Magas szintű absztrackciók kihasználása domén-specifikus nyelvek képességeinek és teljesítményének javítása érdekében

Balogh Gábor Dániel

A PhD Disszertáció tézisei

Témavezető:

Dr. Reguly István

Pázmány Péter Katolikus Egyetem

Roska Tamás Műszaki és Természettudományi Doktori Iskola

Budapest, 2023

1 Bevezetés

A számítási technológia folyamatos innovációja alapvetően befolyásolja a programozók és a tudósok által használt módszereket nagy számítási teljesítmény elérésének érdekében. Míg húsz évvel ezelőtt néhány GFLOP/s elérése volt a cél egyetlen CPU-n belül, a mai legerősebb szuperszámítógépek másodpercenként több millió hardverszál használatával műveletek trillióit hajtják végre. Ahhoz, hogy lépést tudjunk tartani az exponenciális növekedéssel alapvető változásokra volt szükség mind a hardver, mind a szoftver terén.

A 2000-es évek elején a teljesítmény az egyetlen szuperskalár CPU-ra való optimalizálás körül forgott. A programozók a késleltetés minimalizálása és az utasítás végrehajtás maximalizálása érdekében olyan technikákat használtak, mint a ciklus unrolling, a prefetching és a cache-tiling. A többmagos chipek megjelenésével a párhuzamosság egy új dimenziót hozott be a teljesítmény növelés lehetőségeibe, ennek hatékony kihaználása új, párhuzamos végrehajtásra tervezett algoritmusokat követelt meg. A megosztott memória és a szinkronizáció nélkülözhetetlenné váltak, ami gondos ütemezést és összehangolást tett szükségessé a verseny helyzetek által kiváltott szűk keresztmetszetek elkerülése érdekében.

A grafikus processzorok bemutatták hatalmas számítási képességeiket, és az olyan közelítési problémákat, mint a renderelés és a gépi tanulás, nagymértékben adatpárhuzamos feladatokká alakították át. A programozási gondolkodásmód ennek köszönhetően jelentős változáson esett át, és az adatpárhuzamos gondolkodásmód került középpontba. Az utasításközpontú CPU-modellek háttérbe szorulásával a számítási teljesítmény orientált sokmagos architektúrák nyertek teret. A GPU-kernelek absztrakcióját szolgáló könyvtárak jelentek meg az OpenCL-hez és a CUDA-hoz hasonló keretrendszereken keresztül, megkönnyítve az algoritmusok új hardverekre portolására irányuló erőfeszítéseket.

A tudományos számítástechnikai területeken egyre nagyobb teret hódítanak a domén-specifikus nyelvek (DSL-ek), mivel az általános célú nyelvekkel szemben jelentős produktivitásnövekedést kínálnak. Egy DSL egy magas szintű absztrakciót határoz meg, amely egy egyedi nyelvet vagy beágyazott nyelvet biztosít a megoldandó probléma leírására a problématerületre jellemző kifejezésekkel. A doménen belüli közös minták és struktúrák megragadásával a DSL lehetővé teszi a szakterületet ismerő programozók számára, hogy a megoldandó problémákat természetes módon, a terület specifikus kifejezések és absztrakciók használatával fejezzék ki, ezzel az absztrakciós szintet megemelik az általános célú nyelvekhez képest. Azáltal, hogy a szakterületen belüli szakértelmet absztrakt szintaxisba és szemantikába foglalják, a DSL-ek lehetővé teszik, hogy a programozási nyelvek szakértői helyett a szakterülethez tartozó szakértők írjanak programokat. Ezáltal, a DSL-ek középutat biztosítanak az általánosság és a specifikusság között, ami a sejtbiológiától a repülőgép-technikai szimulációig számos területen hasznosnak bizonyult. A beágyazott DSL-ek vagy eDSL-ek egy általános célú nyelv, például a C++ vagy a Fortran hagyományos könyvtáraként működnek az alkalmazásfejlesztő számára, és forráskód transzformációs technikákat használnak a célspecifikus, nagy teljesítményű implementációk biztosítására. Ez lehetővé teszi, hogy a DSL a területre szabott optimalicációkat nyújtson, miközben újra felhasználja az általános célú nyelv meglévő fordítás- és futásidejű infrastruktúráját.

Az első exascale szuperszámítógépek az elmúlt néhány évben jelentek meg, amelyek egyidejűleg több trillió szálat használnak. Többé már nem képes egyetlen eszköz ilyen teljesítményt nyújtani, elosztott rendszerek, klaszterek szükségesek kifinomult hálózati megoldásokkal. Eközben a hardverek sokfélesége egyre nő, ahogy az olyan gyorsító technológiák, mint az FPGA-k is bekapcsolódnak az exaflopokért folytatott küzdelembe. A nagy teljesítmény minden korábbinál jobban függ a szoftver azon képességétől, hogy a benne rejlő párhuzamosságot hordozható módon, az architektúra sajátosságait kerülve tudja kifejezni.

A világ legerősebb számítógépeit működtető, gyorsan változó hardverek és programozási modellek miatt a teljesítmény hordozhatósága és a produktivitás került a jövőbiztos nagy teljesítményű szoftverekről szóló viták középpontjába. Ebben a folyamatosan változó környezetben a jövőbiztos alkalmazások a teljesítményhordozható alkalmazások szinonimájává váltak, ahol a végső álom az összes jelenlegi és jövőbeli hardver támogatása a legjobb teljesítménnyel, egyetlen forráskódból [1]. A doménspecifikus nyelvek és a deklaratív programozás ígéretesek ezen a területen, hiszen az absztrakciót növelve szétválasztják a teljesítmény és a számítások leírásának szempontjait.

A párhuzamos és nagy teljesítményű számítástechnikában (HPC) a DSL-ek különösen ígéretesnek bizonyultak két kulcsfontosságú kihívás megoldására. Először is, megkönnyítik a teljesítmény hordozhatóságát a hadver alacsony szintű részleteitől elvonatkoztató, architektúra-agnosztikus problémaleírások révén. Lehetővé teszik az alkalmazásfejlesztő számára, hogy leírja a kiszámítandó problémát ahelyett, hogy azt írná le, hogy hogyan kell a problémát kiszámítani. Másodszor, segítik az alkalmazás fejlesztő produktivitását azáltal, hogy az algoritmusok kifejezésének absztrakciós szintjét megemelik, miközben megtartják az optimalizálási lehetőségek feletti uralmat. Ezek az előnyök együttesen ösztönözték az elmúlt évek kiterjedt kutatásait a HPC-alkalmazások DSL-jeivel kapcsolatban.

Az egyik ilyen domén-specifikus nyelvcsalád az Oxford Parallel Domain-Specific Languages, amely két aktív könyvtárból vagy beágyazott DSL-ből áll: OPS [2] és OP2 [3]. Ezek a könyvtárak a C/C++ és Fortran nyelvekbe ágyazott parciális differenciálegyenlet (PDE) megoldóknak szánt DSL absztrakciókat nyújtanak. A tudósok és domén szakértők ezekben a DSL-ekben a PDE-megoldókat magas szintű párhuzamos ciklusokon keresztül és elemi kerneleken keresztül fejezik ki, miközben a párhuzamos végrehajtás részleteivel és az adatmozgással nem kell törődniük. A fordítás során a könyvtárak egy kódgenerálási lépést használnak a párhuzamos ciklusok ardver specifikus párhuzamos imple-

mentációinak létrehozására, optimalizációk széles skáláját alkalmazva.

A fő motivációm az, hogy továbbfejlesszem a doménspecifikus nyelvek által kínált lehetőségeket, így a teljesítmény hordozhatóságát és produktivitást új területeken és problémaosztályokban is elérhetővé tegyem. Kutatásaim középpontjában a strukturált és strukturálatlan hálókon végzett számítások állnak. Disszertációm célja, hogy bemutassam a DSL-ek által támogatott problémaosztályok bővítésében, valamint a DSL-ek által használt kódgenerálási lépések robusztusságának és bővíthetőségének javításában elért eredményeimet.

A disszertációm első része a DSL-ekben vagy aktív könyvtárakban alkalmazott forráskódból forráskódra történő transzformációs technikákkal foglalkozik. A kódgenerálási lépések bővíthetősége és komplexitása kritikus fontosságú a DSL-ek hosszú élettartama szempontjából. A doménspecifikus modellből kiindulva ugyanannak a struktúrának vagy kódnak a generálása a generált kód jelentős részét teszi ki, és a kódgenerátor implementálása és fenntartása nehéz feladat. Első tézisemben a strukturálatlan hálókon végzett számítások heterogén hardverre történő optimális leképezésére koncentráltam. Bemutatom az OP2 DSL-en a párhuzamosítási vázon alapuló kódgenerálás módszerét, és összehasonlítom a különböző programozási modellek és nyelvek teljesítményét.

A DSL-ek kódgenerálási lépésén végzett munkám után a strukturált hálós alkalmazások támogatásának kiterjesztésén dolgoztam. A második és a harmadik tézis az OPS DSL bővítésére összpontosít. A második részben az OPS DSL-t kiegészítettem egy, az Alternating-Direction Implicit módszerhez [4] alkalmas, teljesítményben hordozható és skálázható lineáris megoldókönyvtár támogatásával. Az ADI alkalmazások a strukturált hálós számítások fontos speciális esete, ahol a stencillel történő számítások mellett a megoldások közelítése lineáris megoldók segítségével történik. Így az ADI-alkalmazások támogatásához a sok tridiagonális egyenletrendszer egyidejű megoldásának támogatására is szükség van. A DSL-ek egyik legfontosabb jellemzője az általánosság, hogy egy teljes tartományt a lehető legszűkebb absztrakcióval támogassanak.

A kötegelt-tridiagonális megoldók támogatása esetében az absztrakció elég szűk területre fókuszál, így közvetlen könyvtári támogatást tudunk nyújtani CPU- és GPU-klaszterekhez. A második rész a kötegelt-tridiagonális megoldók MPI-skálázható megoldó algoritmusainak kihívásaival foglalkozik a pontos megoldás megtalálásához szükséges kommunikáció lokalizálásán és minimalizálásán keresztül, valamint az egzakt és iteratív megoldók kommunikációs és skálázási tulajdonságainak javításán keresztül.

Számos tudományos számítástechnikai alkalmazás azonban nemcsak a PDE megoldását igényli, hanem a kimenetek érzékenységi információit is néhány bemeneti változóval kapcsolatban. Az algoritmikus differenciálást (AD) a bizonytalanság számítás és a design optimalizálás egyik kulcsfontosságú alaptechnikájaként ismerik el, mivel a függvény kompozíció matematikáját kihasználva képes automatikusan és pontosan kiszámítani a számítógépes programok deriváltjait. Az AD-hez azonban hiányoznak a teljesítményhordozható megoldások, és nincs olyan könyvtár, amely alkalmazható lenne az olyan DSL-ekre, mint az OPS. Továbbá ezek az eszközök nem veszik figyelembe a teljesítménykritikus optimalizációkat, amelyeket a modern, masszívan párhuzamos processzorok, jellemzően a sokmagos GPU-gyorsítók igényelnek az iparilag releváns alkalmazásokhoz. Kutatásom harmadik része az OPS kiterjesztésére összpontosít a adjoint módú AD támogatással, a magas szintű strukturált hálós kód automatikus leképezésével többmagos CPU-kra és sokmagos GPU-kra (III.1. és III.2. tézis), OPS-en kívüli eszközök, például a II. téziscsoportból származó lineáris megoldók további támogatásával, és végül egy absztrakcióval, amely a differenciálás memóriaigényét checkpoint-ok készítésével és újraszámítással szabályozza (III.3. tézis). Bemutatom az elkészült modell teljesítményét és a futásidőhöz hozzájáruló fő tényezőket, valamint az adjoint ciklusok ütemezése során a domén-specifikus információk felhasználásának előnyeit több reprezentatív alkalmazáson.

2 Módszerek

Kutatásom első része az OP2 domain-specifikus nyelv [3] alapján készült, amely egy magas szintű absztrakciót nyújt a strukturálatlan hálós alkalmazások megoldásához, egy API-t definiál a számítási kernelek leírásához, a párhuzamosság összehangolásához szükséges összes információval. Az új forráskód generátor megvalósítása a Clang LibTooling könyvtárának refaktorálási eszköz támogatásán alapul. A generált kód helyességét és teljesítményét két, az OP2 absztrakcióval írt alkalmazáson teszteltük: egy Airfoil nevű, egy repülőgép szárnya körüli légáramlást szimuláló benchmarkon és egy Volna [5] nevű cunami-szimulációs alkalmazáson.

Disszertációm második része sok tridiagonális egyenletrendszer egyidejű elosztott megoldására összpontosít, különös tekintettel az alternáló irányú implicit módszer alkalmazásaira [4]. Munkánk alapját a Tridsolver [6] megoldó könyvtár képezi. Az egzakt iteratív PCR algoritmust [7] a TridiagLU könyvtár [8] elosztott kommunikációs stratégiáival kombináltuk.

Végül kutatásom harmadik része az OP-DSL család másik doménspecifikus nyelvére, az OPS-re [2] és a kimenetek érzékenységi (matematikai derivált) információinak kiszámítására összpontosít. Az OPS felépítése és absztrakciója hasonló az OP2-éhez, de az OPS strukturált hálót céloz meg stencil alapú ciklusok segítségével. Az OPS-ben írt alkalmazásokhoz fordított (adjoint) módú algoritmikus differenciálást (AAD) alkalmaztunk. Három alkalmazást használtunk az AAD és az OPS teljesítményének elemzésére: egy 2D-s kód, amely a Poisson-egyenletet oldja meg Jacobi-iterációkkal, közvetlenül alkalmazva az AAD-t és fix-pont iterációkat [9], a CloverLeaf[10] egy minialkalmazás, amely a kompresszibilis Euler-egyenleteket oldja meg egy rácson explicit, másodrendű módszerrel, valamint egy 2D-s konvekciós-diffúziós egyenletet másodrendű ADI időlépéssel megoldó kódot [11].

Az alkalmazásokat és a könyvtárak kiegészítéseit C++ nyelven, a

GPU-kat célzó CUDA nyelvi kiterjesztéssel kombinálva valósítottuk meg. Míg az OP2 kódgenerátora C++ nyelven, addig az OPS AAD-támogatásának kódgenerátora Python nyelven készült. A node-ok közti kommunikációra az NCCL és MPI könyvtárakat használtam az elosztott memóriás párhuzamossághoz, a megosztott memóriás párhuzamossághoz pedig az OpenMP-t és a CUDA-t használtam.

A teljesítménymérésekhez különböző hardver architektúrákat és platformokat használtam. A Tridsolver könyvtár skálázási teljesítményének mérésére az Egyesült Királyság két HPC-rendszerét használtuk: ARCHER2[1], egy CrayEX rendszer AMD Rome CPU-kkal (nodeonként 2×64 maggal) és 256 GB-nyi RAM, és Cirrus[2], egy HPE/SGI rendszer 36 node-al, amelyek mindegyike 4 darab NVIDIA V100-as 16 GB-os GPU-val rendelkezik, amik NVLinkkel vannak összekapcsolva, és FDR Infinibanddel a csomópontok között. Az AAD teljesítményének értékeléséhez az OPS-ben az OpenMP méréseket egy Intel(R) Xeon(R) Gold 6226R at 2, 9 GHz CPU egyetlen socketén, hyperthreding nélkül és 376 GB RAM-mal, a CUDA méréseket pedig egy AMD EPYC 7F72 24-magos processzorral és egy NVidia A100 GPU-val, 40 GB RAM-mal végeztük. A legtöbb esetben a közölt futási idők 10 ismétlődő futtatás átlagolásának eredményei.

3 Új tudományos eredmények összefoglalása

I. Tézis

Egy új páhuzamosítási váz alapű fordító réteget terveztem nem struktúrált hálókon végzett ciklusok hardverre való leképezésére. Az új réteg javít az OP2 transzlálási lépésének stabilitásán és robosztusságán, CPU és GPU klaszterek támogatására alkalmas kódok generálásával, amelyek kihasználják a memória lokalitást. A generált kód teljesít-

[1] https://www.archer2.ac.uk/
[2] https://www.cirrus.ac.uk/

ményét reprezentatív alkalmazásokon demonstráltam, valamint összehasonlító elemzést végeztem a különböző programozási nyelvek és fordítók párhuzamos ciklusok hatékonyságára gyakorolt hatásáról.

A tézishez tartozó publikációk: [J1, C3, C4]

Az OP2 API úgy épül fel, hogy az elemzés fázis számára megkönnyítse az egyes ciklusokra vonatkozó releváns információk kinyerését, amelyek leírják, hogy milyen számítási és memória-hozzáférési mintákat fognak használni - ez szükséges a különböző architektúrákra és párhuzamosításra irányuló kódgeneráláshoz. A `op_arg_dat` minden részletet megad arról, hogy egy `op_dat` adataihoz hogyan férnek hozzá a ciklusban. Ezen információk birtokában a `op_par_loop` hívás tartalmazza a számítási ciklusra vonatkozó összes szükséges információt a párhuzamosítás elvégzéséhez. Világos, hogy az absztrakciónak köszönhetően a párhuzamosítás csak néhány paramétertől függ, mint például a ciklusban közvetetten elérhető adatok vagy redukciók megléte, valamint az optimalizálásnak kedvező adatelérési módok.

Az a tény, hogy csak néhány paraméter határozza meg a párhuzamosítást, azt jelenti, hogy két számítási ciklus esetén a generált párhuzamos ciklusok ugyanazokkal a kódsorokkal rendelkeznek, és csak kis kódrészek térnek el egymástól. A generált párhuzamos ciklusok azonos kóddarabjai a generálandó célkód fontos tervrajzaként szolgálnak. Ez elvezet bennünket ahhoz az ötlethez, hogy használjuk egy dummy ciklus párhuzamos implementációját (az invariáns darabokkal), és a kódgenerálási folyamatot ennek a párhuzamos ciklusnak a refaktorálásaként vagy módosításaként végezzük el. Az 1 ábra egy olyan párhuzamos váz részletét szemlélteti, amelyet a generált OpenMP implementációból indirekt ciklusok esetén kivehetünk. A kódgenerátor ezt a dummy párhuzamos ciklust vázként (vagy sablonként) használhatja, és módosíthatja a kívánt számítási ciklusjelölt generálásához. Hasonló sablonokat képzelhetünk el az összes cél párhuzamosításhoz. Ez a megközelítés csökkentheti az új célok bevezetésének költségeit, mivel

```
1  // elemental kernel function
2  void skeleton(double * __restrict__ d) {}
3
4  void op_par_loop_skeleton(char const *name,
5                            op_set set,
6                            op_arg arg0) {
7    //number of arguments
8    int nargs = 1; op_arg args[1] = {arg0};
9    int ninds = 1; op_arg inds[1] = {0};
10
11   /*----------- Invariant code -----------------*/
12   int set_size =
13       op_mpi_halo_exchanges(set, nargs, args);
14   op_plan *Plan = op_plan_get(name, set, 256, nargs,
15                               args, ninds, inds);
16   int block_offset = 0;
17   for (int col = 0; col < Plan->ncolors; col++) {
18     if (col == Plan->ncolors_core)
19       op_mpi_wait_all(nargs, args);
20     int nblocks = Plan->ncolblk[col];
21     #pragma omp parallel for
22     for(int blockIdx = 0; blockIdx<nblocks;
23         blockIdx++) {
24       int blockId =
25           Plan->blkmap[blockIdx +block_offset];
26       int nelem = Plan->nelems[blockId];
27       int offset_b = Plan->offset[blockId];
28       for(int n = offset_b; n<offset_b+nelem; n++) {
29   /*-------------------------------------------*/
30         // Prepare indirect accesses
31         int map0idx =
32             arg0.map_data[n * arg0.map->dim + 0];
33         // set up pointers, call elemental kernel
34         skeleton(&((double *)arg0.data)[2*map0idx]);
35       }
36     }
37   }
38 }
```

Figure 1: OpenMP párhuzamosítási sablon (részlet) - indirekt kernelek

csak egy vázként használható dummy ciklus implementálására van szükség ahelyett, hogy a teljes kernel kódgenerálási útvonalakat kellene megvalósítani. Ugyanakkor, mivel egy ciklus implementálása sokkal egyszerűbb és kevesebb a hiba lehetőség, mint a generáló kód megírása, ez a megközelítés csökkenti annak kockázatát, hogy a párhuzamos ciklusok kódjának invariáns részeiben hibák jelenjenek meg.

A párhuzamosítási vázak alapján a párhuzamos ciklus kódgenerálása refaktorálási lépésnek tekinthető. A Clang LibTooling könyvtára nagysz-

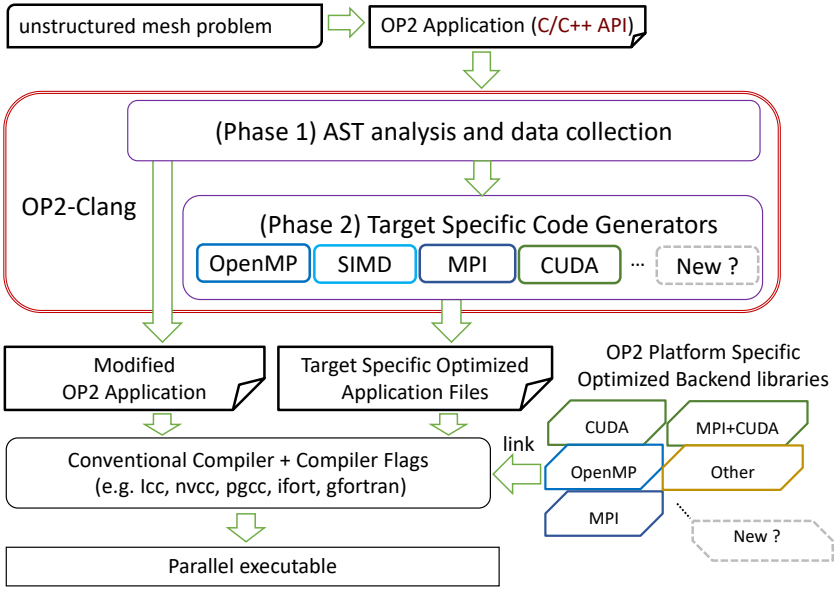


Figure 2: Az OP2-Clang magas szintű felépítése és elhelyezkedése az OP2 keretrendszerben

erű támogatást nyújt a kód refaktorálási feladataihoz a kód absztrakt szintaxisfájának (AST) meghatározott részeinek összevetésével és az összevetett csomópontok mögötti forráskód módosításával. Az OP2-ben a forráskódból forráskóddá alakítás teljes folyamatához a kódgenerátornak két lépésre van szüksége. Az első összegyűjti a generálandó párhuzamos ciklusokra vonatkozó adatokat, és az eredeti `op_par_loop` hívásokat a generált függvények hívásaival helyettesíti, a második pedig a célhardverre vonatkozó kód generálása, ahogyan az az 2 ábrán látható. Az első fázis elemzi a `op_par_loop` hívások összes argumentumát, hogy összegyűjtse az összes olyan információt, amely a párhuzamosítási vázban a hurok-specifikus kód kitöltéséhez szükséges. A második fázis kiválasztja a megfelelő vázszerkezetet, felépíti az AST-t, és egy sor refaktorálási lépést hajt végre, például a függvény szignatúra megváltoztatását, hogy létrehozza a végleges specializált ciklus implementációt.

Ez a megközelítés két előnnyel jár a hagyományos Python kódgenerátorral szemben. Az első az, hogy a kódgenerátor könnyen újra tudja használni a refaktorálási lépések részeit a cél architektúrák között, ami megkönnyíti az új célokkal való bővítést. A második az, hogy a fordítói infrastruktúra felhasználásával az új kódgenerátor a fordításkor kifinomultabb szemantikai ellenőrzéseket végezhet a generált kódon.

II. Tézis

Egy új, nagy teljesítményű, elosztott memóriájú, nyílt forráskódú könyvtárat terveztem kötegelt-tridiagonális egyenletrendszerek megoldására, amely nagy méretű heterogén szuperszámítógépeket céloz. Az algoritmusok lehetővé teszik közelítő és egzakt megoldások kiszámítását modern többmagos és sokmagos processzor architektúrákon alapuló nagyméretű számítógépes rendszereket célozva. A könyvtár zökkenőmentesen integrálódik a nagyméretű parciális differenciálegyenletek megoldásában általánosan használt váltakozó irányú implicit módszerekkel. Az elkészült implementációt a nyílt forrású Tridsolver könyvtár kiegészítéseként tettük elérhetővé.

A tézishez tartozó publikáció: [J2]

A legkorszerűbb elosztott memóriás algoritmusok tridiagonális rendszerekhez a rendszert alrendszerekre osztják, és egy kisebb független tridiagonális rendszert alkotnak, amely összeköti a részeket (redukált rendszer). Ezután általában ezt a redukált rendszert vagy egyetlen processzre gyűjtik, ott megoldják, majd a megoldást szétosztják a folyamatok között, vagy iteratív megoldó algoritmusok, például Jacobi-iterációk segítségével oldják meg. Az előbbi rosszul skálázódik az all-to-all kommunikációs minták miatt, az utóbbi pedig, bár csak pontok közötti kommunikációt használ, közelítő megoldásokat eredményez.

Az ADI-ban az együtthatókat minden egyes rácspontra úgy számítják ki, hogy azok megfeleljenek az alkalmazás mögöttes adatszerkezetének. Az MPI-csomópontokat minden dimenzió mentén definiáljuk, és az diagonálisok adatait egybefüggően tároljuk, vagy sorfolytonos (Z egybe-

függő, Y és X stride-olt), vagy, ami gyakoribb, oszlopfolytonos (X egybefüggő, Y és Z stride-olt) formátumban. Ez kihívást jelent az olyan algoritmusok számára, amelyek ezután egyszerre több tridiagonális rendszert oldanak meg; a különböző irányok különböző memóriaelrendezéseket használnak, amelyek viszont különböző optimalizációt igényelnek. Ennek megfelelően könyvtárunk támogatja mindhárom különböző memóriaelrendezést, ami 3 vagy magasabb dimenziós problémák esetén lehetséges.

A Thomas-PCR hibrid algoritmus kiterjesztésével elosztott memóriás környezetre, megterveztem a egy tridiagonális megoldó algoritmust, amely pontos megoldásokat ad a kötegelt tridiagonális problémákra, miközben megtartja a közelítő algoritmusok skálázási tulajdonságait. Az elosztott tridiagonális megoldó általános felépítése a következőképpen foglalható össze. Minden M méretű alrendszer egy külön MPI-csomóponthoz tartozik, amely a hibrid Thomas-PCR előre haladó fázisát hajtja végre. Ez egy redukált rendszert eredményez, amelynek két sora van MPI-csomópontonként. A redukált rendszer megoldását az elosztott PCR algoritmus segítségével hajtjuk végre. Ez az algoritmus csak egy az egyhez kommunikációt használ, ami a skálázhatóság szempontjából döntő fontosságú kritérium. Miután a redukált rendszert megoldotta, a hibrid Thomas-PCR visszahelyettesítést minden egyes MPI-csomóponton függetlenül hajtja végre.

Table 1: Az elosztott megoldó algoritmusok kommunikációs mintáinak összehasonlítása.

	Accuracy	Communication pattern	number of messages	message size
Allgather	exact	all-to-all	1	$3 \times 2 \times N_{proc} \times N_{sys}$
Jacobi	approximate	local point-to-point, all-to-all for error	$2 \times N_{iterations}$	N_{sys}
PCR	exact	point-to-point with increasing distance	$2 \times log_2(N_{proc})$	$3 \times N_{sys}$

A 1 táblázat a redukált rendszerre vonatkozó három fő megoldási stratégia összehasonlítását mutatja, valamint az ezek használatához szükséges kompromisszumokat, ahol N_{proc} az MPI-folyamatok számát, N_{sys} pedig a kötegméretet jelöli (független egyenletrendszerek száma).

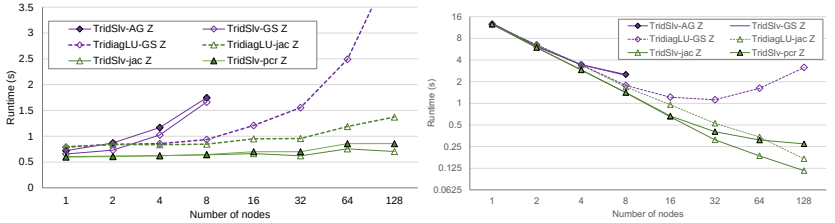
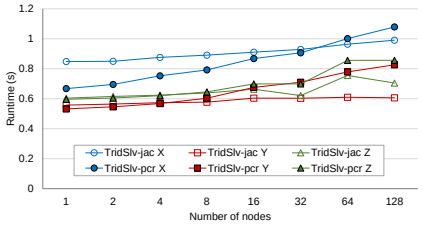


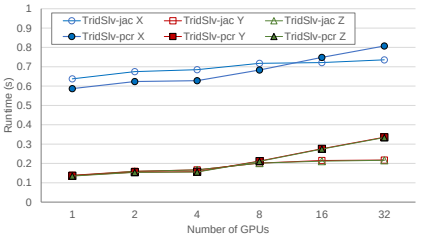
Figure 3: A tridiagLU és a Tridsolver könyvtár összehasonlítása. **Bal oldal:** Gyenge skálázás 512^3 rács pont per node, **Jobb oldal:** Erős skálázás, 8192 pont a megoldás dimenziójában, és 512 a másik két dimenzió mentén. AG - AllGather, GS - Gather-Scatter

Egy skálázható algoritmus létrehozásához kritikus létrehozásához, hogy elkerüljük a mindent mindenkinek kommunikációt, ami ahhoz vezetne, hogy az üzenetméret korrelálna a folyamatok számával. Az egy az egyhez kommunikációt használó algoritmusok esetében az üzenetek száma és a kommunikáló csomópontok közötti távolság befolyásolja a kommunikáció overheadjét. Láthatjuk, hogy a PCR algoritmus nagyobb üzeneteket használ az egymástól távolabbi csomópontok között, cserébe viszont nem igényel globális kommunikációt, és pontos megoldásokat produkál.

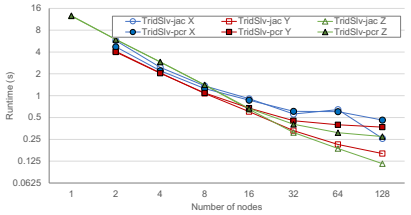
A hibrid algoritmus három lépéséből kettő triviálisan skálázódik. Az egyetlen rész, amely hozzájárul a skálázási tulajdonságokhoz, az a redukált rendszer elosztott megoldója. Az 3 ábrán látható, hogy a globális kommunikációs kollektívákra támaszkodó algoritmusok egy bizonyos pont után átveszik a futási időn a dominanciát. Az egy az egyhez kommunikáció esetén az üzenet mérete és a kommunikációhoz szükséges csomópontok távolsága a két tényező, amely meghatározza a többlet költségeket. Az 3 ábrán látható eredményekben a hibaszámításhoz globális reduce-hívások használata helyett a Jacobi-iterációk számának problémaspecifikus felső korlátját használtuk; ezért a Jacobi-iteráció fix számú kommunikációt használt, és minden csomópont csak a szomszédos csomópontokkal kommunikált, de ilyen felső korlátot nem lehet meghatározni az általános esetre. Másrészt a PCR-nek



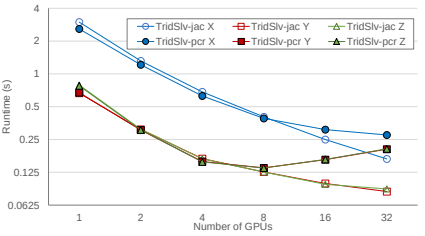
(a) Tridsolver gyenge skálázódás ARCHER2



(b) Tridsolver gyenge skálázódás Cirrus



(c) Tridsolver erős skálázódás ARCHER2



(d) Tridsolver gyenge skálázódás Cirrus

Figure 4: ARCHER2 skálázódás (MPI+OpenMP): (a),(c) Cirrus skálázódás (MPI+CUDA):(b),(d) - Minden gyenge skálázódás 512^3 rács pontot használ node-onként. Az erős skálázódás az ARCHER2-n 8192 pontot használ a megoldás dimenziójában, míg a Cirrus-on végzett mérések 2048 pontot. Mindkét esetben 512 pontot használtunk a másik két dimenzió mentén.

az ábra minden egyes adatpontjánál van egy további kommunikációs lépése, de a megoldás kiszámításához nincs szüksége problémaspecifikus heurisztikára. A kommunikáció növekvő költsége egyértelműen megmutatkozik a 16 MPI-csomópont utáni erős skálázás esetén, ahol a távoli üzenetek költsége (az ARCHER2-csomópontok helyi memóriájából való kilépés) uralja a teljes futási időt, miközben még mindig jelentősen jobb a globális kommunikációs minták alkalmazásához képest.

A 4 ábra a Tridsolver könyvtár skálázási teljesítményét mutatja egy 3D alkalmazás esetén mindhárom irány menti megoldóhívás esetén az ARCHER2 és a Cirrus rendszeren. CPU-kon a PCR változat 128 csomópontig 70% skálázási hatékonyságot ér el, míg GPU-kon az

egyetlen Cirrus csomóponton (4 GPU) kívüli kommunikáció költségei a lassabb összeköttetés miatt jelentősen nagyobbak, ami nagy hatással van a PCR megoldó skálázására.

III. Tézis

Létrehoztam egy összetett, absztrakt modellt a komplex stencil alkalmazások adjoint módú algoritmikus differenciálására, és integráltam az OPS doménspecifikus nyelvbe. Ez a modell lehetővé teszi az OPS számára, hogy a DSL által biztosított meta adatokat kihasználva többmagos CPU és GPU implementációkat generáljon az adjoint ciklusokhoz. A modell egy új leképezést használ az adjoint ciklusok párhuzamos implementációjához. Továbbá az elkészült kiegészítés lehetővé teszi, hogy az OPS a számítási lépéseket a ciklusok szintjén kövesse egy, az OPS DSL-re szabott, AD tape integrálásával egy egyszerűsített tárolási mechanizmust alkalmazva.

A tézishez tartozó publikációk: [J3, C2]

Altézis III.1. - Létrehoztam egy számítási modellt, amely az OPS absztrakción alapul, a strukturált hálós stencil alkalmazásokhoz, amely leírja a számítási mintákat, az adatokat és a control flow-t, és leírtam, hogy az adjoint módú algoritmikus differenciálás hogyan végezhető el ezzel a modellel.

Az érzékenységek hatékony kiszámítása számos területen kulcsfontosságú, és különösen nagy kihívást jelent a gradiens kiszámításának hatékony párhuzamos megvalósítása a adjoint módú AD-ban. A fordított módú AD két fő kihívása párhuzamos környezetben a versenyhelyzetek a hozzáférési minták megfordulása mentén és a control flow futásidejű követése. Az OPS API a ciklusok és a ciklusokon belüli adathozzáférések leírását használja a hatékony párhuzamos megvalósítások létrehozásához. Ezt a leírást felhasználva létrehoztam egy modellt, amely leírja a hozzáférési mintákat, leírja a potenciális versenyhelyzeteket a ciklusok adjointját végrehajtó ciklusokban, lehetővé téve a

párhuzamos implementációk generálását. A futásidőben regisztrált ciklus vagy kernel láncra építve ez a modell lehetővé teszi az OPS számára a deriváltak kiszámítását adjoint módú AD segítségével.

Altézis III.2. - Megterveztem és megvalósítottam a magas szintű modell leképezését optimalizált, alacsony szintű párhuzamos számítási kernelekre, amelyek mind a többmagos CPU-kat, mind a sokmagos GPU-kat támogatják, architektúra-specifikus optimalizációkkal.

Egy OPS ciklusban minden adathalmazhoz egy stencilen keresztül férünk hozzá, amelyhez egy hozzáférési módot rendelünk: olvasás, írás vagy inkrementálás. Az OPS-ben minden ciklus csak gather stencilt használhat, ami azt jelenti, hogy az olvasás tetszőleges számú ponttal rendelkező stencilen jelenhet meg, de az inkrementálás és az írás stencilet egyetlen ponthoz való hozzáféréssel kell rendelkeznie, nulla eltolással, azaz írás csak a stencil középpontján hajtható végre. Reverse módban AD az adjoint ciklusokban az adatáramlás megfordul a gather stencilekben, és a deriváltakon scatter stencileket kapunk. A megfordítás miatt az adatkészleteken lévő írási és inkremens stencilek az derivált adatokon egypontos olvasási stencilekké, az olvasási stencilek pedig többpontos inkremens stencilekké válnak. A fentieknek két következménye van: először is, a primer ciklusok versenymentes, a párhuzamosítás triviális, másodszor, az adjoint ciklusokban versenyhelyzetek lesznek a deriváltakon. Ezeket az versenyhelyzeteket azonban egyértelműen meghatározzák a primer ciklusok olvasási stenciljei. Megterveztem és megvalósítottam egy olyan végrehajtási mintát, amely elkerüli a versenyhelyzeteket a CPU-kon, amely a ciklust két ütemben hajtja végre, a kettő közötti szinkronizálással. Ez a megközelítés elkerüli az atomikus műveletek költségeit. Mivel GPU-kon az atomikus műveletek költsége alacsonyabb; ezért az adjoitn kernelek atomikus műveleteket használnak a deriváltakon.

A teljesítmény szempontjából egy másik fontos hozzáférési minta az alacsonyabb dimenziójú adatok (bizonyos dimenzióban változatlan

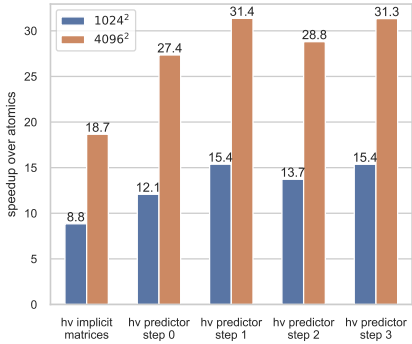


Figure 5: Az elérési mintát figyelembe vevő redukciók alkalmazása által a csak atomikusokon alapuló implementációkhoz képes elért gyorsulás 1024^2 és 4096^2 méretű rácsokon a CDE alkalmazáson.

adatok) olvasása. Ezek a hozzáférések nagy mennyiségű írást eredményeznek ugyanazon derivált értékekre az adjoint propagalás során. A CUDA adjoint kernelekben egy speciális kódgenerálási utat vezettem be az alacsony dimenziójú adathalmazok számára, amely csak a szükséges dimenziókban történő redukciókat használja. Az ábra 5 mutatja a sebességnövekedést egy NVidia A100-as GPU-n az optimalizálás használatával.

Altézis III.3. - Kiterjesztettem az OPS absztrakciót a modell és a leképezés integrálására, automatikusan megszervezve a primer és az adjoint számításokat, beleértve a differenciálás memóriaterhelésének ellenőrzését, és lehetővé téve a derivált számítás hatékony párhuzamosítását. Ennek a kiterjesztésnek a hasznosíthatóságát és teljesítményét ipari szempontból reprezentatív alkalmazásokon mutattam be.

A hagyományos operátor túlterhelés alapú adjoint módú AD eszközök csak legfeljebb a kifejezések szintjén tudják követni a programokban a control flowt, ami nagy memória igényt eredményez. Az OPS absztrakcióra építve bevezettem egy tape adatszerkezetet, amely nyomon követi a párhuzamos ciklusok leírásait, és tárolja a ciklusok felülírt

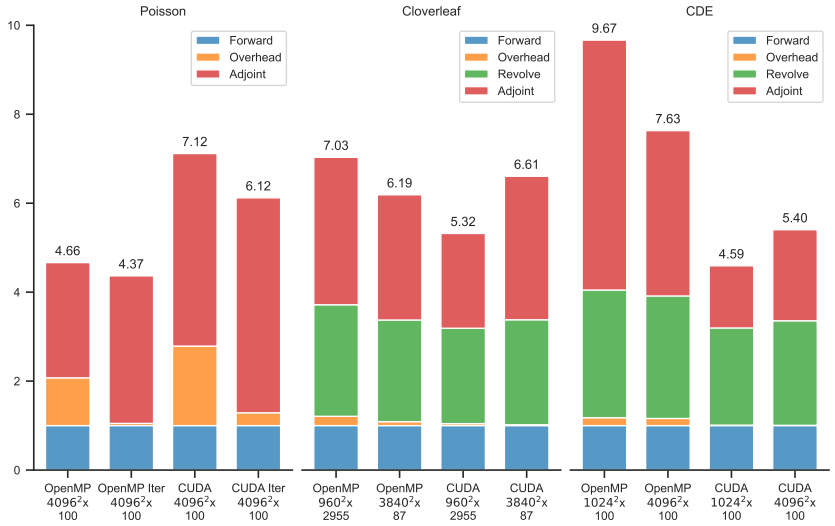


Figure 6: Az AD többletköltsége a referencia alkalmazásokon, összehasonlítva a passzív, eredeti alkalmazás egyszeri kiértékelésével. Az értékek azt mutatják, hogy a gradiens kiértékelése az eredeti alkalmazás kiértékelésének N-szeresét veszi igénybe. A **Overhead** értékek a DAG összegyűjtésének, a köztes állapotok mentésének és a Revolve ellenőrzőpontok tárolásának többletköltségeit mutatják a primer menet során, a **Revolve** értékek az alkalmazások azon szakaszainak újrajátszásával töltött többletidőt jelentik, amelyek az adjoint ciklusok állapotainak visszaállítására szolgálnak, a **Adjoint** rész pedig a tényleges adjoint ciklusokban töltött időt mutatja.

adatait. A magas szintű tape drasztikusan csökkenti a control flow információk tárolásának memória igényét. Azonban egy alkalmazásban az összes felülírt adat tárolása nagyszámú iterációhoz hatalmas tape-t eredményezne. E probléma megoldására a tape-t kibővítettem a Revolve[12] ellenőrzőpontozási stratégiával, amely a felhasználó számára finomhangolt ellenőrzést biztosít az adjoint számítás memóriahasználata felett, a ciklusok újbóli végrehajtását használva a köztes állapotok újraszámításához.

A derivált számítás teljesítményét három ipari szempontból reprezentatív alkalmazáson értékeltem. Az ábra 6 a teljes adjoint számítás relatív futási idejét mutatja (beleértve az eredeti függvény kiértékelését is) az eredeti alkalmazás egyetlen kiértékeléséhez képest.

4 Potenciális alkalmazási területek

Az OP2-Clang eszközzel kapcsolatos munkám közvetlenül alkalmazható volt az OP2 DSL forráskódból forráskódra képezési rétegeként. Egy fordító alapú eszköz használata növelhetné a kódgenerálás robusztusságát és diagnosztikai képességeit, és egyúttal megkönnyítené az ipari fordító rendszerekbe való integrációt.

A kötegelt-tridiagonális megoldókönyvtárakkal összefüggésben végzett eredmények az ADI-módszert alkalmazó nagyméretű tudományos alkalmazásokban is felhasználhatók. Ez a kutatás részben az Egyesült Királyság ExCALIBUR projektjének részeként valósult meg, amelynek célja a nagy teljesítményű szimulációs szoftverek következő generációjának kifejlesztése. A projekt részeként folyamatban van a könyvtár xCompact3D könyvtárban való használatának megvitatása [13]. Ez a könyvtár egy ipari erősségű könyvtár a turbulens áramlások szimulációjára, és olyan szimulációkhoz használják, mint például egy teljes szélerőmű park körüli légáramlás kiszámítása.

Az adjoint módú algoritmikus differenciálást gyakran használják a számítási áramlástanban és a pénzügyi számításokban. Eredményeink

azt mutatják, hogy a domén-specifikus nyelvek vagy absztrakciók, különösen az OPS, drasztikusan csökkenthetik a memóriaterhelést és javíthatják a gradiensek számításának futási idejét az általános célú eszközökhöz képest. Továbbá, a magas szintű OPS alkalmazáskód támogatja a CPU-kat és a GPU-kat is, ami az OPS-t az egyik első, teljesítményben hordozható adjoint módú AD-könyvtárrá teszi.

5 Köszönetnyílvánítás

Mindenekelőtt szeretnék köszönetet mondani témavezetőmnek, Dr. Reguly Istvánnak, amiért bevezetett a nagy teljesítményű számítástechnika világába, számtalan lehetőséget biztosított számomra, valamint hatalmas támogatásáért, útmutatásáért és türelméért, amely végigvezetett ezeken az éveken. Pozitív és támogató hozzáállása rengeteget segített nekem az elmúlt évek során. Mélyen hálás vagyok Dr. Gihan Mudaligenak, amiért a Warwickban töltött időm alatt befogadott a kutatócsoportjába, valamint a segítségéért és a rálátásáért az együttműködésünk során. Hálás vagyok továbbá Jacques du Toitnak, aki nagylelkűen rendelkezésre bocsátotta tudását és szakértelmét. Szeretnék köszönetet mondani Prof. Uwe Neumann professzornak és Dr. Johannes Lotznak a lehetőségért, hogy tanulhattam tőlük az RWTH Aachen Egyetemen töltött időm alatt.

Szeretnék köszönetet mondani minden barátomnak és kollégámnak, amiért nevetéssel és örömmel töltötték meg ezt az elmúlt néhány évet. Szeretnék köszönetet mondani Sulyok András Attilának, Siklósi Bálintnak, Horváth-Keömley Bencének, Rudner Tamásnak, Vághy Mihálynak és még sokaknak a számtalan közös gondolatébresztő beszélgetésért, amelyek állandó inspirációt és fejlődést jelentettek.

Hálás vagyok a Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai Karának, különösen Prof. Szederkényi Gábornak és Prof. Szolgay Péter professzornak a doktori programban való részvétel lehetőségéért és a doktori képzés során nyújtott támogatásukért. Külön

köszönettel tartozom Dr. Tivadarné Vida Tivadarnénak a doktori tanulmányaim során nyújtott szíves segítségéért.

Végül, szeretném kifejezni végtelen hálámat a családomnak a határtalan támogatásért, és különösen Balogh-Lantos Zsófiának, hogy elviselt és átsegített a nehéz időszakokon.

A szerző publikációi

Folyóirat publikációk

[J1] A. A. Sulyok, **G. D. Balogh**, I. Z. Reguly, and G. R. Mudalige, “Locality optimized unstructured mesh algorithms on GPUs", *Journal of Parallel and Distributed Computing*, vol. 134, pp. 50–64, 2019 doi: 10.1016/j.jpdc.2019.07.011.

[J2] G. D. Balogh, T. S. Flynn, S. Laizet, G. R. Mudalige and I. Z. Reguly, "Scalable Many-Core Algorithms for Tridiagonal Solvers", in *Computing in Science & Engineering*, vol. 24, no. 1, pp. 26-35, 1 Jan.-Feb. 2022, doi: 10.1109/MCSE.2021.3130544.

[J3] G. D. Balogh, J. Lotz, J. Du Toit, U. Naumann, and I. Reguly, "Performance portable adjoints for structured mesh applications with OPS", in *ACM Trans. Math. Softw.*, **under submission**.

Konferencia publikációk

[C1] G. D. Balogh and I. Reguly, "Automatic Parallelisation of Sturctured Mesh Computations with SYCL," *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, Portland, OR, USA, 2021, pp. 821-822, doi: 10.1109/Cluster48925.2021.00083.

[C2] G. D. Balogh and I. Z. Reguly, "Automatic parallel implementations of adjoint codes for structured mesh applications," *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet*

Computing (CCGRID), Melbourne, VIC, Australia, 2020, pp. 908-911, doi: 10.1109/CCGrid49817.2020.00019.

[C3] G. D. Balogh, G. R. Mudalige, I. Z. Reguly, S. F. Antao and C. Bertolli, "OP2-Clang: A Source-to-Source Translator Using Clang/LLVM LibTooling," *2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, Dallas, TX, USA, 2018, pp. 59-70, doi: 10.1109/LLVM-HPC.2018.8639205.

[C4] G. D. Balogh, I. Z. Reguly and G. R. Mudalige, "Comparison of Parallelisation Approaches, Languages, and Compilers for Unstructured Mesh Algorithms on GPUs", *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, vol. 10724, pp. 22–43, 2017, doi: 10.1007/978-3-319-72971-8_2

Referenciák

[1] S. J. Pennycook, J. D. Sewall, D. W. Jacobsen, T. Deakin, and S. McIntosh-Smith, "Navigating performance, portability, and productivity," *Computing in Science & Engineering*, vol. 23, no. 5, pp. 28–38, 2021.

[2] I. Z. Reguly, G. R. Mudalige, and M. B. Giles, "Loop tiling in large-scale stencil codes at run-time with ops," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 873–886, 2018.

[3] G. Mudalige, M. Giles, I. Reguly, C. Bertolli, and P. Kelly, "Op2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures," in *2012 Innovative Parallel Computing (InPar)*, pp. 1–12, 2012.

[4] D. W. Peaceman and H. H. Rachford, Jr, "The numerical solution of parabolic and elliptic differential equations," *Journal of the Society for industrial and Applied Mathematics*, vol. 3, no. 1, pp. 28–41, 1955.

[5] D. Dutykh, R. Poncet, and F. Dias, "The volna code for the numerical modeling of tsunami waves: Generation, propagation and inundation," *European Journal of Mechanics - B/Fluids*, vol. 30, no. 6, pp. 598–615, 2011. Special Issue: Nearshore Hydrodynamics.

[6] E. Laszlo, M. Giles, and J. Appleyard, "Manycore algorithms for batch scalar and block tridiagonal solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 4, pp. 1–36, 2016.

[7] W. Gander and G. H. Golub, "Cyclic reduction—history and applications," *Scientific computing (Hong Kong, 1997)*, vol. 7385, pp. 73–86, 1997.

[8] D. Ghosh, E. M. Constantinescu, and J. Brown, "Efficient implementation of nonlinear compact schemes on massively parallel platforms," *SIAM Journal on Scientific Computing*, vol. 37, no. 3, pp. C354–C383, 2015.

[9] B. Christianson, "Reverse accumulation and attractive fixed points," *Optimization Methods and Software*, vol. 3, no. 4, pp. 311–326, 1994.

[10] A. Mallinson, D. A. Beckingsale, W. Gaudin, J. Herdman, J. Levesque, and S. A. Jarvis, "Cloverleaf: Preparing hydrodynamics codes for exascale," *The Cray User Group*, vol. 2013, 2013.

[11] M. Wyns and J. Du Toit, "A finite volume–alternating direction implicit approach for the calibration of stochastic local volatility models," *International Journal of Computer Mathematics*, vol. 94, no. 11, pp. 2239–2267, 2017.

[12] A. Griewank and A. Walther, "Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation," *ACM Transactions on Mathematical Software (TOMS)*, vol. 26, no. 1, pp. 19–45, 2000.

[13] P. Bartholomew, G. Deskos, R. A. Frantz, F. N. Schuch, E. Lamballais, and S. Laizet, "Xcompact3d: An open-source framework

for solving turbulence problems on a cartesian mesh," *SoftwareX*, vol. 12, p. 100550, 2020.