NEM-STRUKTURÁLT TÉRHÁLÓKON ÉRTELMEZETT ALGORITMUSOK ABSZTRAKCIÓJA ÉS IMPLEMENTÁCIÓJA SOKPROCESSZOROS ARCHITEKTÚRÁKON

Reguly István Zoltán

Doktori disszertáció tézisei

Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Témavezetők:
Oláh András Ph.D
Nagy Zoltán Ph.D

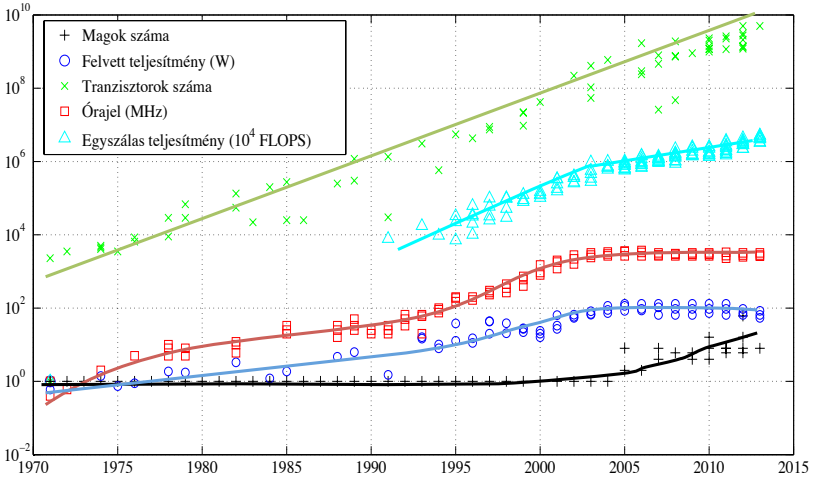
Budapest, 2014

1. Bevezetés

Lassan tíz éve már hogy a processzorok órajelének folyamatos növekedése - és ezzel a frekvencia-skálázódásnak köszönhető teljesítménynövekedés - véget ért. Ez arra kényszerítette a gyártókat, hogy a számítási kapacitás elvárt növekedését más eszközökkel tartsák fenn: ez leghatékonyabban a párhuzamosság növelésével volt elérhető. Megkezdődött a processzormagok számának növekedése, valamint visszatértek a vektor feldolgozó architektúrák: a modern CPU-k egyre hosszabb vektorokat támogatnak és a dedikált gyorsító hardverek, mint a GPU-k vagy az Intel Xeon Phi hatalmas mennyiségű párhuzamosságot tesznek lehetővé - és várnak el a hatékony működés támogatásához.

Míg a piaci igényeknek megfelelően a processzorok egyre nagyobb számítási sebességeket értek el, a memóriák fejlődése inkább a méret bővítésének irányába mozdult el, míg az elérési sebesség növelése lemaradt a fejlődésben. Míg az 1980-as években a memóriák elérési ideje és a számítási ciklusok hossza nagyon hasonló volt, mára már két-három nagyságrend különbség van köztük. Éppen ezért a sorosan végrehajtott algoritmusok esetében a szűk keresztmetszet az adatok mozgatása, melynek alacsony a sávszélessége és magas a késleltetése a processzor számítási képességeihez képest.

Az adatok mozgatásának költsége magas - mind az energiafogyasztás, mind a ciklusidők, mind pedig a késleltetés tekintetében - ami talán az egyik legnagyobb probléma mellyel a mai számítás-tudománynak szembe kell néznie, éppen ezért az adatok lokalitásának megoldása rendkívül fontos feladat. A modern architektúrák többszintes memóriahierarchiákat használnak, annak érdekében, hogy csökkenteni lehessen a chipen kívül eső memória műveletek számát, amelyek nagyságrendekkel költségesebbek, mint akár egy lebegőpontos szorzás elvégzése. A párhuzamos végrehajtást fel lehet használni a memória-elérések késleltetéséből adódó - kényszerűen felszabaduló - szabad számítási kapacitások felhasználására: a GPU-k például ezt a technikát hatékonyan alkalmazzák. Mindezek mellett, a számítási erőforrások szintjén, egyre nagyobb párhuzamosságra van szükség, hogy a számítási kapacitás növekedése fenntartható legyen a korábban megszokott ütemben: az órajelek növekedésének megszűnése miatt néhány év múlva a közepes méretű rendszerek is várhatóan annyi feldolgozó egységből, vagy magból, fognak állni, mint a mai legnagyobb szuperszámítógépek [1]. Végül pedig a hatalmas mennyiségű párhuzamosság és a több-szintű memóriahierarchia párosa elkerülhetetlenül oda vezet, hogy a párhuzamosan



1. ábra. A processzorok paramétereinek változása

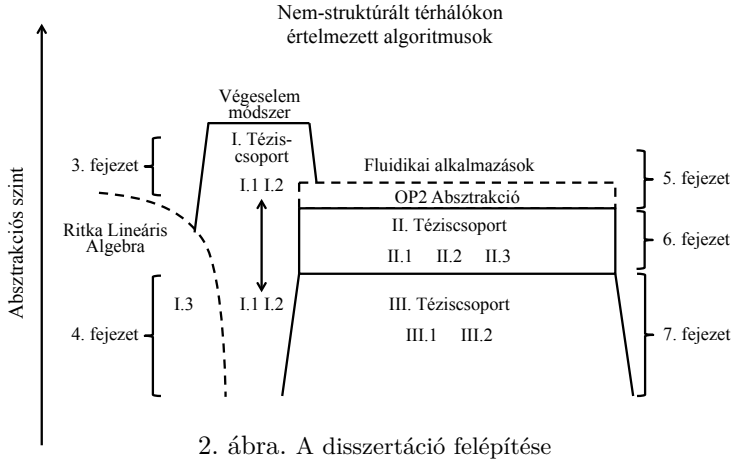
futó, egymással kooperáló számítások közt sebességbeli egyenlőtlenségek alakulnak ki. A párhuzamosság, a lokalitás és a munkaelosztás tehát a programozhatóság három legalapvetőbb kihívása [22] szerint.

A fentiek mellett ugyancsak korlátozó tényező, hogy a tudományos számítások területén dolgozók még mindig az évtizedekkel ezelőtt bevezetett - jól bevált és tesztelt - programozási nyelveket (C és Fortran) és a bennük implementált programokat és módszereket kívánják használni. A korabeli nyelvekben írt kód korábban könnyen hatékony gépi kóddá volt alakítható, hiszen a nyelv eszközkészlete és a hardver képességei közel álltak egymáshoz. Az évtizedek folyamán azonban a hardverek rengeteget fejlődtek, miközben a legnépszerűbb programozási nyelvek és absztrakciók keveset változtak. Mindez oda vezetett, hogy mára komoly különbség van a felhasználó programozásról alkotott mentális modellje és a hardver képességei közt. A fordítók természetesen hasonlóképp rengeteget fejlődtek az elmúlt időszakban, de évtizedek kutatása azt mutatja, hogy ennek a szemantikus résnek az áthidalása rendkívül nehéz.

Nagyszámú új programozási absztrakció és nyelv jelent meg, melyek mind ezeket a problémákat kívánják orvosolni, de sajnos nem áttörő sikerrel. Egyre nehezebb olyan tudományos számítógépes kódot írni mely képes kiaknázni a modern architektúrákban rejlő hatalmas teljesítmény potenciált. A nehézséget elsősorban az jelenti, hogy a megcélzott architektúrák részletes ismerete és ennek megfelelően specifikus optimalizációk alkalmazása szükséges a hatékony kód

készítéséhez. Mindeközben a feladatok algoritmikus bonyolultsága is növekszik, így a különböző szakterületek kutatóitól nem elvárható, hogy a saját területükön kívül még az új hardverek és új programozási módszerek terén is megfelelően mély ismeretekre tegyenek szert. Éppen ezért az informatika-tudományok kutatóinak körében komoly érdeklődés övezi a programozás absztrakciós szint emelésének problémáját: annak leírása, hogy *mit* kell kiszámítani és nem annak hogy *hogyan*, az implementációs kérdéseket ezzel ráhagyva a programozási nyelvre. Egy ilyen nyelv ideális esetben általános célú, könnyen használható, és hatékonyan kihasználja a hardver adottságait - azonban évtizedek kutatása ellenére ilyen nyelv a mai napig nem létezik. Ezért az elmúlt időszak kutatásai egy-egy jól körülhatárolható probléma-osztályt céloznak meg - sűrű és ritka lineáris algebra, strukturált és nem-strukturált térhálók, soktest problémák, Monte Carlo módszer. Az ezeken a probléma osztályokon definiált nyelvek (Domain Specific Languages, DSL-ek) megmutatták, hogy az általánosság elhagyásával a hatékonyság és a felhasználhatóság növelhető. Egy DSL absztrakciót ad egy adott probléma-osztályra, mely segítségével a számítási problémák egy magasabb szinten írhatók le. Ezek után az adott osztállyal kapcsolatos ismeretek jól felhasználhatók a számítások átrendezésére, a lokalitás növelésére, a probléma feldarabolására és ezzel a biztosítani lehet a feladat terhelések egyenletes elosztását, valamint a végrehajtás hardver függő leképzését, specifikus optimalizációk használatát. Egy tanulmány a berkley-i egyetemről [23] 13 jól elkülöníthető probléma-osztályt határozott meg. Ezek egyike a nem-strukturált térhálókon értelmezett algoritmusok: az Oxfordi Egyetemen fejlesztett OP2 [16] pedig egy olyan absztrakció és szoftver-rendszer mely ezt a specifikus probléma-osztályt célozza meg.

Az informatika-tudományok kutatói – köztük jelen dolgozat szerzője is – arra vállalkoznak, hogy az új hardverekkel és programozási módszerekkel kapcsolatos tapasztalataikat, eredményeiket olyan formába öntsék – illetve olyan absztrakciós szinteket definiáljanak – amely lehetővé teszi a más kutatási területeken dolgozók számára a folyamatosan növekedő számítási kapacitások hatékony kihasználását. Kutatásomat a programozhatóság jelen kori nagy kihívásai motiválták: a párhuzamosság, az adatlokalitás, a terheléselosztás és a hibatűrés. A disszertációm célja az, hogy bemutassa a nem-strukturált térhálókon értelmezett algoritmusokkal kapcsolatos kutatási eredményeimet. Munkám során különböző absztrakciós szintekről kiindulva vizsgálom a nem-strukturált térhálókkal kezelhető probléma megoldásokat. Ezen eredményeimet végül szá-



2. ábra. A disszertáció felépítése

mítógépes kódra képeztem le, különös figyelmet szentelve a különböző absztrakciós szintek, a programozási modellek, a végrehajtási modellek és a hardver architektúrák összhangjának. Futtatási eredményekkel bizonyítom, hogy a modern heterogén architektúrákon jelenlévő párhuzamosság és memória-hierarchia hatékonyan kihasználható kutatási eredményeim felhasználásával.

Az 2. ábra mutatja a dolgozat szerkezetét, munkám első része a végeselem-módszert vizsgálja, amely egy magasabb absztrakciós szintet jelent a dolgozat többi részéhez viszonyítva. Ez lehetővé tette, hogy egy magasabb szinten alakítsam át ezt a numerikus módszert a modern párhuzamos architektúrák adottságait figyelembe véve. Kutatásom megvizsgálja a végeselem integrációs lépése során a számítások és az adatmozgatás egyensúlyát, a végeselem módszer során használt adatstruktúrákat, valamint az iteratív megoldás egyik alapvető építőelemét, a ritka mátrixok és vektorok szorzatát.

A második részben – az első rész aránylag szűk kutatási területét kibővítettem, és – azokat az általános nem-strukturált térhálókon értelmezett algoritmusokat vizsgáltam, melyek az OP2 *Domain Specific Language (DSL)* [16] által definiált absztrakcióval leírhatóak. Ez az absztrakciós szint alacsonyabb, mint az első részben a végeselem-módszer esetében alkalmazott absztrakciós szint, ezért a numerikus módszerek magasabb szintű átalakítására itt nincs lehetőség. Ugyanakkor az OP2 ezen absztrakciója által lefedett terület lényegesen szélesebb, így magában foglalja a végeselem-módszert, de emellett sok más algoritmust is, így például a véges térfogat módszert is. Kutatásom

második része az OP2 absztrakciós szintjén definiált algoritmusok magas szintű transzformációival foglalkozik. Azt vizsgálja, hogy hogyan lehet a – tipikusan lineárisan megfogalmazott – számítási algoritmusok végrehajtását párhuzamos architektúrára transzformálni oly módon, hogy az adatlokalitás, a hibatűrés és az erőforrások kihasználtságának problémáját is megoldjuk. Ezen az absztrakciós szinten a kutatásaim még nem veszik figyelembe azt, hogy a végrehajtás milyen hardveren valósul meg. Ennek megfelelően a második rész eredményei megfelelő implementációs paraméterezéssel bármely végrehajtási környezetben felhasználhatóak.

Végül kutatásom harmadik része azzal foglalkozik, hogy az OP2 absztrakciós szintjén definiált algoritmusok miképp képezhetőek le automatikusan különböző, alacsonyabb szintű programozási nyelvekre, végrehajtási modellekre és hardver architektúrákra. Olyan leképezéseket adok meg, amelyeknek segítségével a számításokat egymásra rétegzett párhuzamos programozási absztrakciókon keresztül a mai modern, heterogén számítógépeken optimálisan végre lehet hajtani, kihasználva az összetett memória-hierarchiát és a többszintű párhuzamosságot. Kutatásom azt vizsgálja, hogy a nem-strukturált térhálók probléma osztálya esetében a magas szintű absztrakciók módszere valóban képes-e a modern hardverekben rejlő teljesítmény automatizált kiaknázására és mindezek mellett hatékonyan leegyszerűsíti-e az algoritmikus problémák megfogalmazását és implementációját a felhasználók számára.

2. Módszerek, eszközök

A kutatásaim során numerikus módszerek és analitikus módszerek széles spektrumát alkalmaztam, és különböző programozási nyelveket, modelleket, végrehajtási modelleket és hardvereket használtam fel. Egyik célom éppen az volt, hogy ezek interakcióját vizsgáljam a mai összetett rendszerekben. Kutatásom első része (I. Téziscsoport) a parciális differenciál-egyenletek megoldására használt népszerű végeselem-módszert vizsgálja, implementációim egy egyszerű Poisson problémát oldanak meg [21] alapján. A ritka lineáris egyenletrendszer megoldása során a konjugált gradiens iteratív módszert használtam, prekondicionálásra pedig a Jacobi és a Symmetric Successive Over-Relaxation (SSOR) módszert [24]. A ritka mátrixok és vektorok szorzását, mint a ritka lineáris algebra egyik alapvető építőelemét, részletesebben is tanulmányozom. A kutatásom első része bevezetésként szolgált a nem-strukturált térhálókon értelmezett algoritmusok világába, mely során rengeteg olyan tapasztalatot szereztem, amit a kutatásom későbbi részeiben felhasznál-

tam.

A kutatásom második és harmadik része az OP2 Domain Specific Language-re épül, melynek kutatását és fejlesztését Prof. Mike Giles az Oxfordi Egyetemen kezdte el [16], absztrakciója pedig elődjéből, az OPlus-ból [18] származik. A disszertációban bemutatott kutatást az OP2 felhasználásával megírt alkalmazásokkal próbálom ki a gyakorlatban: három, a véges-térfogat módszerre épülő fluid dinamikai szimulációt használnak. Az első, az Airfoil [25], a repülőgép szárny körüli áramlást modellezi, a második a sekélyvíz egyenleteket oldja meg, ez a cunamik szimulációjára használt Volna kód [20], és a harmadik egy nagyméretű ipari alkalmazás, a Hydra [19], melyet a Rolls-Royce használ repülőgép-turbinák tervezésére és szimulációjára. Az ezen kódok által használt numerikus módszereknek és struktúráknak nem én vagyok a szerzője, de az utóbbi kettő esetében én végeztem el az OP2 absztrakcióra való átültetés legjelentősebb részét. Az OP2 szoftver és az Airfoil tesztprogram elérhetősége: [17].

A számítógépes kódokat nagyrészt C vagy Fortran programozási nyelvben készítettem, a GPU-k esetében a CUDA nyelvi kiterjesztésének segítségével. Kód generálására és manipulálására a Python nyelvet használtam. Különböző párhuzamos programozási modelleket alkalmaztam a modern számítógép-rendszerekben jelenlévő többszintű párhuzamosság kihasználására. A legmagasabb szinten, az elosztott memória rendszerek esetében a Message Passing Interface-t (MPI) alkalmaztam, alacsonyabb - az egyes számítógépeken belüli - szinten pedig a Simultaneous Multi-Threading modellt (SMT) az OpenMP segítségével, a Single Instruction Multiple Threads modellt (SIMT) a CUDA segítségével, és a Single Instruction Multiple Data modellt (SIMD) a vektor utasítások segítségével implementáltam.

Hardverek és architektúrák széles spektrumát használtam fel az algoritmusok és programkódok tesztelésére, a teljesítmény mérésére. Kisebb rendszerek esetén egy-egy munkaállomást használtam, melyek két tokos Intel Xeon processzorokra épülnek (Westmere X5650, Sandy-Bridge E2640, Ivy-Bridge E2697-v2). A felhasznált gyorsító hardverek: egy Intel Xeon Phi 5110P, és NVIDIA Tesla GPU-k (C2070, M2090, K20, K20X, K40). A nagyobb számítási igényű mérések esetén különböző szuperszámítógépeket volt lehetőségem használni: HECToR (Nagy-Britannia nemzeti szuperszámítógépe, Cray XE6, 90112 AMD Opteron CPU maggal), Emerald (Nagy-Britannia legnagyobb GPU szuperszámítógépe, 372 NVIDIA M2090 GPU-val), illetve Jade (Az Oxfordi Egyetem 16 NVIDIA K20m GPU-ból álló rendszere). Az

időméréseket a szokásos UNIX-os rendszerhívások segítségével végeztem, általában a kezdeti beindulási idő torzító hatásának figyelmen kívül hagyásával (pl. fájlok olvasása), mivel a gyakorlatban ezek a szimulációk hosszú órákig futnak, melyhez képest ezek a kezdeti költségek elhanyagolhatóak. A legtöbb esetben az eredményeket 3-5 ismételt mérés átlagolásával számítottam ki. A dokumentált mérési eredmények között abszolút és relatív mérőszámok is szerepelnek, mint az elért sávszélesség (GB/s) vagy számítási teljesítmény (milliárd lebegőpontos művelet egy másodperc alatt - GFLOPS), valamint teljesítmény-növekedés egy referencia implementációhoz hasonlítva, mely az összes CPU magot hasznosítja.

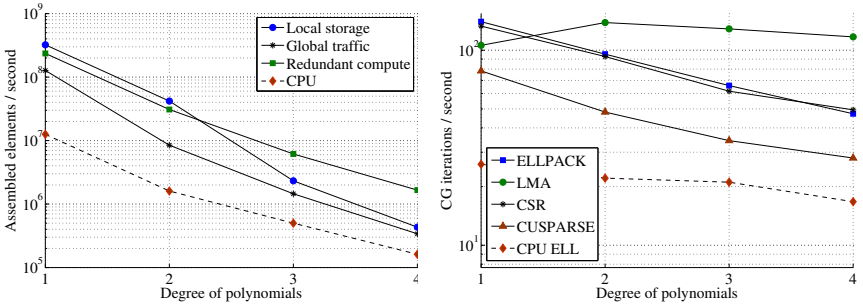
3. Tudományos eredmények

I. Téziscsoport - A végeselem módszer és a kapcsolódó ritka lineáris algebrai módszerek különböző leképzéseit adtam meg a GPU architektúrára, melyek segítségével a párhuzamosság, lokalitás és adatmozgatás problémáinak kezelését céloztam meg, és bemutattam a különböző megközelítések előnyeit és hátrányait, mind elméletben, mind gyakorlatban, valós implementációk esetén.

Kapcsolódó publikációk: [4, 9, 12, 13].

I.1. Tézis - A végeselem módszer integrációs lépésének olyan transzformációit vezettem be, melyek a használt tárhely, memórialokalitás, illetőleg a számítások mennyiségének egyensúlyát változtatják. Ezek segítségével az integrációs lépés új implementációit adtam meg a GPU hardveren, és megmutattam, hogy a redundáns számításokat végző megoldás teljesítménye már alacsony fokszámú elemeknél is versenyképes, valamint hogy sokkal jobban skálázódik magasabb fokszámú elemekre, a számítási teljesítmény visszaesése nélkül.

A végeselem módszer integrációs lépésének algoritmikus transzformációival olyan, az egyes elemeken értelmezett formulációkat adtam meg, melyek különböző tulajdonságokkal rendelkeznek a számítások mennyisége, a tárhelyigény, és a tér és időbeli lokalitás terén. Három variánst dolgoztam ki: (1 - *redundant compute*) ahol a külső iteráció a szabadságfokok párjain, a belső iteráció pedig a gauss kvadratúra pontjain megy végig, a Jacobi mátrix minden alkalommal való újraszámolásával. (2 - *local storage*) melyben az iterációk szervezése azonos (1)-gyel, de a Jacobi mátrixokat minden kvadratúra pontra előre kiszámolom és regiszterekben tárolom, majd a legbelső iterációban onnan olvasom ki őket. (3 - *global memory traffic*) mely használata más végeselem kódok-



(a) A végeselem integráció transzformációi

(b) Iteratív megoldás és adatstruktúrák

3. ábra. A GPU-ra leképzett végeselem módszer algoritmusok teljesítménye

ban elterjedt, ahol a külső iteráció a kvadratúra pontokon megy végig, a Jacobi mátrix egyszeri kiszámításával, és a belső iterációt a szabadságfokok párjain hajtja végre, így a merevségi mátrix elemeit többször is frissíti. A 3a ábrán bemutatott mérési eredményeken jól látható, hogy az első variáns hatékonyan skálázódik magas fokú elemekre, mivel csak a számítások mennyisége nő, ellentétben a másik két variánssal ahol vagy a lokális tárhely növekszik, vagy pedig a memória tranzakciók száma nő és lokalitása csökken. A CUDA implementáció segítségével, mely egy Poisson problémát old meg, megmutattam, hogy alacsony fokszámú elemek esetén az első és második variáns sebessége nagyon közel van egymáshoz, azonban magasabb fokszámok esetén az első akár nyolcszor gyorsabb mint a második, valamint legalább háromszor gyorsabb mint a harmadik, bármely fokszám esetén. Egy NVIDIA C2070 GPU az első variációval 400 GFLOPS-ot ér el [1], és akár tízszer gyorsabb, mint egy Intel Xeon X5650 12 magos szerver processzor, és 120-szor gyorsabb mint annak egy magja.

I.2. Tézis - Egy, a végeselem módszeren értelmezett adatstruktúrát vezettem be a GPU-ra, megmutattam, hogy ezen adatstruktúra kevesebb tárhelyigénnyel rendelkezik, és kevesebb adatmozgatást implikál a végeselem integráció és iteratív megoldás során, különböző fokszámok esetén. Az implementáció segítségével bizonyítom, hogy ez az adatstruktúra a jobb lokalitásnak köszönhetően magasabb teljesítményt ér el, mint a ha-

[1] Ugyanez a GPU 606 GFLOPS teljesítményt ér el egy sűrű mátrix-mátrix teszten

gyományos adatstruktúrák.

A végeselem módszer használata során a probléma nemstrukturáltsága jelenti az egyik legnagyobb kihívást, hiszen a mátrix összeállítása, valamint a lineáris rendszer megoldása során a memória elérések nem strukturáltak. Ha azonban a merevségi értékeket elemenként tároljuk, a vektor-elérés miatt transzponálva, akkor, ahogyan megmutattam, a memória elérések strukturáltak lesznek és az adatok írási konfliktusait az összeállítás helyett a megoldási lépésben kell kezelni, ahol ez sokkal hatékonyabban megoldható. Ez az eljárást a Lokális Mátrixok Módszere (Local Matrix Approach - LMA), amely elsősorban egy adattárolási formátum, de értelemszerűen megváltoztatja a végeselem módszer algoritmusainak egyes részleteit is. Ezt a módszert összehasonlítottam a hagyományos CSR és ELLPACK ritka mátrix tárolási struktúrákkal és feldolgozási eljárásokkal a GPU-n. Megmutattam, hogy az LMA akár kétszer magasabb teljesítményt adhat mind a mátrix összeállítása, mind pedig a rendszer iteratív megoldása során (3b ábra), annak köszönhetően, hogy kevesebb memóriát mozgat, jobb lokalitással. Egy olyan konjugált gradiens iteratív megoldót implementáltam, mely mind a három tárolási módszert támogatja, és prekondicionálóként a Jacobi vagy a Symmetric Successive Over-Relaxation (SSOR) módszert használja. A futási adatok elemzésén keresztül megmutatom, hogy az LMA módszer az esetek túlnyomó többségében - kiváltképp a magasabb fokszámok esetén - magasabb teljesítményt ad, mint a hagyományos módszerek.

I.3. Tézis - A ritka mátrixok és vektorok szorzásának (spMV) új paraméterezését adtam meg a GPU architektúrákon, valamint ezzel kapcsolatban olyan heurisztikát és gépi tanulási algoritmust mutattam be, melyek a lokalitást, párhuzamosságot és munkaelosztást javítják. A spMV elosztott memória rendszereken való kiszámításának javításához egy kommunikáció-elkerülő algoritmust is definiáltam és implementáltam.

A ritka mátrixok és vektorok szorzásának művelete a ritka lineáris algebra egy alapvető építőeleme, a legtöbb algoritmus használja valamilyen formában. Kutatásom alapjául a legelterjedtebb használt tárolási módszert, a Compressed Sparse Row (CSR) formátumot választottam, megmutattam, hogy a szorzás paraméterezhető a mátrix egy sorának a szorzandóval vett pont-szorzását elvégző szálak számával, valamint az egy CUDA blokkban lévő szálak számával és az egy blokkra jutó munkamennyiség változtatásával. Ezek megfelelő megválasztásával a legmodernebb CUSPARSE numerikus könyvtárnál lényegesen magasabb

1. táblázat. Számítási teljesítmény, sávszélesség és gyorsulás a CUSPARSE-hoz képest a 44 teszt mátrixon átlagolva

	CUSPARSE	Fix szabály	Gépi tanulás
32-bit GFLOPS/s	7.0	14.5	15.6
64-bit GFLOPS/s	6.3	8.8	9.2
Sávszélesség 32-bit GB/s	28.4	58.9	63.7
Sávszélesség 64-bit GB/s	38.7	54.0	56.8
Gyorsulás (32-bit)	1.0	2.14	2.33
Gyorsulás (64-bit)	1.0	1.42	1.50

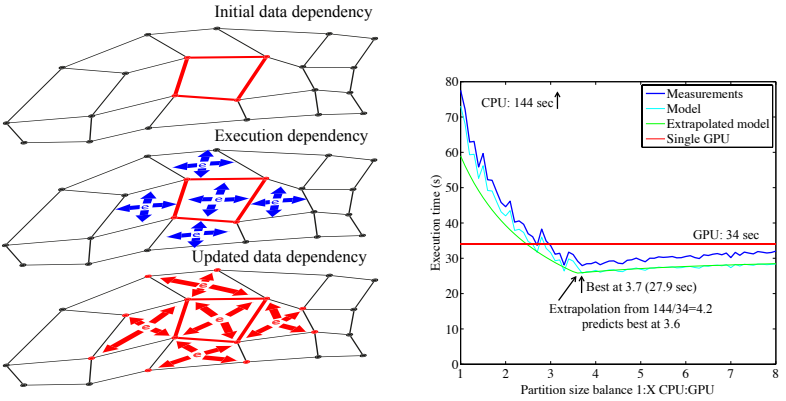
teljesítmény érhető el.

Tetszőleges ritka mátrixok esetén más és más paraméterezés vezet az optimális működéshez: bevezettem egy O(1) komplexitású heurisztikát (fix szabály) mely az optimálishoz közeli paramétereket határoz meg, és azonnali 1.4-2.1-szeres teljesítmény-növekedést eredményez. Felhasználva azt a megfigyelést, hogy az iteratív megoldók rendszerint ugyanazzal a mátrixal végeznek ismételten szorzásokat, bevezettem egy futási idejű gépi tanulási eljárást, mely az ismételt hívások során változtatja a paramétereket, és további 10-15%-os gyorsulást eredményez már 10 iteráció után. Végeredményben ez az eljárás a kimerítő kereséssel megtalált optimális paraméterezéstől átlagosan kevesebb, mint 2%-kal marad el. Az eredmények az 1. táblázatban találhatók. A szorzás elosztott memória rendszerben való végrehajtására egy olyan kommunikáció-elkerülő algoritmust definiáltam, mely átfedő gráf partíciók és redundáns számítások segítségével csökkenti a kommunikáció gyakoriságát, ezzel csökkentve a késleltetés negatív hatását, akár kétszeres teljesítménynövekedést elérve.

II. Téziscsoport - A tudományos számítások területén komoly kihívás a rendszer és szoftverszintű hibatűrő-képesség, az adatok lokalitásának kifejezése és kihasználása, valamint a heterogén hardverrendszerek kihasználása. Kutatásom második része ezeket a kihívásokat célozza meg a nem-strukturált térhálókon értelmezett számítások területén. Olyan új algoritmusokat és adat-transzformációkat vezetek be, melyek beékelődnek az algoritmusok absztrakt - OP2 szintű - leírása és azok hardver-specifikus kód-generálása közé, javítva a hibatűrőképességet, az adatlokalitást és különböző hardverek egyszerre való kihasználtságát.

Kapcsolódó publikációk: [2, 3, 5, 10].

II.1. Tézis - Bevezettem egy olyan hibajavító rendszert, mely a végrehajtás során bizonyos időközönként önállóan megtalálja azokat



(a) Adat- és végrehajtás-függőségek megállapítása *tiling* esetén

(b) Heterogén végrehajtás és annak modellezése

4. ábra. Az OP2-re épülő magas szintű transzformációk és modellek

a pontokat ahol az állapottér mérete minimális, és ezek mentésével biztosítja, hogy egy esetleges hiba esetén a rendszer automatikusan képes legyen ezt az állapotot visszaállítani és onnan folytatni a számításokat.

Ahogy a számítógép-rendszerek komponenseinek száma növekszik, úgy a tapasztalatok szerint a hardveres vagy szoftveres hibák előfordulása közti átlagos idő csökken - könnyen rövidebb lehet mint egy nagyobb szimuláció teljes futási ideje. Éppen ezért létfontosságú, hogy lehetőséget biztosítsunk az ilyen hibákat követően a számítások folytatására - az összes megelőző számítás újbóli elvégzése nélkül. Egy olyan hibajavító rendszert vezettem be, mely az OP2 absztrakción keresztül átadott információk alapján képes megállapítani az állapottér méretét a végrehajtás bármely pontján. Ez lehetővé tette, hogy az OP2 rendszer (1) bizonyos időközönként önállóan megtalálja és megadja azokat a pontokat ahol az állapottér minimális, és hogy (2) hiba esetén, az újraindítást követően, a számítások tényleges elvégzése nélkül a mentési pontig eljuttassa a végrehajtást, majd onnan azt adatveszteség nélkül folytassa. Ez a folyamat teljes egészében elrejthető a felhasználó elől, a program újraindításán kívül külső beavatkozást nem igényel.

II.2. Tézis - Algoritmust és implementációt adtam az OP2-ben a redundant compute tiling *technikára általános nem-strukturált számítások esetén, mely futási időben adat- és végrehajtás-függőségi analízis segítsé-*

gével, valamint késleltetett végrehajtás segítségével magasabb időbeli lokalitást tesz elérhetővé, ezzel kihasználva a modern architektúrákon jelenlévő cacheket.

Az adatlokalitás megadása és kihasználása a modern számítástudomány egyik legnagyobb kihívása: a tudományos számításokat megvalósító kódok nagy része azonban nem úgy lett megtervezve és megírva, hogy a számításokban rejlő lokalitás könnyen kihasználható legyen. A legtöbb ilyen kód rendszerint egy adott számítást végez el egy egész adathalmazon, majd egy következő számítást az előző számítás eredmény-adathalmazán, stb. Abban az esetben, amikor az adathalmaz mérete meghaladja a chipen lévő memória méretét, akkor rendkívül költséges adatmozgatási műveleteket kell végezni. A számítások olyan módon való átszervezése, hogy egyszerre csak a cache méretének megfelelő adathalmaz-részen dolgozzanak az algoritmusok, rendkívül nehéz az adatfüggőségek miatt. Egy olyan módszert dolgoztam ki és implementáltam, mely képes az OP2 magas szintű absztrakcióján keresztül definiált általános nem-strukturált térhálókat felbontani kisebb részekre, valamint a számításokat úgy átszervezni, hogy az figyelembe vegye az adatok és a végrehajtás függőségeit. A módszer alapötletét a 4a. ábra illusztrálja. A módszer biztosítja a különböző műveletek egymás után fűzését a probléma kisebb részein, és ezzel az adatok időbeli lokalitása javul. Ez az algoritmus a felhasználó beavatkozása nélkül alkalmazható bármely OP2-n keresztül definiált problémára.

II.3. Tézis - Új modellt definiáltam a nem-strukturált térhálókon értelmezett számítások heterogén architekúrákon való együttműködő végrehajtására, mely segítségével a különböző tulajdonságokkal bíró hardverek közti munkaelosztás optimalizálása valósítható meg. A modell hatékonyságát - és ezzel a modern heterogén rendszerek kihasználásának lehetőségét - az OP2 rendszerben való implementációval igazoltam.

A modern szuperszámítógépek egyre több esetben rendelkeznek olyan gyorsítóhardverekkel mint a GPU-k vagy a Xeon Phi. A legtöbb tudományos kód ezeken a rendszereken vagy csak a hagyományos processzorokat vagy csak a gyorsítókat használja hasznos számítások elvégzésére, mely az erőforrások pazarlását eredményezi. A teljes kihasználtsághoz tehát szükséges az elérhető erőforrások egyidejű és hatékony használata, figyelembe véve a különbözőségeiket. Bevezettem egy olyan új modellt, ami képes leírni a nem-strukturált térhálókon értelmezett algoritmusok heterogén rendszerekben való végrehajtását, figyelembe véve, hogy a térháló dekompozíciója során más tulajdonságokkal rendelkező hardve-

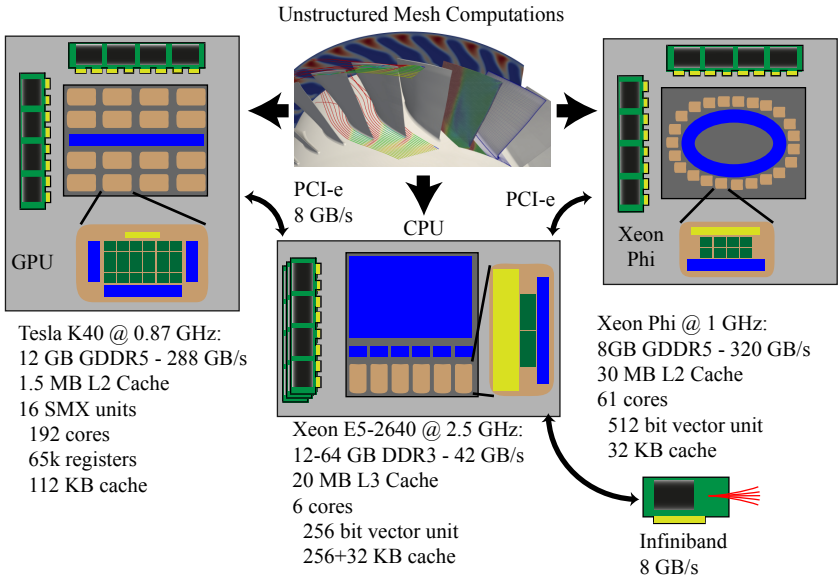
rek más méretű rész-problémákat kaphatnak, valamint hogy bizonyos számítások elvégzéséhez kommunikációra, szinkronizációra van szükség. Az OP2 rendszerben implementáltam a heterogén végrehajtás támogatását, ezzel igazolva a modell leíróképességét, valamint megmutattam és értelmeztem az összteljesítmény viselkedését, melyet a 4b ábra mutat.

III. Téziscsoport - A *Domain Specific Language*k szélesebb elterjedésének egyik akadálya volt, hogy eddig nem volt arra egyértelmű bizonyíték, hogy jól használhatóak, hatékonyak és hordozhatóak (jövő-biztosak) képesek lenni nagy, ipari szimulációk esetén is. Az Airfoil tesztprogram, a Volna cunami szimuláció és a Rolls-Royce Hydra ipari alkalmazás segítségével széles felhasználási körben egyértelmű bizonyítékot adtam arra, hogy a nem-strukturált térhálókon értelmezett számításokat, az OP2 magas szintű absztrakciójára - egyszer - leképezve, lehetséges különböző hardverek széles spektrumán optimálishoz közeli teljesítményt elérni, valamint hogy ezen leképzéssel a kódok fenntarthatósága és jövőbeli hatékonysága biztosítható.

Kapcsolódó publikációk: [2, 3, 5, 6, 8, 11, 14, 15].

III.1. Tézis - Bevezettem egy teljesen automatikus módszert mely képes a nem-strukturált térhálókon értelmezett számításokat a GPU architektúrára leképezni, felhasználva különböző adat és végrehajtási transzformációkat annak érdekében, hogy a hardver limitált erőforrásait, többszintű párhuzamosságát és memória-hierarchiáját közel optimálisan használja ki, és ezt kísérletileg igazoltam.

A végrehajtás GPU-kra való leképzése során a Single Instruction Multiple Threads (SIMT) modell és a CUDA nyelv használata szükséges. Egy olyan teljesen automatikus kódgenerálási eljárást mutattam be, mely az OP2 adat-transzformációival együttműködve lehetővé teszi a Kepler generációs GPU-kon az optimálishoz közeli teljesítmény elérését. Ezzel olyan kódot generáltam, mely a legújabb hardverekre szabott különböző optimalizációkat is felhasználja, a memória-rendszer adottságainak kihasználása érdekében. Ezeket a technikákat mindhárom fent említett alkalmazáson kipróbáltam, és megmutattam, hogy egy GPU-n 2-4-szeres gyorsulás érhető el egy 12-magos Intel Xeon processzorhoz képest. Elvégeztem a végrehajtás szoftveres és hardveres aspektusainak vizsgálatát, a különböző teljesítmény-metrikák segítségével (számítási teljesítmény, elért sávszélesség) bizonyítottam, hogy az optimálishoz közeli teljesítményt valóban sikerült elérni. Az erős és gyenge skálázódás vizsgálatán keresztül bemutattam, hogy az OP2 rendszer ezen



5. ábra. A nem-strukturált térhálókon értelmezett algoritmusok különböző hardverekre és szuperszámítógépekre való leképzésének problémája

automatikus módszer segítségével képes a GPU-kkal felszerelt modern szuperszámítógépeket hatékonyan használni.

III.2. Tézis - Bevezettem egy automatizált eljárást a számítások sokmagos CPU architektúrákra (valamint az Intel Xeon Phi-re) való leképzésére, mely lehetővé teszi, hogy a nagyszámú processzor magot és a széles vektor egységeket hatékonyan használják ki a nem-strukturált térhálókon értelmezett irreguláris számítások, és ezt kísérletileg igazoltam.

A modern CPU-k egyre hosszabb vektor egységeket tartalmaznak, használatuk elengedhetetlen a magas teljesítmény eléréséhez. Mivel a mai fordítók a legtöbb esetben nem képesek az ilyen strukturálatlan számítások automatikus vektorizációjára, ezért alacsony szintű vektor-utasításokat használtam (melyek megfeleltethetőek assembly utasításokkal), hogy biztosítsam a vektor egységek kihasználtságát. Bemutattam egy olyan kódgenerálási módszert, mely C++ osztályok és operátorok túlterhelésével lehetővé teszi a végrehajtás vektorizációját, az adatok vektorokba gyűjtésével, vektorműveletek végzésével és végül az adatok

vektorból való szétszórásával. Méréseket végeztem a legmodernebb Intel szerver processzorokon és az Intel Xeon Phi-n, melyek 1,5-2,5-szeres gyorsulást mutatnak a vektorizálatlan kódhoz képest. A teljesítményt leíró metrikák vizsgálatának segítségével megmutattam, hogy mely tényezők korlátozzák a teljesítményt különböző típusú számítások esetében, különböző hardvereken. Megmutattam továbbá, hogy ezek a módszerek képesek több ezer magig skálázódni, melyet a Hydra erős és gyenge skálázódásával igazoltam.

4. Az eredmények alkalmazhatósága

A végeselem módszerrel kapcsolatos eredményeim többféleképp is alkalmazhatóak. Sok területen egyre fontosabb az olyan algoritmusok tervezése melyek más számítási vagy adat-mozgatási tulajdonságokkal bírnak - az I.1 tézisben megfogalmazottakhoz hasonlóan. Eredményeim közvetlenül is felhasználhatóak egy általános célú végeselem szoftverben. Léteznek olyan végeselemes rendszerek, mint a ParaFEM [26], melyek a CPU-n egy, az LMA-hoz hasonló adatstruktúrát használnak, amennyiben a GPU-k támogatását be tervezik vezetni, eredményeim közvetlenül hasznosíthatók. A ritka mátrixok és vektorok szorzatával kapcsolatos eredményeim egy szélesebb területen alkalmazhatóak: a ritka lineáris algebrában. Az itt bemutatott heurisztika [9] azóta felhasználásra került az NVIDIA CUSPARSE [27] könyvtárban, és bár a futási időben végzett gépi tanulási módszerhez hasonlót jelenleg csak kevés szoftver használ, azonban a jövőben ez egyre fontosabb lehet, hiszen a különböző hardverek tulajdonságai egyre jobban eltérőek lesznek. Az NVIDIA-nál való gyakornokságom során én dolgoztam ki az AmgX [28] ritka lineáris megoldó rendszer elosztott memóriás működését, felhasználva a szerzett tapasztalatokat.

Az OP2 rendszer keretein belül elért eredményeim azonnal felhasználhatóak, és felhasználásra kerültek, olyan kódokban melyek az OP2-t használják: a Volna cunami szimulációs kódot Serge Guillas kutatócsoportja használja a University College London-ban, és az Indiai Tudományos Intézet Bangalore-ban, *uncertainty quantification* céljából, hiszen a cunamikat kiváltó földrengések pontos helye és intenzitása sokszor nem ismert. Hasonlóképp, az Rolls-Royce Hydra rendszert folyamatosan használják a repülőgép-turbinák tervezésére, az OP2-re átültetett változat adoptációja pedig folyamatban van. Eredményeim egy része (első sorban a második téziscsoportban) pedig más problémaosztályok esetén is felhasználhatóak, a strukturált térhálókat célzó

kutatásunkban többet már most, illetve fizikus és kémikus kutatókkal való együttműködésemben is [7] alkalmazunk.

5. Köszönetnyilvánítás

Mindenek előtt szeretném megköszönni témavezetőimnek, Oláh Andrásnak és Nagy Zoltánnak a segítségüket, vezetésüket és türelmüket, valamint Roska Tamásnak az inspiráló és gondolatteremtő beszélgetéseket melyek ebbe az irányba vezettek el. Köszönöm Garay Barnának a támogatását és példamutatását. Hálával tartozom Mike Giles-nak, hogy befogadott a kutatócsoportjába oxfordi tartózkodásom alatt, köszönöm támogatását és útmutatását.

Köszönöm az összes barátomnak és kollégámnak, hogy ilyen élvezetessé és felejthetetlenné tették ezt az elmúlt pár évet, akikkel a legbolondabb ötleteimet is meg tudtam vitatni: Józsa Csabának, Halász Gábornak, Feiszthuber Helgának, László Endrének, Gihan Mudaligenek, Carlo Bertolli-nak, Fabio Luporini-nek, Tuza Zoltánnak, Rudán Jánosnak, Zsedrovits Tamásnak, Tornai Gábornak, Horváth Andrásnak, Bihary Dórának, Borbély Bencének, Tisza Dávidnak és még sok másnak.

Köszönöm a Pázmány Péter Katolikus Egyetemnek és Szolgay Péter Dékán Úrnak hogy felvettek a doktori iskolába és végig támogattak, valamint az Oxfordi Egyetemnek. Köszönöm Jon Cohennek, Robert Strzodka-nak, Justin Luitjens-nek, Simon Layton-nak és Patrice Castonguay-nek az NVIDIA-nál töltött nyári gyakornokságot, rengeteget tanultam.

Örökké adósa leszek a családomnak, akik elviselték otthonlétemet és távollétemet, hogy végig támogattak minden elképzelhető módon.

A szerző folyóirat-publikációi

[1] M. B. Giles and **I. Z. Reguly**. “Trends in high performance computing for engineering calculations”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* (2014). Invited paper, accepted with minor revisions.

[2] G. R. Mudalige, M. B. Giles, J. Thiyagalingam, **I. Z. Reguly**, C. Bertolli, P. H. J. Kelly, and A. E. Trefethen. “Design and initial performance of a high-level unstructured mesh framework on heterogeneous parallel systems”. In: *Parallel Computing* 39.11 (2013), pp. 669–692. DOI: 10.1016/j.parco.2013.09.004.

[3] M. B. Giles, G. R. Mudalige, B. Spencer, C. Bertolli, and **I. Z. Reguly**. "Designing OP2 for GPU Architectures". In: *Journal Parallel and Distributed Computing* 73.11 (Nov. 2013), pp. 1451–1460. DOI: 10.1016/j.jpdc.2012.07.008.

[4] **I. Z. Reguly** and M.B Giles. "Finite Element Algorithms and Data Structures on Graphical Processing Units". In: *International Journal of Parallel Programming* (2013). ISSN: 0885-7458. DOI: 10.1007/s10766-013-0301-6.

[5] **I. Z. Reguly**, G. R. Mudalige, C. Bertolli, M. B. Giles, A. Betts, P. H. J. Kelly, and D. Radford. "Acceleration of a Full-scale Industrial CFD Application with OP2". In: *submitted to ACM Transactions on Parallel Computing* (2013). Available at: http://arxiv.org/abs/1403.7209.

[6] **I. Z. Reguly**, E. László, G. R. Mudalige, and M. B. Giles. "Vectorizing Unstructured Mesh Computations for Many-core Architectures". In: *submitted to Concurrency and Computation: Practice and Experience special issue on programming models and applications for multicores and manycores* (2014).

[7] L. Rovigatti, P. Šulc, **I. Z. Reguly**, and F. Romano. "A comparison between parallelisation approaches in molecular dynamics simulations on GPUs". In: *submitted to The Journal of Chemical Physics* (2014). Available at: http://arxiv.org/abs/1401.4350.

A szerző konferencia-publikációi

[8] G. R. Mudalige, **I. Z. Reguly**, M. B. Giles, C. Bertolli, and P. H. J. Kelly. "OP2: An Active Library Framework for Solving Unstructured Mesh-based Applications on Multi-Core and Many-Core Architectures." In: *Proceedings of Innovative Parallel Computing (InPar '12)*. San Jose, CA. US.: IEEE, May 2012. DOI: 10.1109/InPar.2012.6339594.

[9] **I. Z. Reguly** and M. B. Giles. "Efficient sparse matrix-vector multiplication on cache-based GPUs." In: *Proceedings of Innovative Parallel Computing (InPar '12)*. San Jose, CA. US.: IEEE, May 2012. DOI: 10.1109/InPar.2012.6339602.

[10] M.B. Giles, G.R. Mudalige, C. Bertolli, P.H.J. Kelly, E. László, and **I. Z. Reguly**. "An Analytical Study of Loop Tiling for a Large-Scale Unstructured Mesh Application". In: *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* 2012, pp. 477–482.

[11] **I. Z. Reguly**, E. László, G. R. Mudalige, and M. B. Giles. “Vectorizing Unstructured Mesh Computations for Many-core Architectures ”. In: *Proceedings of the 2014 International Workshop on Programming Models and Applications for Multicores and Manycores*. PMAM ’14. Orlando, Florida, USA: ACM, 2014. DOI: 10.1145/2560683.2560686.

A szerző egyéb publikációi

[12] **I. Z. Reguly** and M. B. Giles. “Efficient and scalable sparse matrix-vector multiplication on cache-based GPUs.” In: *Sparse Linear Algebra Solvers for High Performance Computing Workshop*. July 8-9, Warwick, UK, 2013.

[13] **I. Z. Reguly**, M. B. Giles, G. R. Mudalige, and C. Bertolli. “Finite element methods in OP2 for heterogeneous architectures”. In: *European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2012)*. September 10-14, Vienna, Austria, 2012.

[14] **I. Z. Reguly**, M. B. Giles, G. R. Mudalige, and C. Bertolli. “OP2: A library for unstructured grid applications on heterogeneous architectures”. In: *European Numerical Mathematics and Advanced Applications (ENUMATH 2013)*. August 26-30, Lausanne, Switzerland, 2013.

[15] **I. Z. Reguly**, G. R. Mudalige, C. Bertolli, M. B. Giles, A. Betts, P. H. J. Kelly., and D. Radford. “Acceleration of a Full-scale Industrial CFD Application with OP2”. In: *UK Many-Core Developer Conference 2013 (UKMAC’13)*. December 16, Oxford, UK, 2013.

Kapcsolódó publikációk

[16] M. B. Giles, G. Mudalige, Z. Sharif, G. Markall, and P. H. J. Kelly. “Performance Analysis and Optimization of the OP2 Framework on Many-Core Architectures”. In: *The Computer Journal* 55.2 (2012), pp. 168–180.

[17] *OP2 GitHub Repository*. https://github.com/OP2/OP2-Common. 2013.

[18] P. I. Crumpton and M. B. Giles. "Multigrid Aircraft Computations Using the OPlus Parallel Library". In: *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers* (). A. Ecer, J. Periaux, N. Satofuka, and S. Taylor, (eds.), North-Holland, 1996., pp. 339–346.

[19] Michael B. Giles, Mihai C. Duta, Jens-Dominik Müller, and Niles A. Pierce. "Algorithm Developments for Discrete Adjoint Methods". In: *AIAA Journal* 42.2 (2003), pp. 198–205.

[20] Denys Dutykh, Raphaël Poncet, and Frédéric Dias. "The VOLNA code for the numerical modeling of tsunami waves: Generation, propagation and inundation". In: *European Journal of Mechanics - B/Fluids* 30.6 (2011), pp. 598 –615. ISSN: 0997-7546. DOI: j.euromechflu.2011.05.005.

[21] M. S. Gockenbach. *Understanding and implementing the finite element method*. SIAM, 2006. ISBN: 978-0-89871-614-6.

[22] B. Dally. "Power, Programmability, and Granularity: The Challenges of ExaScale Computing". In: *Proceedings of the Parallel Distributed Processing Symposium (IPDPS'11)*. 2011, pp. 878–878. DOI: 10.1109/IPDPS.2011.420.

[23] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. *The Landscape of Parallel Computing Research: A View from Berkeley*. Tech. rep. UCB/EECS-2006-183. EECS Department, University of California, Berkeley, Dec. 2006. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.

[24] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. ISBN: 0898715342.

[25] M. B. Giles, D. Ghate, and M. C. Duta. "Using Automatic Differentiation for Adjoint CFD Code Development". In: *Computational Fluid Dynamics Journal* 16.4 (2008), pp. 434–443.

[26] I.M. Smith, D.V. Griffiths, and L. Margetts. *Programming the Finite Element Method*. Wiley, 2013. ISBN: 9781118535936. URL: http://books.google.co.uk/books?id=iUaaAAAAQBAJ.

[27] NVIDIA. *cuSPARSE library, last accessed Dec 20th*. http://developer.nvidia.com/cuSPARSE. 2012.

[28] *NVIDIA AmgX Library*. https://developer.nvidia.com/amgx. 2013.