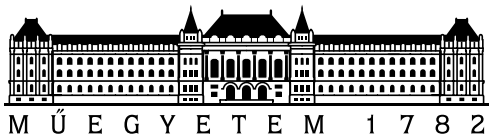


Budapesti Műszaki és Gazdaságtudományi Egyetem
Irányítástechnika és Informatika Tanszék

PARTICIONÁLÓ ALGORITMUSOK A HARDVER/SZOFTVER EGYÜTTES TERVEZÉSBEN

Ph.D. értekezés tézisei

Mann Zoltán Ádám

Témavezető: Dr. Arató Péter
a Magyar Tudományos Akadémia levelező tagja

Budapest

2004

1. Bevezetés

Napjainkban a számítógépes rendszerekkel szemben támasztott igények egyre nagyobbak. A tervezendő rendszerek egyre bonyolultabbak, ugyanakkor a kiélezett piaci verseny miatt lényeges cél a tervezési ciklus lerövidítése is. A legtöbb alkalmazásban a funkcionalitás mellett szigorú korlátok vannak a rendszer árára, sebességére és megbízhatóságára is. A mobil eszközök széleskörű elterjedésével kiemelten fontos lett az energiafelvétel és hőleadás minimalizálása is [15].

Ez különösen így van beágyazott rendszerek esetén, melyek mindennapjaink részeivé váltak a szórakoztató elektronikai termékekben, mobiltelefonokban, chipkártyákban, autóelektronikában stb. E rendszerek tipikus további sajátossága, hogy működésük különböző hardver és szoftver egységek szoros együttműködésével jön létre. Itt hardver alatt alkalmazásspecifikus – tehát konkrétan az adott rendszer számára tervezett – hardver értendő, míg szoftver alatt egy program, mely valamilyen általános célú hardveren (például mikroprocesszoron) fut.

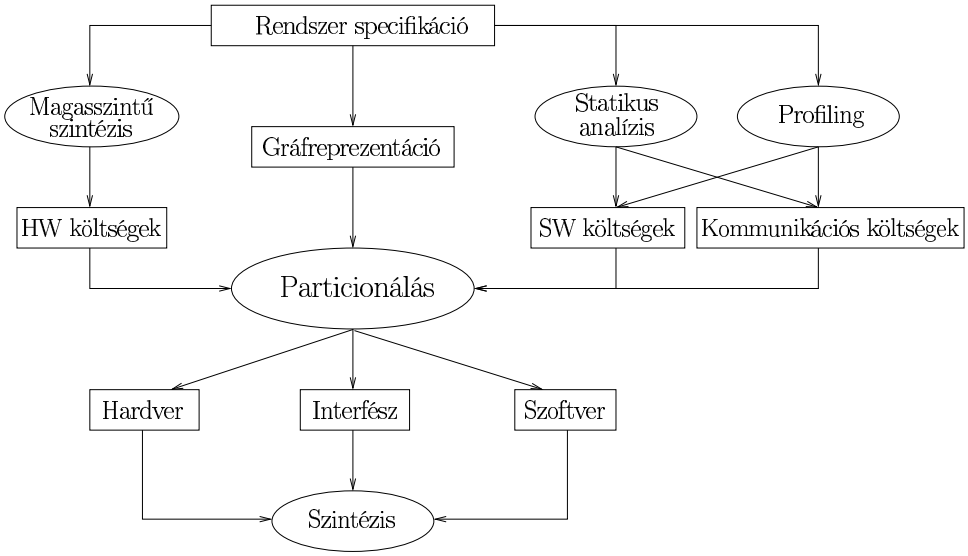
A tervezendő rendszer funkcionalitásának általában jelentős része megvalósítható akár hardverben, akár szoftverben. Mindkét lehetőségnek megvannak a maga előnyei és hátrányai. A szoftver tipikusan sokkal olcsóbb, de lényegesen lassabb, ráadásul az általános célú processzorok több energiát fogyasztanak, mint egy alkalmazásspecifikus hardvermegoldás. Másrészről a hardver gyorsabb és kedvezőbb energiafelhasználású, ám lényegesen drágább.

Ebből következően célszerű a rendszer teljesítmény vagy energiafogyasztás szempontjából kritikus komponenseit hardverben megvalósítani, a kevésbé kritikusakat pedig szoftverben. Így optimális kompromisszumot találhatunk a sebesség, energiafogyasztás és költségek vonatkozásában [C6]. Azonban az ilyen optimum megtalálása korántsem egyszerű, főleg a komponensek nagy száma és eltérő tulajdonságai miatt. Továbbá ha rendszerünk hardver és szoftver komponenseket egyaránt tartalmaz, ez számos új elemmel növeli a tervezési feladat komplexitását, hiszen gondoskodni kell a hardver és szoftver közötti hatékony kommunikációról, szinkronizációról stb.

Ilyen összetett, hardver és szoftver komponenseket egyaránt tartalmazó rendszerek tervezésének támogatásával foglalkozik a *hardver/szoftver együttes tervezés* (angolul hardware/software co-design, vagy röviden HSCD), mely az 1990-es évek eleje óta aktívan kutatott terület [8, 10, 11, 16, 19, 23, 20, 24]. Egy ilyen rendszer blokkvázlata látható az 1. ábrán.

A HSCD központi lépése a *particionálás*, melynek feladata eldönteni, hogy a rendszer mely komponensei kerüljenek hardverbe és melyek szoftverbe. Nyilvánvalóan ez az a lépés, melynek során a fent említett optimális kompromisszumot kell megtalálni az egymásnak ellentmondó szempontok alapján, természetesen figyelembe véve a hardver és szoftver közötti kommunikáció költségeit is [C5].

A kommunikáció megragadására a particionálandó rendszert általában egy gráffal szokták reprezentálni, melynek csúcsai a rendszer komponensei, élei pedig a komponensek közötti kommunikációt tükrözik. A gráf csúcsai és élei általában különféle költségértékekkel vannak ellátva, melyek jelenthetnek például futási időt, elfoglalt chipterületet vagy akár hőtermelést. Amint az az 1. ábrán jól látható, a HSCD lényeges feladata ennek a gráfnak



1. ábra. Egy hardver/szoftver együttes tervezési rendszer blokkvázlata

az előállítása, valamint a költségértékek meghatározása is, azonban a particionálás során ezeket már adottnak feltételezzük.

2. Kutatási célkitűzések

Kutatásom célja a hardver/szoftver együttes tervezés során fellépő particionálási probléma algoritmikus szempontból történő vizsgálata volt. Ez a következő két fő feladatra bontható:

- A probléma algoritmikus komplexitásának vizsgálata. A szakirodalomban számos helyen találkozhatunk azzal a kijelentéssel, hogy a particionálási probléma $\mathcal{NP}$-teljes [3, 7, 16, 22]. Bizonyításból azonban eddig csak egyet közöltek [16], mégpedig a probléma egy meglehetősen bonyolult megfogalmazására, mely magában foglalja az ütemezés problémáját is, és valójában az $\mathcal{NP}$-teljességi bizonyítás is ezt használja ki. Másrészt a szakirodalomban a particionálási problémának számos más, egyszerűbb megfogalmazása is található, és ezek komplexitása nem bizonyított. Éppen ezért fontos célom volt, hogy a particionálási probléma különböző megfogalmazásainak komplexitását vizsgáljam, esetleg polinomidőben megoldható eseteket találjak, illetve az $\mathcal{NP}$-nehéz változatok komplexitását formálisan is igazoljam.

- Kutatásaim másik fő célja a probléma $\mathcal{NP}$-nehéz változataihoz kapcsolódott, mert – mint vizsgálataimból kiderült – a probléma számos, ütemezést nem tartalmazó változata is valóban $\mathcal{NP}$-nehéz. Mivel e problémák megoldása a mérnöki gyakorlatban nélkülözhetetlen, így célul tűztem ki hatékony heurisztikák kidolgozását. Ennek során – szemben a legtöbb korábbi, ilyen irányú kutatással – az elsődleges cél nem az

volt, hogy a particionálási problémakör minél több részletét kezelő algoritmusokat dolgozzak ki, hanem inkább egy limitált problémára próbáltam minél hatékonyabb algoritmusokat adni.

A heurisztikák kifejlesztéséhez természetesen szorosan kapcsolódó feladat a heurisztikák gyakorlati példákon való tesztelése, hangolása, összevetése is. Különösen fontos az egyes heurisztikákra jellemző eredményesség/futási idő viszony meghatározása, mivel ennek alapján dönthető el, hogy egy adott szituációban melyik algoritmust érdemes használni.

Az általam használt probléma definíciók lényege a következő. A gráf minden csúcsához adott két nemnegatív szám, melyek neve szoftver költség illetve hardver költség. Ezen kívül minden élre adott egy úgynevezett kommunikációs költség. Egy adott partíció hardver költségén a hardverbe kerülő csúcsok hardver költségeinek összegét értjük. Hasonlóan értelmezhető egy partíció szoftver költsége. Egy partíció kommunikációs költsége pedig az elvágó élek kommunikációs költségeinek összege. A gyakorlatban sokszor mind a szoftver költség, mind pedig a kommunikációs költség idő dimenziójú, és a kettő összege adja lényegében a rendszer teljes futási idejét. Ezért érdemes e két költséget külön-külön is és összesítve is vizsgálni.

P1: Eldöntendő, hogy létezik-e olyan partíció, melyre mind a szoftver költség és a kommunikációs költség összege, mind pedig a hardver költség adott korlát alatt van.
P2: Keressünk olyan partíciót, melynek hardver költsége adott korlát alatt van, a szoftver költség és a kommunikációs költség összege pedig minimális.
P3: Keressünk olyan partíciót, melyre a szoftver költség és a kommunikációs költség összege adott korlát alatt van, a hardver költsége pedig minimális.
P4: Keressünk olyan partíciót, melynek mind a hardver költsége, mind a szoftver költsége adott korlát alatt van, a kommunikációs költsége pedig minimális.
P5: Keressünk olyan partíciót, melyre a hardver költség, szoftver költség és kommunikációs költség adott nemnegatív α, β, γ súlyokkal vett súlyozott összege minimális.

2. ábra. Probléma definíciók

Mindezek alapján a vizsgált változatok a 2. ábrán láthatóak.

3. Kutatási módszertan

A disszertációmban használt megfogalmazásban a hardver/szoftver particionálás problémája nem más, mint egy irányítatlan gráf valamilyen szempontból optimális kettévágása.

Ebből fakadóan a kutatásaim során felhasznált módszerek többnyire a gráfelmélet, algoritmuselmélet és a heurisztikus optimalizálás köréből kerülnek ki.

Az $\mathcal{NP}$-nehézségi eredmények bizonyításánál ismert $\mathcal{NP}$-teljes problémákat – nevezetesen a HÁTIZSÁK és a MINIMÁLIS GRÁFFELEZÉS problémát [4] – vezettem vissza a particionálási probléma különböző változataira. A polinomialitási eredmények bizonyításához azt használtam ki, hogy egy gráfban mind a minimális vágás, mind a minimális $s-t$ vágás polinomidőben megtalálható [1].

Heurisztikus algoritmusaimban vizsgáltam általános célú heurisztikákat (determinisztikus illetve randomizált mohó algoritmusok, genetikus algoritmusok [6]), valamint általában gráfparticionálásra kifejlesztett módszereket (Kernighan–Lin algoritmus [17], Fiduccia–Mattheyses algoritmus [9]). Ezen kívül itt is alkalmaztam minimális vágás feladatra történő visszavezetést. Amelyik algoritmusnál az implementáció és a megfelelő adatstruktúra kiválasztása is kérdéses volt, ott ezzel is foglalkoztam, így például a geometriai algoritmusok világából kölcsönöztem adatstruktúrát az egyik heurisztikus algoritmushoz.

Az algoritmusokat meg is valósítottam, és számos gyakorlati feladaton (ipari benchmark feladatok és nagyméretű véletlen gráfok) teszteltem. Amelyik algoritmusnak hangolandó paraméterei voltak, ott ezt a hangolást is a benchmark problémákon végzett futtatások segítségével végeztem. Továbbá az algoritmusok futási idejét és eredményességét is ilyen módon tudtam összehasonlítani.

4. Új eredmények

1. Tézis: Formálisan igazoltam a hardver/szoftver particionálási probléma különböző lehetséges definícióira vonatkozó alábbi bonyolultsági eredményeket [C5, J3].

1. Tétel. *A **P1** probléma $\mathcal{NP}$-teljes, még abban a speciális esetben is, ha csak élek nélküli gráfokat tekintünk.*

A bizonyításból kiderül, hogy valójában a **P1** probléma élek nélküli gráfokra ekvivalens a HÁTIZSÁK feladattal. Ugyanebből adódik a következő is:

2. Tétel. ***P2** és **P3** $\mathcal{NP}$-nehéz, még abban a speciális esetben is, ha csak élek nélküli gráfokat tekintünk.*

A HÁTIZSÁK probléma, azzal együtt, hogy $\mathcal{NP}$-teljes, viszonylag hatékonyan kezelhető, nevezetesen léteznek rá úgynevezett pszeudo-polinomiális algoritmusok [4]. A következő két tétel azt mutatja, hogy a hardver/szoftver particionálás problémája általános gráfokra ennél nehezebb:

3. Tétel. *A **P1** probléma erős értelemben is $\mathcal{NP}$-teljes.*

4. Tétel. ***P2*** *és* ***P3*** *erős értelemben is $\mathcal{NP}$-nehezek.*

A HÁTIZSÁK problémával való kapcsolatról mondottak folytán, amennyiben a kommunikációs költségek elhanyagolhatóak, úgy a probléma viszonylag hatékonyan kezelhetővé válik. A következő tétel szerint ez a másik végletre még inkább így van:

5. Tétel. *Ha a hardver és szoftver költségek elhanyagolhatóak, vagyis egyedül a kommunikációs költséget kell figyelembe venni, akkor mindegyik definiált probléma polinomidőben megoldható. Sőt, ez arra az általánosabb megfogalmazásra is igaz, melyben néhány csúcsra előírható, hogy mindenképp szoftverben kell lenniük, míg másokra, hogy mindenképp hardverben.*

Az előbbiekben hatékonyan kezelhető speciális esetekről volt szó. A következő tétel szerint azonban a **P5** probléma általánosságban is polinomidőben megoldható:

6. Tétel. *A* ***P5*** *probléma megoldható optimálisan polinomidőben.*

Végül egy approximálhatatlansági eredmény:

7. Tétel. *A* ***P4*** *probléma $\mathcal{NP}$-nehéz. Sőt, ha $\mathcal{P} \neq \mathcal{NP}$, akkor nem létezhet rá approximációs algoritmus sem.*

2. Tézis: Kifejlesztettem egy új genetikus algoritmust a hardver/szoftver particionálás problémájára [C7].

Disszertációmban az algoritmust a **P3** problémára mutatom be, de könnyen adaptálható a probléma többi változatához is.

Ugyan már korábban is léteztek genetikus particionálók, de az általam kifejlesztett algoritmus hatékonyabb ezeknél. Ennek fő oka az úgynevezett érvénytelen egyedek (tehát a korlátot sértő partíciók) konzisztens kezelése. Ennek leglényegesebb elemei a következők:

- Ugyan az érvénytelen egyedek nem tekinthetők a probléma korrekt megoldásának, és éppen ezért logikus lenne nem megengedni ilyeneket, kimutattam, hogy az algoritmus eredményesebb, ha mégis lehetővé tesszük, hogy érvénytelen egyedek is legyenek a populációban. Ennek oka, hogy gyakran a majdnem érvényes egyedek hordozzák a legértékesebb genetikai mintákat.

- Természetesen előnyben kell részesíteni az érvényes egyedeket, ezért a fitness függvényt úgy kell megválasztani, hogy büntesse az érvénytelenséget. Több különböző függvényt is kipróbáltam annak érdekében, hogy a legmegfelelőbbet választhassam. Az empirikus tesztek tanulsága szerint ugyanis ennek nagy jelentősége van. A következő alakú függvény bizonyult a legjobbnak:

$$f(P) = \begin{cases} \Theta' - H_P & \text{ha } P \text{ érvényes} \\ \Theta' - H_P - c * exc(P) - M & \text{ha } P \text{ nem érvényes.} \end{cases}$$

Itt $f(P)$ a P partíció fitness értéke, H_P a hardver költsége, $exc(P)$ pedig az érvénytelenségének mértéke, vagyis azt adja meg, hogy P hány százalékkal lépi át a futási időre vonatkozó korlátot, végül Θ', c és M megfelelő konstansok. Látható, hogy a particionálás egymással ellentétes céljai úgy vannak a fitness függvényben egyesítve, hogy a fitness maximalizálása a hardver költség csökkentésének és az időre vonatkozó korlát betartásának az irányába hat. Ugyanakkor a korlát kismértékű túllépése megfelelően alacsony hardver költséggel kompenzálható.

- Annak érdekében, hogy az algoritmus végül egy érvényes partíciót adjon, gondoskodni kell arról, hogy mindvégig legyenek a populációban érvényes egyedek. E célból kipróbáltam több különböző módszert, amivel a kezdeti populációba érvényes egyedek generálhatók anélkül, hogy a populáció degenerálódása fenyegetne. További fontos paraméter, hogy a kezdeti populáció mekkora részét érdemes érvényes egyedekkel kitölteni, és mekkorát nem feltétlenül érvényes, véletlenül generált egyedekkel.

- Kimutattam, hogy a szelekció (elitizmus) operációjának hatékonysága nagyban növelhető azzal, hogy nem csupán a populáció legjobb egyedeit örökítjük tovább, de a legjobb érvénytelen egyedeket is.

Ezen kívül több különböző lehetőséget kipróbáltam a rekombináció operációjára is, valamint különböző rekombinációs és mutációs rátákat. A paraméterek hangolása nagyszámú empirikus tesztet igényelt, azonban a legmegfelelőbb paraméter kombináció kiválasztásával az algoritmus hatékonysága jelentősen növekedett.

3. Tézis: Kifejlesztettem egy új, Kernighan–Lin típusú heurisztikus algoritmust a hardver/szoftver particionálás problémájára [J3, O6].

Az eredeti Kernighan–Lin algoritmus a MINIMÁLIS GRÁFFELEZÉS problémájára készült [17], mely sok szempontból egyszerűbb a hardver/szoftver particionálásnál. Ezért az algoritmus adaptációja korántsem triviális. Ráadásul megjelenése óta számos javítás született már a Kernighan–Lin algoritmushoz [9, 13, 18, 21, 14, 5, 25], így azt is meg kellett vizsgálni, hogy ezek mennyiben vihetők át a hardver/szoftver particionálás problémájára. Konkrétan a következő kérdéseket kellett megoldanom az adaptáció során:

- A hardver/szoftver particionálás különféle költségfüggvényeinek beépítése az algoritmus által használt nyereség függvénybe. Ennek során ismét előkerült az érvénytelen partíciók dilemmája, és ismét az derült ki, hogy kismértékű érvénytelenséget érdemes megengedni, a megfelelő büntető mechanizmus mellett.

- A kezdeti partíció kialakítása. Szintén cél volt, hogy érvényes legyen a kiindulási partíció, emellett az algoritmus eredményessége nagyban múlik a kezdeti partíció minőségén is. Ezért disszertációmban részletesen foglalkozom a kezdeti partíció keresésének különböző lehetőségeivel.

- Azonos nyereségű lehetőségek között való döntés problémája. Kimutattam, hogy érdemes a közel azonos nyereségű választási lehetőségeket is úgy kezelni, mintha egyenlőek lennének, és LIFO típusú stratégiát [13] alkalmazni a közülük való választáshoz.

- Az elmozdított csúcsok lezárásának (locking) módja. Itt is az derült ki, hogy az eredeti, Kernighan–Lin-féle stratégiánál jobb is alkalmazható [14].

Az algoritmus kifejlesztése során a legnagyobb kihívást a hatékony implementáció érdekében a megfelelő adatstruktúra megválasztása jelentette. Bebizonyítottam, hogy tetszőleges adatstruktúra mellett az algoritmus időigénye legalább $\Omega(n \log n)$ lépés fázisonként. Az általam adott implementáció – mely a range tree adatstruktúrára [2] épül – ritka gráfok esetén eléri ezt az alsó korlátot.

Végezetül megvizsgáltam azt is, hogy a használt hardver/szoftver particionálási modell hogyan befolyásolja a Kernighan–Lin heurisztika alkalmazhatóságát. Megmutattam, hogy az algoritmus lényeges lassulás nélkül tud több költségfüggvényt is kezelni. Megmutattam azt is, hogy az ütemezés problémája hogyan integrálható az algoritmusba. Azonban ez jelentős teljesítményromlást okoz.

4. Tézis: Kifejlesztettem egy új, minimális vágás keresésén alapuló heurisztikus algoritmust, melynek lényege, hogy az eredeti gráfból különböző súlyozásokkal előálló segédgráfok minimális vágásai közül a korlátnak megfelelő legjobb vágást választja ki [J3, O6].

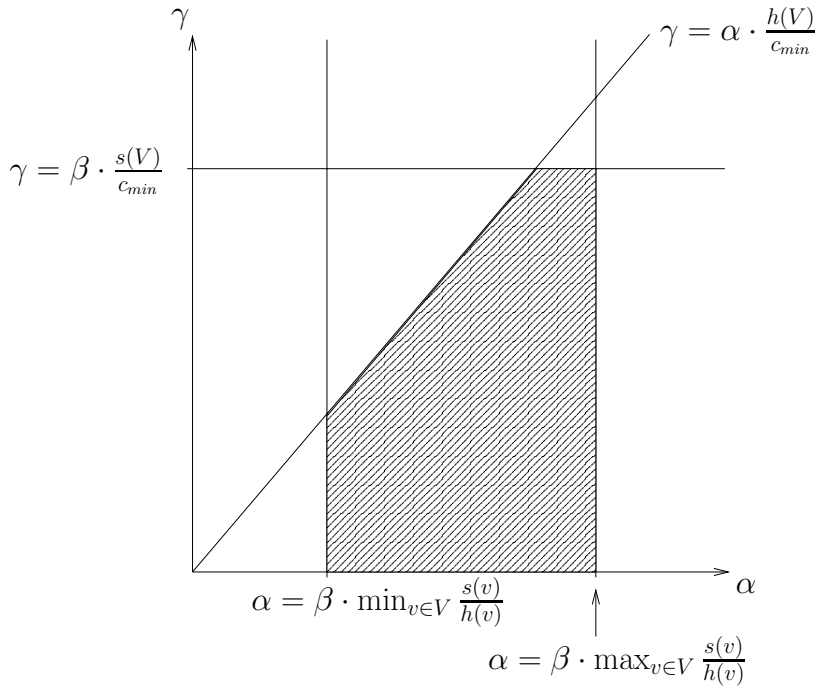
Az algoritmus leglényegesebb újítása, hogy a korábbi megközelítésekkel ellentétben nem valamely ismert, általános célú heurisztika alkalmazása, hanem kifejezetten a hardver/szoftver particionálás problémájára lett kifejlesztve, annak kombinatorikus tulajdonságaira épül.

A módszer magja a 6. Tételben hivatkozott algoritmus, mely polinomidőben optimális eredményt szolgáltat a **P5** problémára. Ebből úgy nyerhetünk egy heurisztikát a **P3** problémára, hogy különböző α, β, γ súlyokhoz tartozó **P5** probléma példányokat konstruálunk, és ezeket optimálisan megoldjuk az erre alkalmas polinomidejű algoritmussal. Ezután a kapott partíciók közül kiválasztjuk azt, amely a **P3** probléma szempontjából a legjobb, tehát egy olyat, amely kielégíti a korlátot, és az ilyenek közül minimális hardver költségű.

Ilyen módon elérhető, hogy az algoritmus csak olyan partíciókat vizsgáljon, melyek valamilyen szempontból jó minőségűek, nevezetesen egy rokon probléma optimális megoldásai megfelelő paraméterek mellett. Ennek köszönhetően az algoritmus jó eredményeket ad annak ellenére, hogy a keresési térnek csak egy kis hányadát nézi át.

Szintén ennek a módszernek köszönhető, hogy az algoritmus képes arra is, hogy működése során egyre jobb *alsó becsléseket* találjon az optimumra. Így az algoritmusnak mindig

van egy felső becslése arra nézve, hogy az általa eddig talált legjobb eredmény milyen messze lehet az optimumtól. Ez egy olyan tulajdonság, mellyel egyetlen korábban javasolt particionáló algoritmus sem rendelkezik.

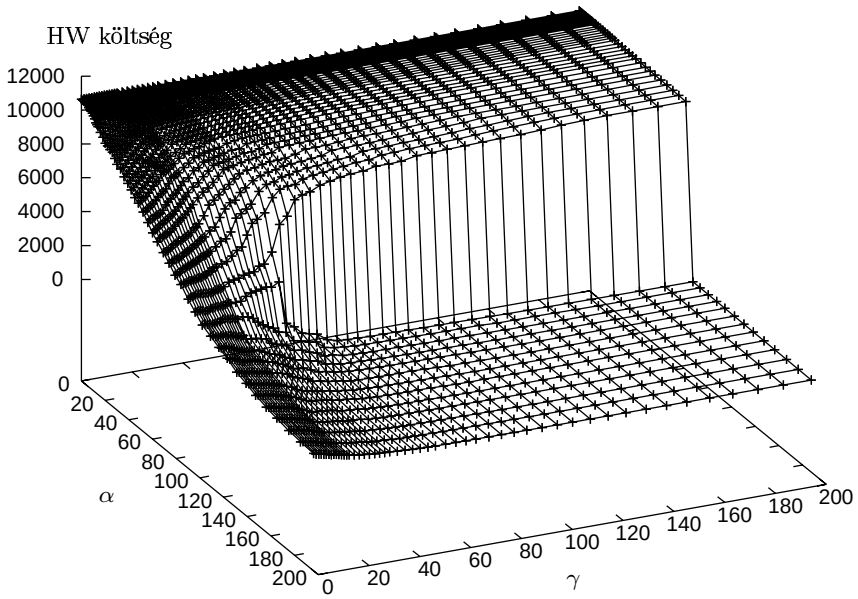


3. ábra. A vizsgálandó tartomány

Az algoritmus sebessége nagyban függ attól, hogy hány különböző (α, β, γ) hármast próbál ki, és ezek kiválasztása hogyan történik. Nyilván csak a három paraméter aránya számít, ezért az egyik, például β, rögzíthető, és így csak a másik két paramétert kell változtatni. Tehát az $\alpha - \gamma$ sík bizonyos pontjai kerülnek kipróbálásra. Disszertációmban formálisan igazoltam, hogy e síknak csak egy korlátos tartományával kell foglalkozni, mivel az ezen kívüli pontok nem adnak új eredményt. A tartomány a 3. ábrán látható.

A tartományon belül valamilyen $d\alpha$ illetve $d\gamma$ lépésközzel halad az algoritmus. A hatékonyság érdekében $d\alpha$ és $d\gamma$ adaptív módon változik az algoritmus futása során: amíg nem találunk jobb megoldást, a lépésköz $(1+\varepsilon)$-nal szorzódik, vagyis a keresés exponenciálisan gyorsul, azonban ha az eddigieknél jobb megoldásra akadunk, a lépésköz ismét lecsökken, és a keresés alaposabbá válik.

Ilyen módon közelítőleg megtalálható a legkedvezőbb pont a síkon, ám lehet, hogy az algoritmus a ténylegesen legjobb pontot átugorja. Ezért egy második fázisban sor kerül az eddig talált legjobb pont környezetének alaposabb átfésülésére. Ez a módszer persze akkor működik igazán jól, ha az optimális partíció költségei α és γ függvényében simán és valamilyen egyszerű struktúra szerint változnak. Hogy ez valóban így van, azt a 4. ábra példája szemlélteti. Az ábrán az is látható, hogy az optimális partíció hardver költsége monoton csökken α-ban. Ezt formálisan is igazoltam.



4. ábra. A **P5** probléma optimális megoldásának hardver költsége α és γ függvényében ($\beta = 100$)

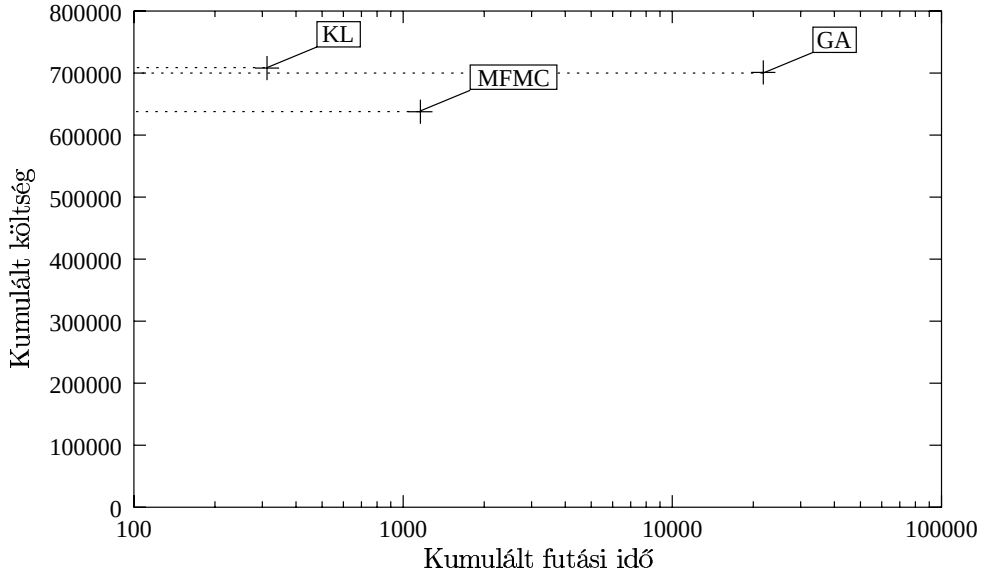
5. A 2., 3. és 4. Tézis eredményeinek összehasonlító értékelése

A három javasolt algoritmust összehasonlítottam mind elméleti tulajdonságaik alapján analitikusan, mind pedig ipari benchmark feladatok segítségével empirikusan. Az analitikus összehasonlítás során az algoritmusok következő tulajdonságait vizsgáltam:

- Kezdeti partíció(k) szerepe, fontossága, generálási módja
- Leállási feltételek
- Párhuzamosítási lehetőségek
- Stabilitás, vagyis az a képesség, hogy az algoritmus „meg tudja jegyezni" a keresési tér ígéretes részeit
- Kiterjesztési lehetőségek
- Hangolandó paraméterek, hangolás nehézsége
- Időegység alatt megvizsgált partíciók száma, következő vizsgálandó partíció meghatározásának bonyolultsága
- A költségfüggvények egymáshoz viszonyított fontosságának megjelenése az algoritmus során

- Determinizmus illetve randomizálás

Az empirikus teszteket számos benchmark feladaton végeztem: a MiBench benchmark-család [12] elemein, a kutatócsoportunk által tervezett összetettebb rendszereken, végül nagyméretű véletlen gráfokon.

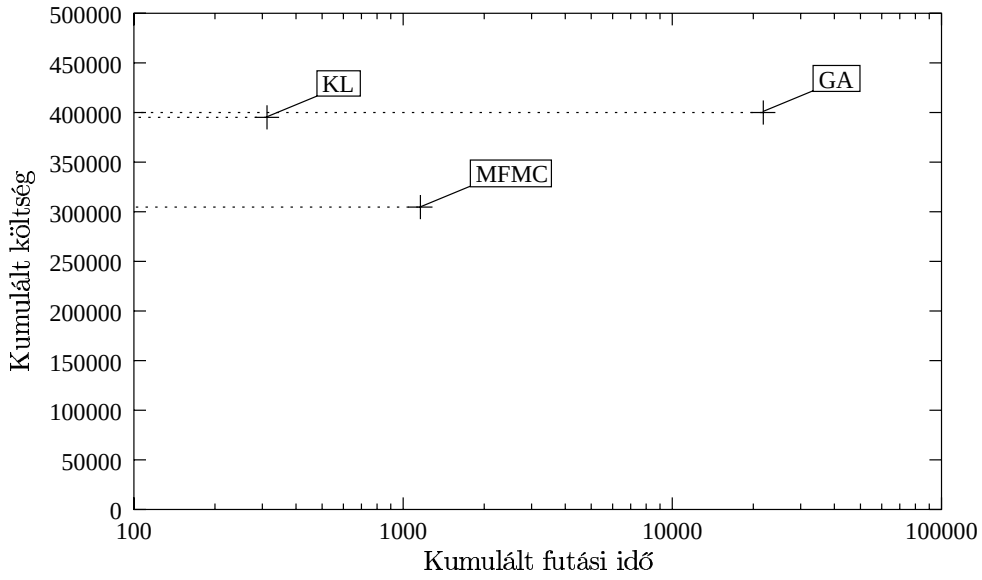


5. ábra. Az algoritmusok futási idejének valamint a talált megoldás költségének a viszonya alacsony CCR és szoros korlát esetén

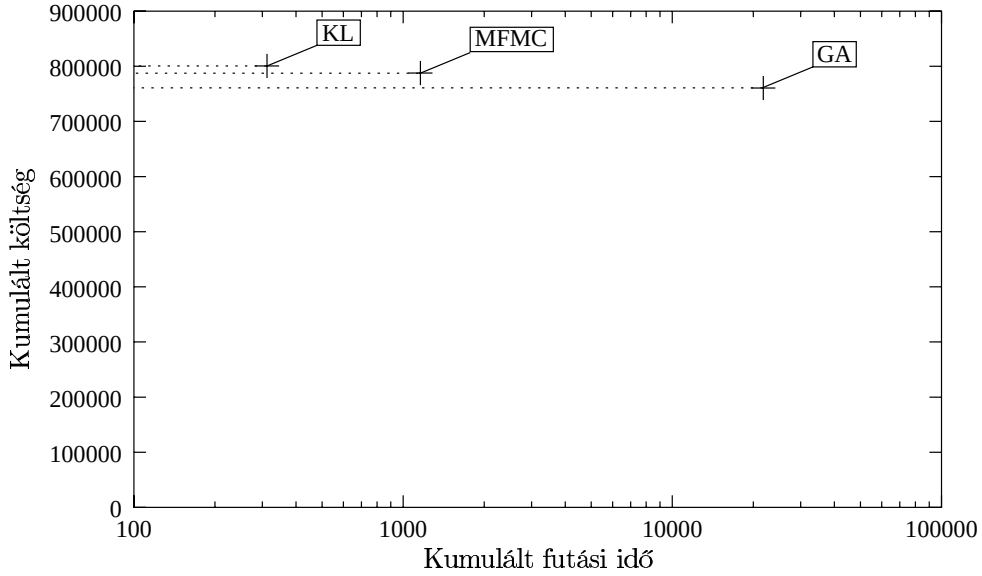
A tesztekből kiderült, hogy az algoritmusok teljesítménye a probléma méretén kívül erősen függ a korlát szigorúságától, valamint a kommunikációs költségek és szoftver költségek arányától (communication to computation ratio, CCR). A három algoritmus jellemző sebesség/minőség viszonyai láthatók különböző korlát és különböző CCR érték mellett az 5-8. ábrákon. A kísérletekben a „szoros korlát" az értelmes korlát értékek intervallumának alsó feléből, „laza korlát" pedig a felső feléből véletlenszerűen választott korlátot jelentette. „Alacsony CCR" esetén a kommunikációs költségek átlaga nagyjából megegyezett a maximális szoftver költséggel, míg „magas CCR" esetén a maximális szoftver költség 10-szeresével.

Mint az ábrákból jól látható, alacsony CCR esetén (a korlát szigorúságától függetlenül) a 4. Tézisben bemutatott MFMC-alapú algoritmus adja a legjobb eredményeket. Magas CCR és laza korlát esetén nincs egyértelmű győztes az algoritmusok között, magas CCR és szoros korlát esetén pedig a GA eredménye a legjobb. Futási idő tekintetében minden esetben a KL-típusú algoritmus a legkedvezőbb, második az MFMC-alapú, és leglassabb a GA.

A tesztekből az is kiderült, hogy legtöbb esetben az MFMC-alapú algoritmus által szolgáltatott alsó korlátok nincsenek messze az algoritmus által adott eredmény költségétől, ami mind az eredmény, mind az alsó korlát jó minőségét bizonyítja.



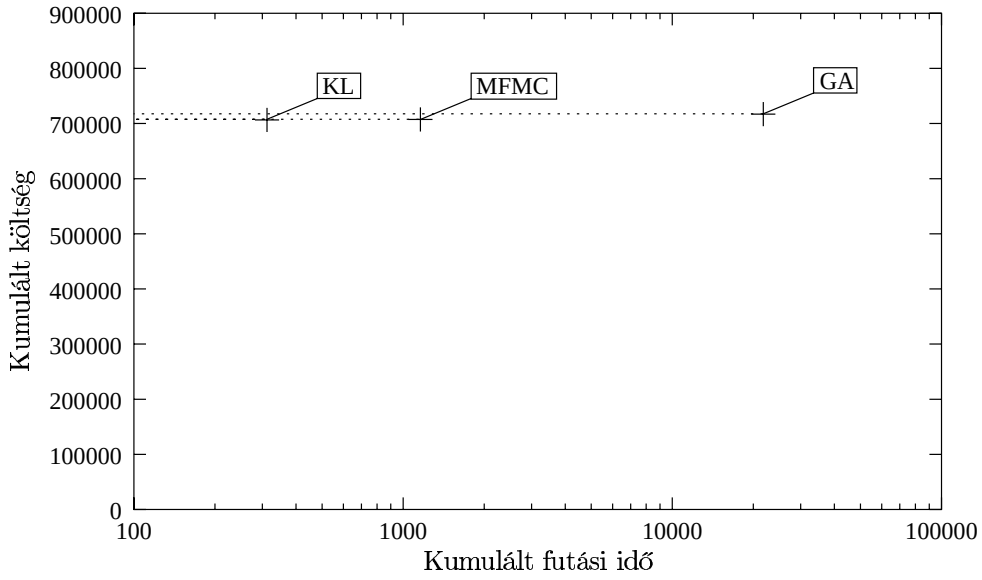
6. ábra. Az algoritmusok futási idejének valamint a talált megoldás költségének a viszonya alacsony CCR és laza korlát esetén



7. ábra. Az algoritmusok futási idejének valamint a talált megoldás költségének a viszonya magas CCR és szoros korlát esetén

6. Az eredmények alkalmazása

Az 1. Tézis eredményei közvetetten kerültek alkalmazásra: a **P5** problémára bemutatott polinomidejű algoritmus lett az alapja a 4. Tézis algoritmusának; az $\mathcal{NP}$-nehézségi ered-



8. ábra. Az algoritmusok futási idejének valamint a talált megoldás költségének a viszonya magas CCR és laza korlát esetén

mények pedig segítették a problémák jobb megértését, és így vezettek a 2-4. Tézisek heurisztikáihoz.

A 2-4. Tézisek heurisztikus algoritmusai már több esetben gyakorlati alkalmazásra kerültek:

- Egy, az elágazás és korlátozás (branch-and-bound, BB) elvén működő, kutatócsoportunk által kifejlesztett *egzakt* particionáló algoritmusban [J6]. A szóban forgó algoritmus többszáz pontú gráfok optimális particionálását el tudja végezni belátható időn belül. Ráadásul lényegesen gyorsabb a hasonló célú, egészértékű programozáson alapuló egzakt algoritmusnál. A BB algoritmusban mind a 2. Tézisben bemutatott GA, mind pedig a 4. Tézis MFMC-alapú algoritmusa alkalmazásra került. Szerepük az előzetes optimalizálás, azaz kezdeti partíció és ezzel *kezdeti felső korlát* meghatározása a BB számára. A tapasztalatok szerint mindkét alkalmazott heurisztika jelentősen javít a BB algoritmus futási idején, mégpedig anélkül, hogy az algoritmus egzakt volta sérülne.

  Továbbá az 1. Tézisben szereplő polinomiális algoritmus is fontos szerephez jut a BB algoritmusban. Ugyanis használható annak megállapítására, hogy egy adott részleges megoldás (tehát amikor még nem minden csúcsról dőlt el, hogy hardverbe vagy szoftverbe kerüljön) kiterjeszthető-e érvényes teljes megoldássá úgy, hogy költsége kevesebb legyen, mint az eddig talált legjobb érvényes partíció költsége. Így az 1. Tézis polinomidejű algoritmusa jelentősen hozzájárul ahhoz, hogy a BB le tudja vágni a keresési fa jobb megoldással nem kecsegtető ágait, így csökkentve a futási időt.

- Egy, a japán Akita Industrial Development Center által finanszírozott projektben kutatócsoportunk részt vett különféle orvosi informatikai alkalmazások elkészítésében. A mi feladatunk az volt, hogy néhány neurális algoritmus (Kohonen önszervező térképe, egy képszegmentáló algoritmus és egy fuzzy logikán alapuló klaszterezési algoritmus) hatékony implementációját elkészítsük a hardver/szoftver együttes tervezés módszertanának alkalmazásával. Ennek során a leglényegesebb lépés a különböző, egymásnak ellentmondó célok (sebesség, költség, méret) közötti megfelelő kompromisszum megtalálása volt [C6]. A projekt első fázisában ezt még kézzel végeztük, azonban később elkészültek a most bemutatott algoritmusok, melyek a munkánkat lényegesen egyszerűbbé és hatékonyabbá tették.

- Kutatócsoportunk részt vett az Európai Unió által finanszírozott EasyComp projektben, melynek célja új komponens alapú tervezési módszerek kidolgozása volt. A projekt keretében kutatócsoportunk kidolgozott egy új, komponens alapú hardver/szoftver együttes tervezési módszertant [C8, J2]. Ennek célja komplex rendszerek absztrakt építőelemekből történő tervezésének támogatása. A kompozíció egy intuitív grafikus környezet segítségével történik, és ennek során a komponensek implementációjával kapcsolatos részleteket a rendszer elrejti a tervező elől. Ez azt is jelenti, hogy bár egyes komponensek rendelkezhetnek egyaránt hardver és szoftver implementációval, a tervezőnek nem kell ezekkel a részletekkel törődnie. Ebből következően a particionálás feladatát automatizálni kell, és ezen a ponton kerültek alkalmazásra az itt bemutatott particionáló algoritmusok.

Ezúton is szeretnék köszönetet mondani az anyagi támogatásért a következő szervezeteknek: OTKA (T030178, T043329, T042559), Európai Unió (IST 1999-14191), Akita Industrial Development Center, Timber Hill LLC.

Hivatkozások

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[3] N. N. Binh, M. Imai, A. Shiomi, and N. Hikichi. A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate counts. In *Proceedings of the 33rd Design Automation Conference*, 1996.

[4] Th. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2nd edition, 2001.

[5] A. Dasdan and C. Aykanat. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(2):169–177, February 1997.

[6] L. Davis. *Handbook of genetic algorithms.* Van Nostran Reinhold, 1991.

[7] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. Hardware/software partitioning of VHDL system specifications. In *Proceedings of EURO-DAC '96*, 1996.

[8] R. Ernst, J. Henkel, and T. Benner. Hardware/software cosynthesis for microcontrollers. *IEEE Design and Test of Computers*, 10(4):64–75, 1993.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, 1982.

[10] R. K. Gupta. *Co-Synthesis of Hardware and Software for Digital Embedded Systems.* PhD thesis, Stanford University, December 1993.

[11] R. K. Gupta and G. de Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3):29–41, 1993.

[12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*, 1997.

[13] L. Hagen, J. H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on CAD*, 16(10):1199–1205, 1997.

[14] A. G. Hoffmann. The dynamic locking heuristic – a new graph partitioning algorithm. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 173–176, 1994.

[15] E. Hwang, F. Vahid, and Y. C. Hsu. FSMD functional partitioning for low power. In *Proceedings of the Design Automation and Test in Europe Conference*, 1999.

[16] A. Kalavade. *System-level codesign of mixed hardware-software systems.* PhD thesis, University of California, Berkeley, CA, 1995.

[17] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[18] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, 33(5):438–446, 1984.

[19] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen. LYCOS: The Lyngby co-synthesis system. *Design Automation of Embedded Systems*, 2(2):195–236, 1997.

[20] R. Niemann. *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems.* Kluwer Academic Publishers, 1998.

[21] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

[22] F. Vahid and D. Gajski. Clustering for improved system-level functional partitioning. In *Proceedings of the 8th International Symposium on System Synthesis*, 1995.

[23] W. Wolf. An architectural co-synthesis algorithm for distributed embedded computing systems. *IEEE Transactions on VLSI Systems*, 5(2):218–229, June 1997.

[24] W. Wolf. A decade of hardware/software codesign. *IEEE Computer*, 36(4):38–43, 2003.

[25] C.-W. Yeh, C.-K. Cheng, and T.-T. Y. Lin. A general purpose, multiple-way partitioning algorithm. *IEEE Transactions on CAD*, 13(12):1480–1487, 1994.

Publikációk

Külföldön megjelent idegen nyelvű folyóiratcikk

[J1] Z. Á. Mann and K. Kondorosi. Tracing system-level communication in distributed systems. *Software: Practice & Experience*, 34:727–755, 2004.

[J2] P. Arató, Z. Á. Mann, and A. Orbán. Extending component-based design with hardware components. *Elsevier Journal of Science of Computer Programming, Special Issue on New Software Composition Concepts*, accepted.

[J3] P. Arató, Z. Á. Mann, and A. Orbán. Algorithmic aspects of hardware/software partitioning. *ACM Transactions on Design Automation of Electronic Systems*, accepted.

[J4] P. Arató, Z. Á. Mann, and A. Orbán. Time-constrained scheduling of large pipelined datapaths. *Elsevier Journal of Systems Architecture*, submitted.

[J5] G. Ziegler, Z. Á. Mann, A. Orbán, Zs. Palotai, L. Grad, and A. Lőrincz. Three-Level Memory for Optimization. *Journal of Applied Soft Computing*, submitted.

[J6] P. Arató, Z. Á. Mann, and A. Orbán. Finding optimal hardware/software partitions. *IEEE Transactions on Computers*, submitted.

Magyarországon megjelent idegen nyelvű folyóiratcikk

[J7] I. Zs. Berta and Z. Á. Mann. Smart cards – present and future. *Híradástechnika, Journal on C5*, 2000/12, 24-29.

[J8] I. Zs. Berta and Z. Á. Mann. Implementing elliptic curve cryptography on PC and smart card. *Periodica Polytechnica, Series Electrical Engineering*, 46(1-2):47-73, 2002.

[J9] A. Orbán, Z. Á. Mann, and P. Arató. Time-constrained design of pipelined control-intensive systems. *Periodica Polytechnica, Series Electrical Engineering*, accepted.

Magyar nyelvű folyóiratcikk

[J10] Berta I. Zs. és Mann Z. Á. Programozható chipkártyák és azok biztonsága. *Magyar Távközlés*, 2000/4, 8-13.

Cikk szerkesztett könyvben

[C1] P. Arató, Z. Á. Mann, and A. Orbán. Genetic scheduling algorithm for high-level synthesis. *Intelligent Systems at the Service of Mankind, Volume 1, Selected publications of INES 2002 Conference*, Editors: Wilfried Elmenreich, J. Tenreiro Machado, Imre J. Rudas, 311-322, 2003. (Eredetileg: Proceedings of the IEEE 6th International Conference on Intelligent Engineering Systems, 2002.)

[C2] B. Golda, B. Laczay, Z. Á. Mann, Cs. Megyeri, and A. Recski. Implementation of VLSI routing algorithms. *Intelligent Systems at the Service of Mankind, Volume 1, Selected publications of INES 2002 Conference*, Editors: Wilfried Elmenreich, J. Tenreiro Machado, Imre J. Rudas, 349-360, 2003. (Eredetileg: Proceedings of the IEEE 6th International Conference on Intelligent Engineering Systems, 2002.)

Nemzetközi konferencia-kiadványban megjelent idegen nyelvű előadás

[C3] Z. Á. Mann and A. Orbán. Integrating formal, soft and diagrammatic approaches in high-level synthesis and hardware-software co-design. In *Proceedings of Informatik 2001, Workshop on Integrating Diagrammatic and Formal Specification Techniques*, volume I, pp. 649–654, 2001.

[C4] Z. Á. Mann, J. Calmet, and P. Kullmann. Testing access to external information sources in a mediator environment. *Proceedings of the 14th IFIP International Conference on Testing Communicating Systems*, Kluwer Academic Publishers, 111-126, 2002.

[C5] Z. Á. Mann and A. Orbán. Optimization problems in system-level synthesis. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 222-231, 2003.

[C6] P. Arató, Z. Á. Mann, and A. Orbán. Hardware-software co-design for Kohonen's self-organizing map. In *Proceedings of the IEEE 7th International Conference on Intelligent Engineering Systems*, 173-178, 2003.

[C7] P. Arató, S. Juhász, Z. Á. Mann, A. Orbán, and D. Papp. Hardware/software partitioning in embedded system design. In *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, IEEE Press, 197-202, 2003.

[C8] P. Arató, Z. Á. Mann, and A. Orbán. Component-based hardware-software co-design. In *Proceedings of the 17th International Conference on Architecture of Computing Systems, Lecture Notes in Computer Science (LNCS 2981)*. Springer, 169-183, 2004.

Magyar nyelvű konferencia-előadás

[C9] Berta I. Zs. és Mann Z. Á. Programozható chipkártyák kriptográfiai alkalmazása. *Networkshop*, 2001.

Nyomtatott egyetemi jegyzet

[U1] Z. Á. Mann, A. Orbán, A. Recski. Aufgaben zur theoretischen Informatik. *Számítástudományi és Információelméleti Tanszék, Budapesti Műszaki és Gazdaságtudományi Egyetem*, 85 o., 2001.

Nem publikációértékű munkák

Tudományos Diákköri dolgozat

[O1] Berta I. Zs. és Mann Z. Á. A hitelesség biztosításának lehetőségei intelligens smartcard segítségével. *Tudományos diákköri dolgozat* (TDK 1999: 1. díj, OTDK 2001: 2. díj), 48 o., 1999.

[O2] Berta I. Zs. és Mann Z. Á. Elliptikus görbéken alapuló nyilvános kulcsú kriptográfia elemzése chipkártyás és PC-s környezetben. *Tudományos diákköri dolgozat* (TDK 2000: 1. díj, OTDK 2001: különdíj), 41 o., 2000.

[O3] Mann Z. Á. CORBA alkalmazások nyomkövetése interceptor-ok segítségével. *Tudományos diákköri dolgozat* (TDK 2000: 1. díj, OTDK 2001: 3. díj), 43 o., 2000.

[O4] Mann Z. Á. és Orbán A. Új ütemező algoritmusok a magas szintű szintézisben. *Tudományos diákköri dolgozat* (TDK 2000: 1. díj, OTDK 2001: 1. díj), 56 o., 2000.

Poszter nemzetközi konferencián

[O5] Z. Á. Mann, A. Orbán, and P. Arató. Component-based Hardware/Software Co-Design. *Microsoft Research Academic Conference*, 2003.

[O6] Z. Á. Mann and A. Orbán. Hardware/software partitioning. *Design Automation and Test in Europe*, PhD Forum, 2004.

Nem teljes szöveggel megjelent konferencia-előadás

[O7] Z. Á. Mann. Tracing CORBA applications using interceptors. *OMG Information Day*, 2001.