdon M. (szerk.) Informatika a felsőoktatásban 2005 konferencia. Debrecen, Magyarország, 2005. Debrecen: Debreceni Egyetem, Paper E35. ISBN:963 472 909 6

[Molnárka, Varjasi 10] G. Molnárka, N. Varjasi: *A simultaneous solution for general linear equations on a ring or hierarchical cluster.* Acta Technica Jaurinensis Vol. 3. No. 1., 2010. pp. 65 – 73.

[Molnárka, Varjasi 11a] G. Molnárka, N. Varjasi: *A Simultaneous Solution for General Linear Equations with Subspace Decomposition*, In: Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Civil-Comp Press, 2011. ISBN 978-1-905088-43-0.

[Molnárka, Varjasi 11b] G. Molnárka, N. Varjasi: *Load Balancing using Potential Functions for Hierarchical Topologies*: in Acta Technica Jaurinensis Vol. 4. No. 4., 2011. pp. 413–424.

[Varjasi 06a] N. Varjasi: *Parallel algorithm for linear equations in different architectural systems*, 77th GAMM annual meeting 2006, Berlin, 2006. pp.440.

[Varjasi 06b] Varjasi, Norbert: *Lineáris egyenletrendszerek megoldására alkalmas párhuzamos algoritmus.* In: Kegyes-Brassai Orsolya Katalin (szerk.) Tavaszi Szél, 2006, Kaposvár: konferenciakiadvány. Kaposvár. Doktoranduszok Országos Szövetsége, Budapest, 2006. pp. 311 – 314. ISBN:963 229 773 3

[Varjasi 07] N. Varjasi: *Parallel Algorithm for linear equations with different network topologies*, Proceedings of International e-Conference on Computer Science (IeCCS) 2006. In: Lecture Series on Computer and Computational Sciences, Brill Academic Publishers, 2007. pp. 502 – 505. ISBN 978-90-04-15592-3

[Varjasi 08a] Varjasi Norbert: *Akusztikai minőség-ellenőrzési mérések sokcsatornás feldolgozása ipari gyártósoron.* Informatika Korszerű Technikái konf. kiadványa. Dunaújváros,

Magyarország, 2008.03.07–2008.03.08. pp. 144-151. ISBN 978-963-87 780-2-4

[Varjasi 08b] Varjasi Norbert: *Real-time and parallel quality control processing on industrial production lines.* POLLACK PERIODICA vol. 3. no. 3. 2008. pp. 105 – 111. ISSN 1788-1994.

KIVONAT

A lineáris egyenletrendszerek iteratív megoldási módszerei a numerikus matematika alapfeladatai közé tartoznak. Az általános iteratív módszerek alkalmasak a nagy, de ritka mátrixú feladatok megoldására, viszont az általános $n \times m$ dimenziós feladatokra csak korlátozásokkal alkalmazhatók. A dolgozatban bemutatott párhuzamosított algoritmusok újdonsága a reziduum minimalizáció és az altér-dekompozíció. Ezen párhuzamos algoritmusok alkalmasak az általános és rosszul kondicionált feladatok megoldására különböző párhuzamos architektúrán, különböző hálózati struktúrákat használva. Megmutattuk, hogy a párhuzamosítás során előálló új algoritmusok evolúciós jellegűek. Számítógépes tesztek segítségével elemeztük az algoritmusok működését és feltártuk a párhuzamos gyorsítás lehetőségeit és az előálló szuper-gyorsítási eseteket is.

A dolgozatban definiált algoritmusokat klaszteren és multiprocesszoros gépen *(HP Blade Systems c3000)* 88 processzoron teszteltük. A részletes tesztelési adatgyűjtés megmutatta, hogy a bemutatott algoritmusok alapfeltevése helyes és alkalmasak a nagyméretű rosszul kondicionált lineáris egyenletrendszerek megoldására.

A bemutatott algoritmusok alkalmasak lehetnek olyan homogén, vagy heterogén sokprocesszoros architektúrára tervezett szoftverrendszerek részeként működni, amelyekben gyakoriak a nagyméretű rossz kondicionáltságú egyenletrendszerek. Ilyen problémakörök megjelenhetnek az optimalizáció, vezérlő- és szimulációs rendszerek, adatbányászat, autóipar, stb. területén.

Új, ún. potenciálfüggvénnyel leírt modellt alkottunk homogén párhuzamos rendszereken használt hierarchikus topológiák jellemzésére. Napjainkban egyre gyakoribbak a különböző architektúrákkal szerelt sokprocesszoros gépek, és ezeken a párhuzamos hierarchikus topológiával dolgozó szoftverek különböző viselkedést produkálhatnak. A bemutatott speciális modell alkalmas a hierarchikus topológiák kiegyensúlyozásának *„amortizációs potenciál módszer”* segítségével egy topológia *„jóságát”* meghatározni. A modell hasznosságát számítógépes tesztek eredményeivel egészítettük ki és támasztottuk alá.

Kulcsszavak: *párhuzamos programozás, lineáris egyenletrendszerek, iteratív módszerek, klaszter, multiszámítógép, elosztott rendszerek, amortizációs potenciál, kiegyensúlyozás.*

ABSTRACT

For solving linear systems of equations the iteration algorithms are often recommended for the large linear systems with sparse matrix. But in the case of general $n \times m$ matrices the classical iterative algorithms are not applicable with a few exceptions. The theoretical basis of our algorithms is a new approach of the minimal residual algorithms and the subspace decomposition technique that permits different, highly parallel realizations on different processors connected with different topology. The algorithms presented here based on the minimization of the residual of the solution and it has some genetic character. More effective parallel algorithms could be to give for general large ill-conditioned linear system of equations. Here we describe some sequential and parallel versions of these algorithms.

Moreover we show some numerical test results of the algorithms including the speed-up of parallel execution. We have realized the proposed algorithm on a multiprocessor computer type *HP Blade systems c3000* with 88 processors. We made some comparison and we have found, that our approach could be efficient in parallel environment. Computer tests have confirmed our theoretical results: the parallel implementation realized shows much better efficiency than any sequential one.

The goal of creating these algorithms was therefore useful method for several practical and realistic problems such as optimization, finite element methods, control or simulation problems etc. even if they lead to general large, dense ill-conditioned linear system of equations. Proposed parallel algorithms easily can be adjusted to the large scale of different parallel architectures, such as distributed and heterogeneous clusters or supercomputers.

We consider a new approach to the load balancing for parallel systems. Today's parallel computers use multiprocessors and multi-core architectures. The presented new model for balancing tree topologies is based on the *"amortization potential method"* for homogeneous systems. Based on our proposed model a notion can be introduced: the *"goodness of the hierarchical topology"*, which can be characterized by potential functions. We give some examples for these potential functions, and we propose the usefulness of the model with computer experiments.

Keywords: *parallel programming, linear equation, cluster computing, iterative method, LU decomposition, cluster, distributed system, load balance, amortization potential.*