COSMOS

Framework for Combining Simulation and Optimization Software

Rezumat Teză de Doctorat

Cosmin-Gabriel Samoilă

Departamentul de Calculatoare
Universitatea Națională de Știință și Tehnologie POLITEHNICA Bucureşti

Preşedinte comisie doctorat: CS I Dr. **Călin Alexandru UR**
Universitatea Naţională de Ştiinţă şi Tehnologie POLITEHNICA Bucureşti

Conducător de doctorat: Prof. Dr. Ing. **Emil-Ioan SLUȘANSCHI**
Universitatea Naţională de Ştiinţă şi Tehnologie POLITEHNICA Bucureşti

Referenţi oficiali:
Prof. Dr. Ing. **Vasile MANTA**
Universitatea Tehnică "Gheorghe Asachi" din Iaşi

Prof. Dr. Ing. **Adrian FLOREA**
Universitatea "Lucian Blaga" din Sibiu

CS II. Dr **Viorel CHIHAIA**
Institutul de Chimie Fizică "Ilie Murgulescu" din Bucureşti

SDIALA

Cuprins

Capitolul 1

Introducere

Dacă în urmă cu zece ani aveam servere cu performanță maximă în virgulă mobilă cu dublă precizie de cel mult 10 TFLOPS, acum atingem limite cu ordine de mărime mai mari atât în ceea ce privește puterea brută de calcul, cât și eficiența energetică. Un domeniu în care am observat progrese majore este complexitatea hardware și principala dificultate cu care ne confruntăm în acest moment este dezvoltarea de software care poate beneficia de aceste progrese, maximizând în același timp performanța.
Un exemplu notabil care ne arată că nu numai software-ul trebuie să țină pasul cu hardware-ul este implementarea unităților aritmetice INT8 în acceleratoare. Această cerință a venit din progrsul framework-urilor de Invaţare Automată care utilizează in principal calculele INT8. Proiectanții de sisteme hardware au fost astfel nevoiți să faca design si sa implementeze soluții eficiente din punct de vedere energetic. După cum am menționat, complexitatea hardware și software a crescut la niveluri în care scrierea aplicațiilor de la zero necesită multe cunoștințe, atat arhitecturale cat și de programare. Luând în considerare costurile de dezvoltare și testare a software-ului, în multe scenarii este mai bine să utilizam sau să extindem codul deschis sau proprietar existent. Deoarece o mare parte din acest software de simulare este utilizat împreună cu pachetele de optimizare, integrarea acestora nu este o sarcină usoara in cele mai multe cazuri.
Având în vedere această problemă, am dezvoltat o soluție open source care are scopul de a face posibilă această cuplare cât mai usoare, fără a fi necesar să aveți cunostinţe avansate de programare.

1.1 Framework COSMOS

COSMOS este un framework care combină software-ul de simulare cu pachete de optimizare. Scopul principal este de a oferi o experiență cat mai usoara in integrarea aplicațiilor software

existente fără a fi nevoie să modificați codul sursă original. Deși acest lucru s-ar putea să nu fie întotdeauna posibil, ne vom strădui să explicăm în această teză cum se poate face folosind COSMOS și cod minimal scris in Python.

O altă funcție pe care am dezvoltat-o este programarea optimă a sarcinilor pe arhitecturi eterogene. După cum vom descrie în continuare, acest lucru se va face luând în considerare caracteristicile sarcinilor și resursele disponibile pe baza unor strategii definite de utilizator. Având în vedere structura deschisă a framework-ului și principiile de proiectare pe care le-am folosit, implementarea de noi algoritmi de planificare a spațiului utilizatorului este trivială. Una dintre ideile de bază pe care ne-am construit framework-ul este că trebuie să oferim o metodă simplă și simplă de a integra pachete de optimizare deja disponibile pentru fiecare clasă de probleme. De asemenea, oferim soluții utilizatorilor pentru a scrie pachete de optimizare în limbajele de programare dorite, fara a ne limita la a utiliza doar Python.

Motivată de o evoluție continuă a constrângerilor și cerințelor în software-ul de simulare, această lucrare propune o soluție ușor de utilizat pentru integrarea software-ului de simulare și optimizare în mediile HPC și își propune să:

- prezinte contextul general și problemele comune întâlnite de pachetele software de simulare și optimizare în mediile HPC
- descrie arhitectura framework-ului COSMOS, beneficiile cat si limitarile sale
- contributii in domeniile software-ului de simulare (de exemplu, CORSIKA, LAMMPS, GROMACS) și optimizare pe sisteme HPC care utilizează arhitecturi de calcul diferite (de exemplu, x86-64 și arm64)

Începem această teză analizând CORSIKA, o simulare de din domeniul astrofizicii, scrisă în Fortran și C++. La începutul proiectului nostru, singurul nostru obiectiv a fost să reducem timpul de simulare pentru un scenariu real, scenariu al cărui timp de execuție era mai mare de o săptămână. Prin profilarea executiei simularii, am determinat ce secțiunile din cod pot fi accelerate prin paralelizare. Cu toate acestea, analizând teoretic cel mai bun scenariu – conform Legii lui Amhdahl – ar putea fi atinsă doar o accelerare estimată de 2.2x . Datorită acestei limitări teoretice a accelerației am decis să abordăm problema dintr-un alt unghi. Urmatorul pas in analiza noastra a constat in studierea modelelor de bază implicate în simulare, a parametrilor de intrare și a fișierele de ieșire. Am ajuns la concluzia că execuția poate fi împărțită în simulări mai mici prin și generarea automata a sub-taskurilor independente. De asemenea, am dezvoltat un pachet pentru CORSIKA integrat in COSMOS pentru a face schedule automat al instanțelor simulării. Datorită naturii modelului de bază, timpul de simulare este proporțional cu numărul de parametri ai fișierului de intrare. Astfel, am reușit să reducem timpul total de simulare proporțional cu nucleele

de procesor disponibile. Pentru simularile cu un fisier de intrare suficient de mare pentru a reprezenta un scenariu real, numărul de procese intensive de calcul generate de scripturile noastre a ajuns la peste o sută.

Având în vedere aceste cerințe, am decis să construim COSMOS ca un framework universal care poate oferi o interfață pentru combinarea diverselor pachete software de simulare și optimizare, precum și un wrapper pentru trimiterea sarcinilor în sisteme de HPC.

1.2 Context si lucrări conexe

Framework-ul COSMOS își propune să integreze software-ul de simulare cu metode de optimizare. Pachetele de software de simulare sunt de obicei modele matematice care descriu fenomene din lumea reală (cum ar fi reacții chimice, circuite electronice, procese biologice, măsurători fizice etc.) traduse în programe de calculator. Deoarece acest tip de simulări sunt, de obicei, intensive din punct de vedere al calculului, relaxarea sau înăsprirea condițiilor inițiale și a parametrilor de simulare înainte de orice experiment numeric este crucială. Majoritatea modelelor computaționale de calcul științifice necesită o perioadă semnificativă de timp pentru a fi dezvoltate și sunt scrise în C, C++ sau Fortran. În consecință, înțelegerea lor și reglarea parametrilor lor nu este o sarcină trivială. Din punct de vedere al optimizării, COSMOS poate fi configurat să utilizeze fie software-ul de optimizare numerică existent, fie să suporte adăugarea altor algoritmi din literatură. Această abordare ne oferă un avantaj semnificativ, deoarece putem fi flexibili în alegerea dintre mai multe metode de optimizare, precum și în posibilitatea de a regla parametrii de intrare.

O abordare similară, oferind un mediu software interactiv care combină codul numeric de simulare cu pachete software de optimizare, este propusă de Rasch și Bücker în EFCOSS [2]. EFCOSS este axat pe design experimental optim și este prezentat ca o extensie a unei implementări anterioare [6] care s-a concentrat în principal pe ajustarea parametrilor. Principalul dezavantaj al acestei abordări pe care am încercat să-l rezolvăm în implementarea noastră este capacitatea de a specifica dependențe între sarcinile de simulare și optimizare.

EASY-FIT [42] ofera un sistem software interactiv pentru gasirea solutiilor si estimarea parametrilor pentru modele matematice, aparținând unor categorii precum ecuații algebrice diferențiale, transformări Laplace, ecuații diferențiale obișnuite, sisteme în stare staționară sau sisteme de ecuații diferențiale parțiale unidimensionale dependente de timp. Acest software oferă patru rutine de optimizare și un număr de rezolvatori de ecuații diferențiale. În COSMOS, ne propunem să suportam rutine de optimizare si pachete externe care sa vina in ajutorul cercetatorilor pentru analiza datelor si rezultatelor simularilor.

O abordare similară a COSMOS este propusă de Network Enabled Optimization System [24] care oferă posibilitatea de a rezolva probleme de optimizare pe un server diferit de cel folosit de client. Această soluție oferă o flexibilitate ridicată și joacă același rol pe care îl are modulul Optimizer în cadrul COSMOS. Dacă luăm în considerare tipul de probleme pe care încercăm să le rezolvăm cu software-ul nostru, NEOS poate fi utilizat fie ca înlocuitor, fie împreună cu modulul Optimizer.

În cele din urmă, pachetele software care oferă suport pentru modelarea sistemelor de calcul la scară largă și evaluarea soluțiilor algoritmice și de sistem sunt oferite de sistemele SimGrid [4] și CloudSim [46]. În COSMOS, am proiectat un modul Broker ca o versiune simplificată a acestor pachete, permițând utilizatorilor să descrie sarcini și care pot folosi metode de programare statică personalizate.

Capitolul 2

Proiectarea si Implementarea Framework-ului

În acest capitol vă prezentăm o descriere detaliată a *Framework-ului COSMOS*. Această aplicație este disponibilă pentru sub licenta open source și se poate contribui cu imbunatatiri pe GitHub [1].

Frmework-ul este destinat optimizării aplicațiilor și trimiterii task-urilor pentru a fi executate remote într-un cluster de calculatoare. Scopul principal al acestui proiect este de a oferi o interfață stabilă și ușor de utilizat oricărui program de optimizare, o abstractizare a arhitecturilor hardware și o metodă automată de reprogramare a sarcinilor pe baza rezultatului generat.

În secțiunile următoare sunt prezentate și detaliate elementele care compun aplicația. Framework-ul nostru este scris în Python și este compus din module ce pot funcționa împreună sau independent - Controller, Broker și Optimizer, așa cum se arată în Figura 2.1.

Unul dintre principalele avantaje ale acestui cadru este că modulele pot fi utilizate independent și orice utilizator își poate scrie propriul modul care înlocuiește o anumită implementare implicită.

2.1 Setari

În această secțiune vom prezenta setările hardware și software ale sistemelor HPC utilizate în testarea cadrului COSMOS. Vom prezenta fiecare microarhitectură și vom motiva alegerile noastre hardware. Având în vedere faptul că rulăm cod intensiv de calcul, cele două valori principale de performanță care urmau să fie îmbunătățite au fost operațiunile cu virgulă

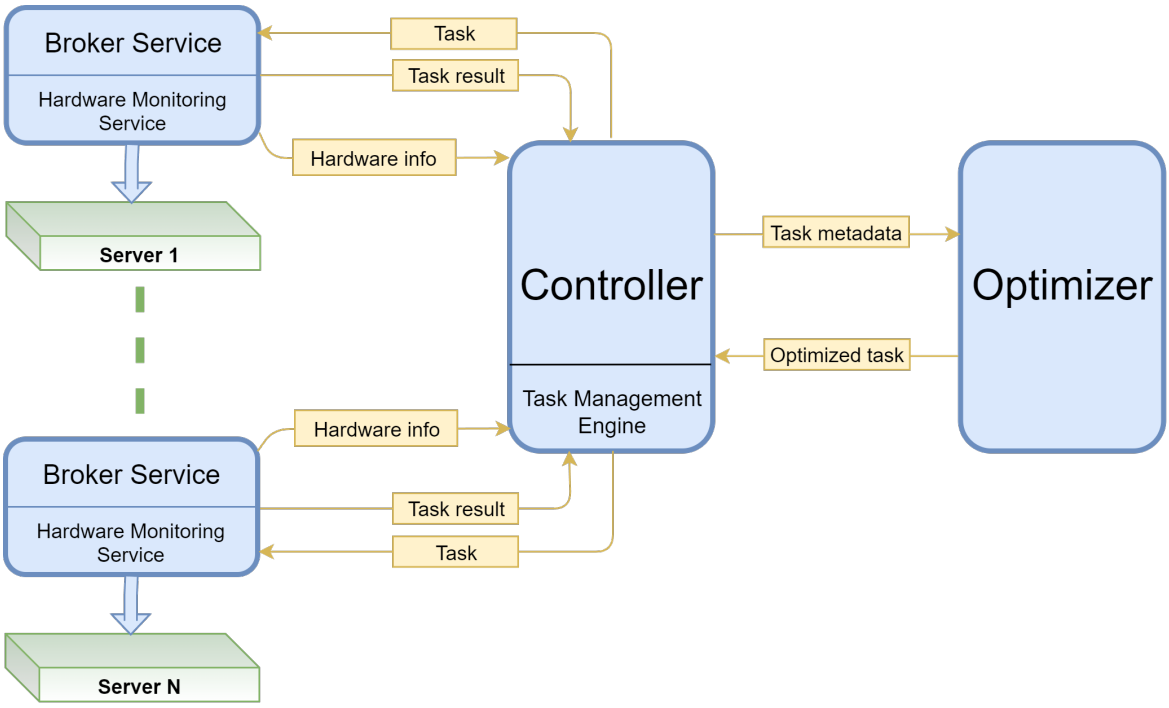

Fig. 2.1 COSMOS Framework Design

mobilă pe secundă (FLOPS) și eficiența energetică - măsurată folosind instrumente standard de profilare.

Pentru a maximiza metrica FLOPS, a trebuit să alegem dintr-o varietate de mașini cu procesoare server Intel [8]: și anume Nehalem, Ivy-Bridge și Haswell. Datorită faptului că, în general, avem de programat multe sarcini independente cu un singur thread, am optat pentru nodurile cu microarhitectura Ivy Bridge care ne-ar putea oferi cel mai mare număr de nuclee disponibile cu o penalizare minima de performanță în comparație cu microarhitecturi mai noi.

O problemă care merită menționată este tehnologia hyper-threading, care este activată implicit pe toate mașinile noastre de testare Intel. În timp ce existența mai multor fire de execuție logice pentru fiecare nucleu fizic este utilă pentru sarcinile de zi cu zi, în aplicațiile de înaltă performanță, executia a două procese pe același nucleu fizic poate da rezultate mai proaste decât o execuție secventială, prin urmare, trebuie să luăm în considerare acest aspect la implementarea algoritmilor de planificare a task-urilor. Pentru experimentele noastre, am folosit servere x86 cu procesoare Intel și plăci Raspberry PI 4B cu specificațiile hardware prezentate în Tabel 2.1

Configuraţie Software Sistemul de operare instalat pe serverele x86 (Intel) folosite de noi este CentOS 7.2.1511, cu un nucleu Linux 3.10.0. Nodurile bazate pe ARM rulează un sistem

Procesor	Intel Xeon E5-2670 v2	ARM Cortex-A72
Frecvenţa	2.5 GHz	1.7 GHz
Noduri	1	2
Socheturi	2	1
Nuclee	20	4
Fire de executie	40	4
Microarhitectura	Ivy Bridge	ARMv8-A
Memororie	128 GB	8 GB
Tip stocare	HDD	SSD via USB3.0 bus

Table 2.1 Specificatii Hardware

de operare Raspbian Stretch cu un nucleu Linux 4.14. Pentru a minimiza diferențele dintre cele două platforme de calcul am decis să folosim aceleași versiuni de pachete software, cum ar fi versiunea 4.9 a GNU Compiler Collection și versiunea 2.7 a runtime-ului Python.

2.2 COSMOS

Modulul Controller

Acest modul a fost conceput pentru a avea următorul set de capabilități: analizarea si validarea fișierelor de configurare, creearea instanțelor de optimizator și broker, trimiterea de comenzi către optimizator și broker și luați decizii bazate pe rezultatul fiecărei task. Diagrama de interacțiuni cu controlerul este prezentată în Figura 2.2.

Pe lângă funcționalitatea de bază, acest modul poate fi configurat să folosească informațiile de încărcare a mașinii trimise de serviciile Broker pentru a programa mai bine sarcinile. Dacă informațiile de încărcare a mașinii nu pot fi preluate în timp real, sarcinile vor fi programate și deciziile vor fi luate numai pe fișierele inițiale de descriere a mașinii (scrise de utilizator). O cerință netriviala este replicarea mediului virtual Python pe toate mașinile folosite pentru executia task-urilor. Acest mediu virtual trebuie să îndeplinească toate dependențele necesare care sunt enumerate în documentația COSMOS. Pentru a utiliza COSMOS, trebuie să creați JSON-urile de configurare pentru fiecare modul și fișierul de comenzi.

Optimizer

Acest modul joacă un rol crucial în designul curent, fiind instanța care oferă indicații Controlerului despre următorii pași ce trebuie executați pentru fiecare task.

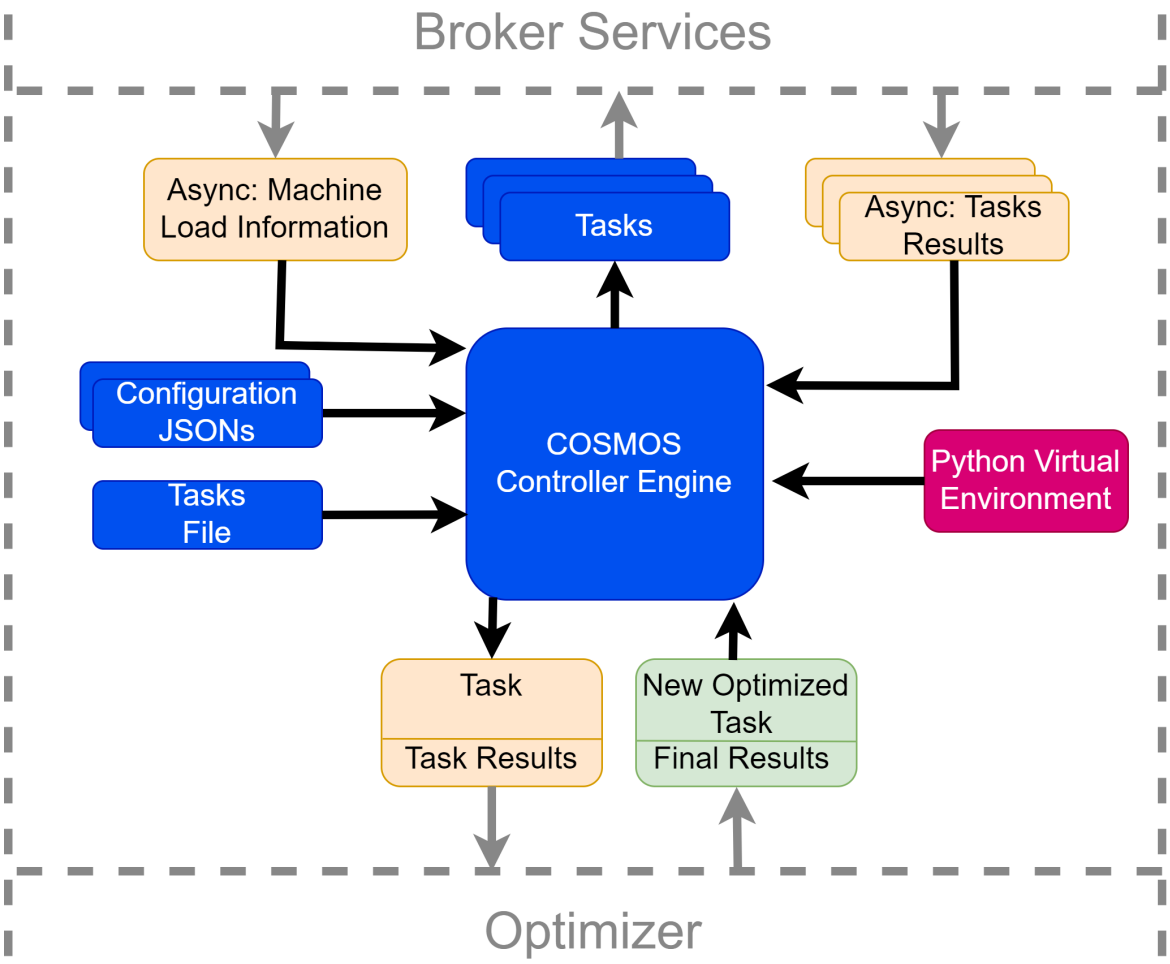

Fig. 2.2 Controller Interactions Diagram

Avand ca principal scop flexibilitatea in implementare, modulul de optimizare preia un JSON de intrare și execută metodele de optimizare aferente fiecarui task. Aceste metode de optimizare bazate pe sarcini pot fi, fie implementate de utilizator direct în clasa Optimizer, fie le pot folosi ca un wrapper pentru o implementare a unui algoritm de optimizare deja existent, cum ar fi: minimizarea neconstrânsă și constrânsă, minimizarea celor mai mici pătrate, algoritmi de ajustare a curbei, rezolvatori de sisteme de ecuații multivariate. Diagrama de interacțiuni pentru modulul Optimizer poate fi văzută în Figura 2.3. După cum se poate observa, pe lângă mediul virtual Python, avem și alte grele care trebuie rezolvate pentru acest modul. Dacă bucla de optimizare conține vreo metodă dintr-un pachet de optimizare extern (prin extern înțelegem orice nu este implementat în framework sau în aplicația de simulare), utilizatorul trebuie să rezolve toate dependențele. Aceste dependențe pot conține, dar nu se limitează la, aplicații de optimizare, biblioteci de sistem și stiva de software. De exemplu,

dacă ciclul de optimizare conține o metodă din biblioteca cuSolver, stiva CUDA va trebui să fie instalată și pe mașina pe care rulează modulul de optimizare.

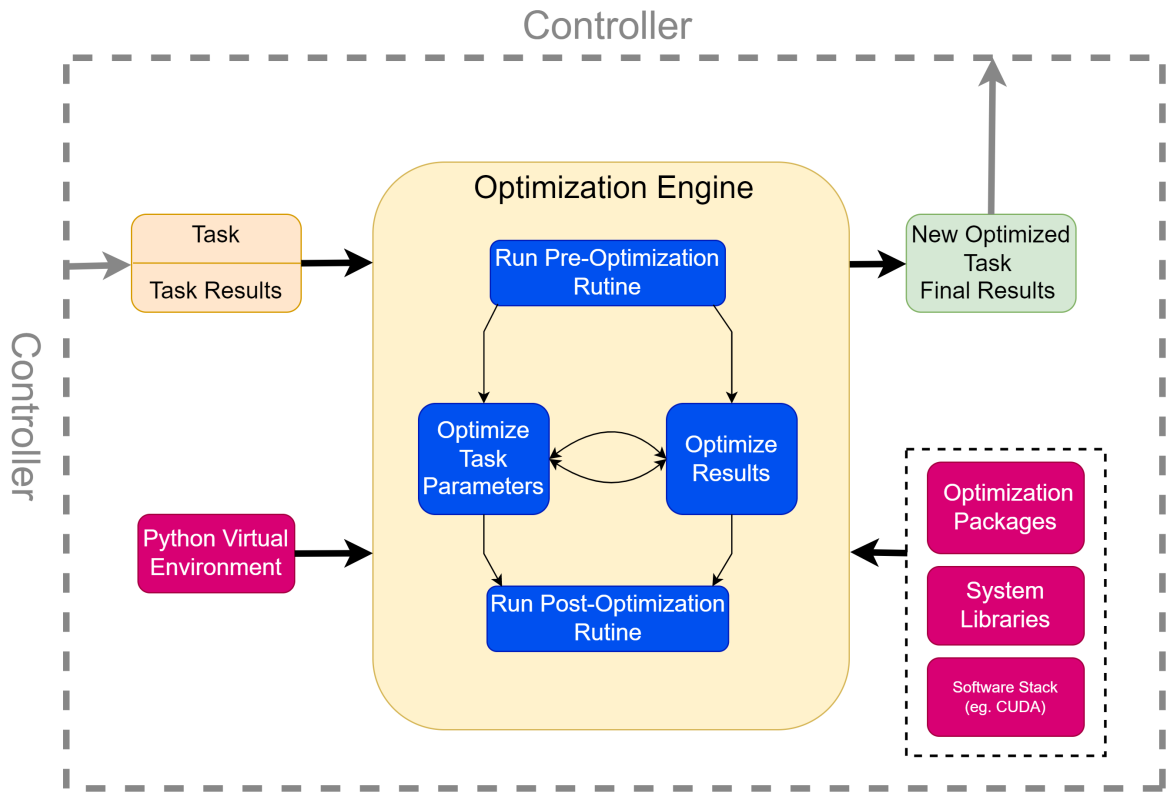

Fig. 2.3 Optimizer Interactions Diagram

Broker

Acest modul a fost creat ca un wrapper pentru un manager de resurse, cum ar fi *Terascale Open-source Resource and Queue Manager (TORQUE)* [44] . Pe lângă operațiunile obișnuite, cum ar fi trimiterea sau terminarea unui job, o instanță de broker are câteva caracteristici suplimentare, cum ar fi:

- specifică dependențele sarcinilor și creează mediul de lucru pentru fiecare task
- evaluează încărcarea actuală a mașinilor și programeaza task-urilor pe baza descrierii facute de utilizator in fisierele de configurare

Principalele funcții pe care Brokerii trebuie să le îndeplinească și interacțiunile din interiorul modulelor Broker sunt prezentate în Figura 2.4. Similar cu Controller și Optimizer, Brokerul trebuie să îndeplinească cerințele de dependențe Python și biblioteci de sistem / stiva software. Dacă simularea nu este implementată ca parte a framework-ului sau nu este

furnizat prin mediul virtual Python, utilizatorul trebuie să instaleze toate pachetele necesare fiecarui task pentru a rula pe o anumită mașină.

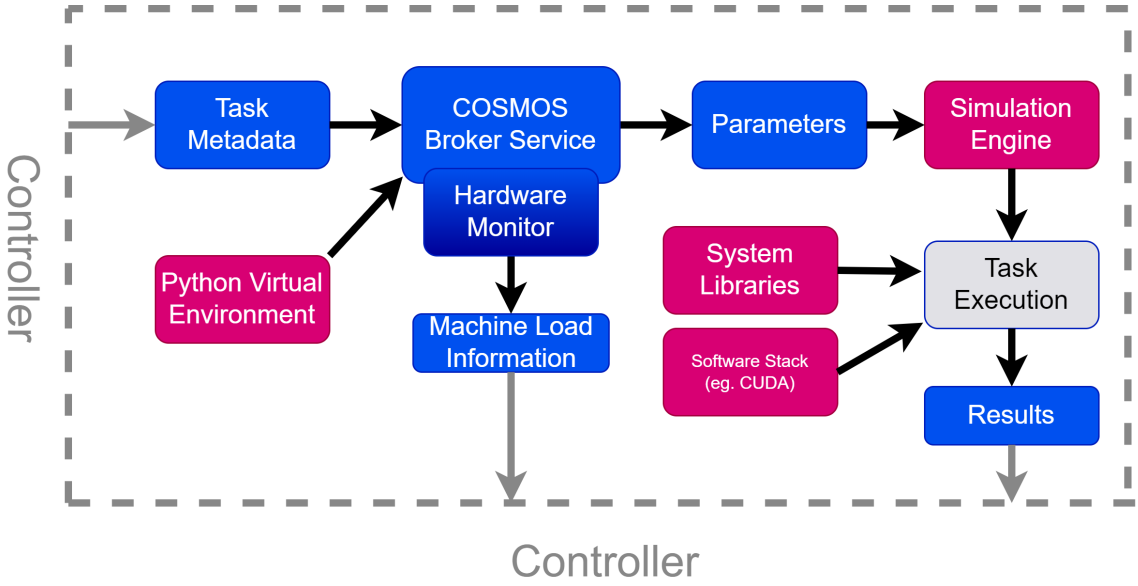

Fig. 2.4 Broker Interactions Diagram

Calculul eterogen distribuit se referă la o serie de mașini interconectate cu arhitecturi si specificații de performanță diferite. Cunoașterea specificaţiilor mașinilor, cum ar fi memoria, lățimea de bandă, puterea de procesare, dimensiunea si tipul mediului de stocare, poate fi utilă atunci când încercați să scrieți o politică personalizată de programare a sarcinilor pentru un mediu de cluster.

În paragrafele următoare, vom încerca să abordăm problema prezentată anterior. Având în vedere faptul că într-un scenariu din lumea reală, cel mai probabil, sarcinile au dependențe, folosim COSMOS nostru pentru a testa diferite politici de programare a sarcinilor.

Chiar dacă ar fi să simplificăm problema prin eliminarea dependențelor task-urilor, găsirea unei soluții optime pentru problema de planificare rămâne NP-completa. Prin urmare, este necesar să existe un algoritm euristic pentru generarea unui program care să utilizeze toate resursele disponibile cu o risipă minimă și să obțină un rezultat *rezonabil*. Înainte de a proiecta soluția finală pentru programarea sarcinilor în Brokerul nostru, folosim CloudSim [46] și propriul nostru generator de sarcini pentru a simula un mediu real și a verifica performanța fiecărei metode de planificare a sarcinilor.

În această secțiune, prezentăm trei euristici de planificare statică utilizate de Broker pentru a minimiza durata de finalizare a task-urilor într-un mediu distribuit eterogen simulat.

Intrarea pentru brokerul nostru este reprezentată de T = [T_1, T_2 ... T_n], un set de sarcini și M = [M_1 , M_2 ... M_m], o listă de mașini disponibile.

In urmatorul paragraf, o sa prezentam principalele euristici pentru programare a task-urilor implementate in COSMOS:

- **Min-Min** - Euristica Min-Min preia setul U de sarcini nemapate și prioritizează sarcina cu timpul minim de finalizare (MCT). În prima fază a algoritmului, sarcinile sunt inserate într-o coadă de priorităţi folosind lungimea task-urilor ca metrică de comparație. În faza următoare, timpul de finalizare pentru fiecare mașină disponibilă este calculat pentru sarcina principală a cozii prioritare și este selectată mașina care poate rezolva sarcina cel mai rapid. Primul task din coada de prioritati este actualizat cu următoarea sarcină (din punct de vedere al prioritatii) și este adaugat timpul de finalizare a sarcinii la timpul total de încheiere estimată pentru timpul de mașină selectata. A doua fază a algoritmului se repetă până când coada de prioritate cu sarcini nemapate este goală.

- **Max-Min**: Euristica Max-Min utilizează, de asemenea, metrica MCT. Diferența față de euristica Min-Min descrisă anterior este că această euristică ia setul U de sarcini nemapate, prioritizează sarcina cu lungimea maximă și selectează mașina unde această sarcină are timpul minim de finalizare. În prima fază a algoritmului, sarcinile sunt inserate într-o coadă de prioritate folosind lungimea sarcinii ca metrică de comparație. În următoarea fază, timpul de finalizare pentru fiecare mașină disponibilă este calculat pentru prima sarcina din coada de prioritati și selectațata mașina care poate rezolva sarcina cel mai rapid. Primul task din coada de prioritati este actualizat cu următoarea sarcină (din punct de vedere al prioritatii) și este adaugat timpul de finalizare a sarcinii la timpul total de încheiere estimată pentru timpul de mașină selectata. A doua fază a algoritmului se repetă până când coada de prioritate cu sarcini nemapate este goală.

- **Work-Queue**: WorkQueue este una dintre cele mai simple euristici pentru programarea seturilor de sarcini independente. În primul pas, euristica selectează aleatoriu o sarcină și o atribuie mașinii cu o sarcină de lucru minimă (adică mașinii în care timpul de încheiere a sarcinii este cel mai mic).

- **Priority-Queue** : De asemenea, am implementat programarea prioritară ca politică în cadrul nostru. După cum sugerează și numele, sarcinile de intrare sunt programate pe baza priorității specificate de utilizator, alegand masina ce finalizeaza task-ul ales cel mai rapid. Cu toate acestea, nu analizăm performanța acestei metode, deoarece programarea efectivă se face strict folosind specificațiile utilizatorului și nu se iau decizii bazate pe un algoritm.

Rezultate CloudSim

După cum am explicat pe scurt în secțiunile anterioare, avem nevoie de câteva sarcini de intrare pentru metodele noastre de programare. Din perspectiva CloudSim, am implementat metode care pot genera cloudlet-uri sau le pot citi din fișierele existente. Cloudlet-urile sunt generate cu valori ale atributelor aleatorii în anumite intervale predefinite și sunt scrise în fișiere.

Motivul principal pentru care am limitat valorile intre niste limite prestabilite este că am dorit să simulăm arhitecturile noastre de testare. 4.744 MIPS reprezintă valoarea de vârf estimată pentru un nucleu Cortex A-72 la frecventă maximă, iar 43.144 MIPS reprezintă valoarea de vârf estimată pentru un nucleu Ivy-Bridge.

Dacă analizăm algoritmii descriși în secțiunile de mai sus, pentru sarcinile **n** și mașinile **m** euristicile de planificare au următoarea complexitate:

- Min-Min: $O(n*m+n*log(n))$
- Max-Min: $O(n*m+n*log(n))$
- Work-Queue: $O(n*m)$

Pe baza timpului mediu de finalizare pentru toate cele 5000 de sarcini de lucru, putem clasifica algoritmii pe baza unor valori relevante, cum ar fi:

- timpul mediu de finalizare al unui task: Max-Min, WorkQueue, Min-Min
- timpul minim de finalizare al tuturor task-urilor: Max-Min, WorkQueue, Min-Min
- complexitatea algoritmului: WorkQueue, Max-Min & Min-Min

Capitolul 3

Aplicaţii de Simulare si Optimizare

Acest capitol prezintă aplicații ale software-ului de calcul și optimizare de înaltă performanță integrate folosind framework-ul COSMOS.

CORSIKA (COsmic Ray SIMulations for KAscade)[15] simulează interacțiuni hadronice neliniare de înaltă energie. Datorită timpului mare de execuție pentru experimentele ce simuleaza un eveniment reprezentativ pentru lumea reală, reprezintă pentru noi o țintă viabilă pentru paralelizare și ajustarea parametrilor folosind COSMOS Framework.

GROMACS (GROningen Machine for Chemical Simulations)[13] este un software de dinamică moleculară și simulează evoluția sistemelor de particule folosind ecuații newtonice ale mișcărilor. Am ales această aplicație pentru a simula evoluția moleculelor de ADN atunci când se aplică forțe externe asupra lor. Integrarea GROMACS în COSMOS ne permite să oferim un design experimental optim și să determinăm automat cea mai bună configurație a task-urilorde simulare pentru a minimiza timpul de execuție pentru o anumită configurație hardware.

LAMMPS (Large-Scale Atomic/Molecular Massively Parallel Simulator)[28] este de asemenea un software de dinamică moleculară care poate simula evoluția sistemelor de particule, utilizat în special în modelarea materialelor. Pentru aceast tip de aplicație, ne concentrăm pe generarea automată a fișierelor de comandă de intrare și a fisierului de intrare ce specifica geometria atomilor, utilizate pentru simularea absorbției hidrogenului în nanofire de oxid de zinc.

Următoarele secțiuni din acest capitol concentrează aceste trei aplicații și le prezentăm în scopul experimentelor noastre.

3.1 CORSIKA - Simularea Jerbelor

O jerbă reprezintă întreaga evoluție a traiectoriei prin atmosferă unei particule primare de energie mare. CORSIKA [14] este un program scris în FORTRAN și C++ ce aceasta jerbă folosind modele fizice și matematice. Vom prezenta soluții pentru paralelizarea și optimizarea acestui tip de simulare într-un mediu de calcul eterogen.

Având în vedere faptul că interacțiunile cu energie ridicată nu au fost încă pe deplin înțelese, studiul jerbelor rămâne un *instrument* important care ajută acest domeniu de cercetare. Punctul initial al unei jerbe este reprezentat de momentul când o particulă primară cu energie mare intră în atmosferă. Un exemplu de astfel de simulare în CORSIKA, realizat de Fabian Schmidt de la Universitatea din Leeds, poate fi văzut în Figura 3.1a din perspectiva planului XOZ, în timp ce aceeași simulare din perspectiva XOY este prezentată în Figura 3.1b.

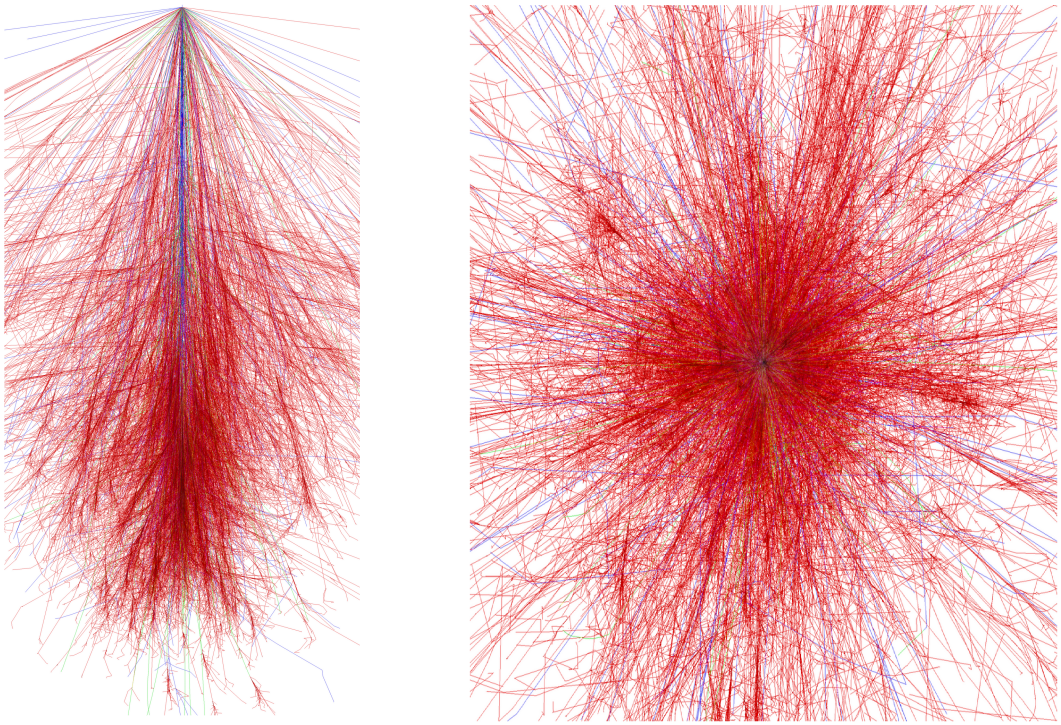

Fig. 3.1 Jerbă generată de particulă de Fier cu energie primară de 1 TeV [43] a) proiectie in plan XZ b) proiectie in plan XY

Particula primară poate fi un proton, un nucleu, un foton, un electron sau un pozitron. Când această astro-particulă se ciocnește cu nucleul unui atom din atmosferă la o viteză foarte

mare (adică aproape de viteza luminii), se creează hadroni energetici, ca muoni, protoni, anti-protoni, electroni etc. Într-un scenariu de viață reală, jerba poate fi observată datorită efectelor care se creează în atmosferă sau la nivelul solului: un *fulger de lumină* și lumină fluorescentă produsă de particulele secundare și undele radio emise de deviația pozitronilor și electronilor de către câmpurile geomagnetice.

Detectarea la nivelul solului se face folosind rezervoare de apă și se bazează pe efectul Cerenkov [7]: radiația electromagnetică este emisă atunci când o particulă încărcată trece printr-un mediu dielectric cu viteze mai mari decât viteza de fază a luminii în acel mediu. O alta metoda de detectare, alta decat folosirea telescoapelor folosite pentru a observa lumina produsa in atmosfera, este detectia radio cu ajutorul antenelor. Metoda ulterioară are un mare avantaj față de alte tehnici optice: este eficientă și în timpul zilei și când cerul este înnorat.

Structura unei jerbe este influențată de unele dintre atributele cheie ale particulei primare, cum ar fi tipul particulei, energia sau direcția. Un alt factor care joacă un rol cheie în evoluția acesteia este mediul înconjurător: compoziția și densitatea atmosferei, puterea și orientarea câmpului magnetic al pământului etc. După cum am afirmat anterior, suntem interesați să studiem interacțiunile de înaltă energie. Datorită faptului că fluxul de particule de energie foarte mare (mai mare de 10^{20}eV) este foarte mic (o particulă pe kilometru pătrat la fiecare 100 de ani), particulele cu energii mai mici (mai mare de 10^{16} eV), care sunt mai frecvente, merită și ele urmărite.

Având în vedere că probabilitatea de observare a unei particule de energie foarte mare este extrem de scăzută, este absolut necesara dezvoltarea unei aplicatii pentru simularea detaliată a averselor de aer extinse. Motivați de decalajul dintre software și hardware, în ceea ce privește cât de eficient sunt utilizate resursele hardware de către această simulare fizică, vom oferi o soluție pentru paralelizarea și optimizarea codului folosind cadrul COSMOS descris anterior.

Pachet de optimizare pentru CORSIKA

Etapa de optimizare este realizată de modulul de optimizare din COSMOS. Pe baza coordonatelor particulelor primare, a unghiurilor Azimut și Zenit, precum și a cotei observatorilor specificate în fișierele de intrare, putem calcula traiectoria jerbei și câteva puncte de proiecție discretizate pe planul XY, așa cum se arată în Figura 3.2. Pentru fiecare dintre punctele de proiecție, căutăm antenele din proximitate folosind distanța euclidiană și excludem acele antene care sunt prea departe de traiectoria jerbei. Această optimizare este posibilă datorită faptului că semnalul relevant dintr-o simulare poate fi analizat de către fizicieni doar pentru antenele din apropierea dușului de aer.

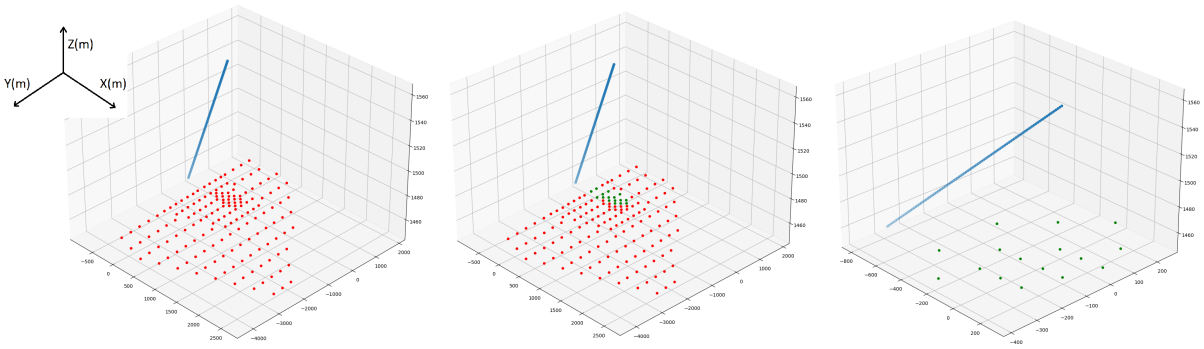

Fig. 3.2 Optimizator CORSIKA cu simulare de 156 de antene

O altă capacitate a optimizatorului este de a împărți fișierele de intrare și de a selecta doar un subset de antene folosind fie o divizare N = N x 1 (N fișiere de intrare mai mici de 1 antenă), fie o divizare N = P x Q (P fișiere de intrare mai mici de antena Q).

Rezultate

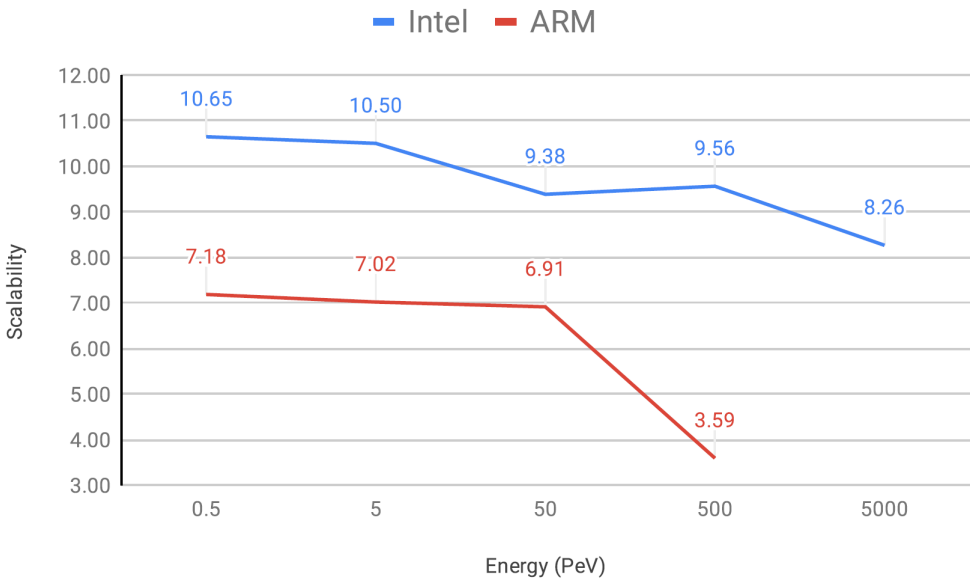

Fig. 3.3 Scalare de tip *weak* pentru aplicatii optimizate

După cum am explicat în Capitolul 2.1, am folosit în testarea noastră două arhitecturi diferite, și anume Intel pentru cea mai bună performanță (operatii in virgula mobila pe secunda - *FLOPS*) și ARM pentru cea mai bună eficiență energetică. După cum am explicat anterior, am reușit să paralelizăm CORSIKA împărțind fișierele de intrare și împărțind

sarcina în sarcini mai mici. Am ales strategia N = P x Q prin distribuirea automată a antenelor $floor(N/P)$ pe procesele P și apoi adăugând o antenă la primele procese $N - floor(N/P) * Q$. Folosind algoritmul pe care tocmai l-am prezentat, am ajuns la următoarele configurații:

- Configurație Intel: 16 fișiere de intrare cu 8 antene și 4 fișiere de intrare cu 7 antene. În total, am generat 20 de procese independente.
- Configurație ARM: 4 fișiere de intrare cu 20 de antene și 4 fișiere de intrare cu 19 antene. În total, am generat 8 procese independente.

Având în vedere configurarea acestui experiment, rezultatele scalabilității pot fi observate în Figura 3.3. Deoarece timpul de simulare crește exponențial cu energia particulelor primare, pe ARM a fost fezabil să se simuleze doar energii de intrare în intervalul de la $0,5$ PeV la 500 PeV. O altă limitare cu care ne-am confruntat cu simulările pentru particule de energie mare a fost lipsa memoriei disponibile pentru fiecare proces. Scăderea scalabilității a fost vizibilă pentru energiile de intrare mai mari de 300PeV atunci când fiecare proces a rămas fără memoria principală și a început să folosească din spatiul de stocare de pe disk.

3.2 GROMACS - Simulare de Dinamică Moleculară

În această secțiune, vom prezenta capabilitățile COSMOS pentru a oferi un design experimental optim pentru simulările GROMACS. Experimentul vizat este format din simulari de dinamică moleculară si explicat in detaliu in lucrarea *Molecular Dynamics Simulations of DNA Adsorption on Graphene Oxide and Reduced Graphene Oxide-PEG-NH_2 in the Presence of Mg^{2+} and Cl^{-} ions* [33]. Grafenul și derivații săi funcționalizați transformă dezvoltarea biosenzorilor care sunt capabili să detecteze hibridizarea acidului nucleic. Folosind o abordare a dinamicii moleculare, am explorat adsorbția acidului nucleic dezoxiriboză monocatenar sau dublu catenar (ssDNA sau dsDNA, așa cum se arată în Figura 3.4) pe două specii grafenice: oxidul de grafen și reducerea oxid de grafen funcționalizat cu polietilen glicol aminat (rGO-PEG-NH2). În mod inovator, am inclus ioni de clorură (Cl-) și magneziu (Mg2+) care au influențat atât adsorbția ssDNA, cât și dsDNA pe suprafețele GO și rGO-PEG-NH2. Spre deosebire de Cl-, ionii divalenți de Mg2+ au format punți între suprafața GO și moleculele de ADN, promovând adsorbția prin interacțiuni electrostatice. Pentru rGO-PEG-NH2, ionii Mg2+ au fost respinși de pe suprafața grafenică. Adsorbția ulterioară ssDNA, influențată în principal de forțele electrostatice și legăturile de hidrogen, ar putea fi susținută de interacțiuni de stivuire pi-pi care au fost absente în cazul dsDNA.

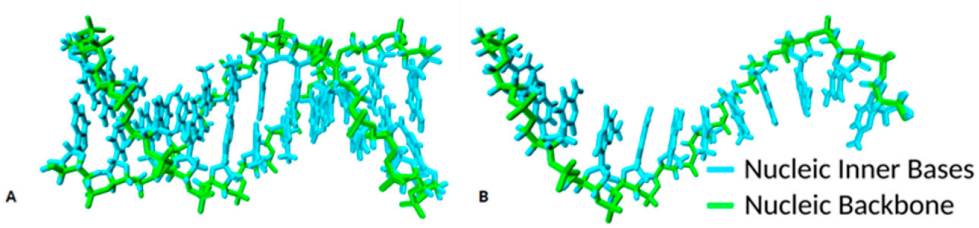

Fig. 3.4 DNA molecules used, generated with the sequence 5'-CGCGAATTCGCG-3': (A) Double-stranded deoxyribose nucleic acid (dsDNA); (B) Single-stranded DNA (ssDNA)

Pachet de optimizare pentru GROMACS

Așa cum am prezentat anterior, COSMOS oferă o flexibilitate ridicată, care permite sarcinilor curente să creeze task-uri și să rescrie dependențele acestora (singura limitare este că un job nu ar trebui să fie în stare de execuție sau terminat în punctul in care se adauga noi dependențe). Pentru a putea obține o programare optima spre executie, a fost necesar să reducem complexitatea experimentului nostru de dinamică moleculară, astfel încât să putem estima performanța configurației experimentului final într-o perioadă limitată de timp. Folosind opțiunea oferita de GROMACS *maxh* [13], am permis simulării noastre să ruleze timp de maximum 0,33 ore per task. În acest experiment de profilare, am creat 72 de sarcini ce necesita un singur server și 144 de sarcini ce necesita doua sau trei servere, cu un timp total de execuție de 72 de ore. Pentru a evalua performanța configurației experimentului, extragem pasul curent de simulare, calculăm *nanosecunde pe zi* și îl folosim ca metrică de evaluare si comparare. Scopul nostru este de a furniza automat parametrii pentru simulare care controlează utilizarea hardware-ului și programarea pe serverele disponibile, vizând fie o utilizare optimă a resurselor, fie un timp minim de execuție.

Pentru acest experiment, folosim aceleași servere ca cele descrise în Secțiunea 2.1 cu două NVidia Tesla K40 conectate la fiecare mașină. Primul pas în acest proces este de a descrie formal parametrii care controlează:

- NSERV: numarul de servere: 1, 2 or 3
- NPROC: numarul de procese MPI: 1, 2, 4, 6, 10, 20, 40 or 60
- NOMP: numarul de threaduri OpenMP per proces MPI: 1, 2, 4, 10, 16, 20
- NGPUS: numarul de GPU-uri: 0, 2, 4, 6
- THAFF: afinitate threaduri: on, off

După ce experimentul a fost parametrizat, am scris metode de optimizare care au fost capabile să ofere valorile optime fie pentru cea mai bună utilizare a hardware-ului, fie pentru cea mai bună performanță. În acest moment, vom folosi terminologia *execuție în serie* pentru

un experiment care se rulează pe 1 server, 1 proces MPI, 1 fire de execuție OpenMP per proces MPI, 0 GPU și afinitățile threadurilor dezactivate. În plus, execuția seriala va fi punctul nostru de referință atunci când vom analiza rezultatele scalabilității pentru fiecare configurare.

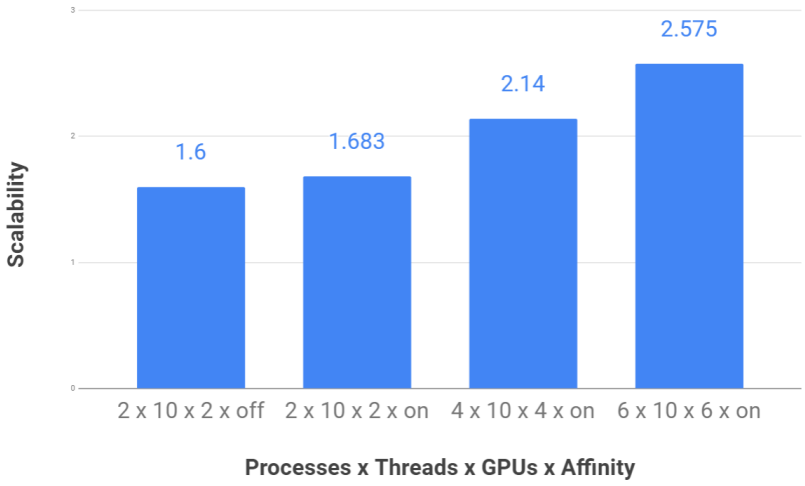

Fig. 3.5 Scalabilitatea configuraţiilor

Rezultate

În figura 3.5a, putem vedea rezultatele de scalabilitate pentru un set de configurații reprezentative. **Rezultatele referinţă** pentru această diagramă sunt reprezentate de ceea ce considerăm o rulare serială: un proces MPI, un thread OpenMP per proces MPI, fără acceleratoare și afinitatea threadurilor dezactivată. Pentru configurația serială, scalabilitatea este considerată 1. Prima și a doua coloană arată scalabilitatea obținută utilizând un singur server cu afinitatea setată *on* și, respectiv, *off*. Am ales să le trasăm pe ambele pentru a contura impactul setării afinității threadurilor. Datorita acestei setări, sistemul de operare nu petrece timp cu migrarea threadurilor de pe un nucleu pe altul. De asemenea, deoarece firele sunt fixate pe un nucleu, cache-urile CPU nu mai sunt invalidate ca in cazul in care threadurile incep executia pe alt nucleu. În comparație cu valoarea de referinta, se obține o accelerare de 1.683 ori dacă sunt utilizate toate resursele unei mașini.

Ultimele două coloane din Figura 3.5a arată scalabilitatea obținută atunci când se utilizează două și, respectiv, toate cele trei mașini disponibile. Cu toate acestea, câștigurile de accelerare prezentate în aceste două configurații sunt umbrite de scăderea eficienței. Pentru a rula cu 53% mai rapid, în comparație cu configurația *2 x 10 x 2 x off*, trebuie să folosim de 3 ori mai multe resurse.

Pentru a obține rezultatele noastre finale ale absorbției ADN pe acoperiri cu grafen, am ales o configurație de 2 procese MPI, 10 fire OpenMP per proces MPI, 2 GPU și afinități ale firului setate la *on* (a doua coloana din Figura 3.5). Chiar dacă scalabilitatea este mai bună atunci când utilizați toate cele 3 mașini, așa cum am explicat anterior, eficiența scade semnificativ și nu este rentabil economic să rulăm în acea configurație timp de mai multe săptămâni de simulare.

3.3 LAMMPS - Simulare de Dinamică Moleculară

Dinamica moleculară este o tehnică de simulare a evoluției în timp a unui sistem de particule prin calcularea forțelor dintre atomi și integrarea celei de-a doua legi a lui Newton. Vitezele inițiale ale particulelor sunt în general produse prin scheme de generare aleatorie bazate pe funcții gaussiene foarte apropiate de distribuția Maxwell-Boltzmann care caracterizează starea de echilibru a sistemelor moleculare. Cu toate acestea, aceste viteze inițiale nu iau în considerare interacțiunea dintre particule și sistem trebuie aduse la echilibru înainte de a colecta date structurale și dinamice pentru a caracteriza sistemul. Astfel de simulări de dinamică moleculară necesită resurse de calcul mari în faza de echilibrare, dar, din această etapă a simulării, doar pozițiile și vitezele finale ale particulelor care urmează forțele de interacțiune ale sistemului vor fi salvate și utilizate în următoarea etapă a simulării. Pentru a reduce timpul de calcul pentru echilibrarea sistemului, pot fi dezvoltate metode de determinare a vitezelor inițiale.

LAMMPS este un pachet de metode de simulare a particulelor și a câmpului care permit dinamica moleculară sau simulări Monte Carlo pentru sistemele de particule. Programul conține un număr foarte mare de câmpuri de forță care pot fi aplicate la scară nano-, mezo- sau chiar macro-scopică pentru majoritatea materialelor existente, sau virtuale. Particulele pot fi electroni, atomi, dipoli, atomi uniți și particule cu granulație grosieră, granule sferice și elipsoidale și combinațiile lor. După cum sugerează și numele programului LAMMPS - *Large-scale Atomic/Molecular Massively Parallel Simulator* - acesta este unul dintre cele mai avansate programe dezvoltate pentru sistemele HPC, cu o scalare paralelă foarte bună, capabil să fie rulat pe sisteme HPC cu sute de mii de nuclee [30], pentru sistemele de intrare formate din milioane de particule.

Fișierul de intrare LAMMPS este de un script ce permite într-un mod flexibil construcția sistemului, alocarea particulelor, vitezele inițiale, aplicarea forțelor externe, precum și controlul evoluției sistemului. Pentru a analiza rezultatele și a modifica sistemele de particule, programul Tools4Lamps [45, 26] a fost dezvoltat în cadrul Programului *Summer Guest-Student* la Juelich Supercomputing Center, Forschungszentrum Jülich GmbH, Germania.

Programul este un instrument care permite caracterizarea echilibrării sistemului, manipularea geometrică a subcomponentelor sistemului și modificarea vitezei acestora pe baza unor noi algoritmi.

COSMOS este folosit aici pentru a integra cele două programe LAMMPS și Tools4Lammps și pentru a automatiza execuția lor pe sisteme de calcul HPC.

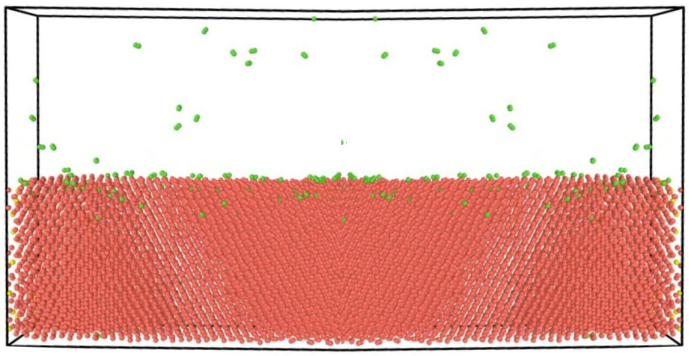

Fig. 3.6 Simularea absoptiei de Hidrogen in structuri de Oxid de Magneziu

În figura 3.6, putem vedea un sistem de simulare care conține un instantaneu în care unii atomi de hidrogen (verzi) sunt absorbiți în structura firului (roșu și galben). Vom considera simularea LAMMPS ca o cutie neagră și nu vom discuta aplicabilitatea și detaliile simulării în sine, deoarece aceastea nu reprezintă scopul acestei teze. Integrarea COSMOS are două obiective principale:

- oferirea unei interfață comună de comunicare între LAMMPS și pachetul de optimizare Tools4Lammps printr-un limbaj de pseudo-scripting
- generarea structurilor de intrare pentru simulările LAMMPS: nanofire ZnO cu parametri specifici de lungime, lățime și procente de imperfecțiuni

După cum am precizat anterior, obiectivul nostru pentru integrarea LAMMPS în COSMOS este de a închide bucla de optimizare folosind Tools4Lammps, acesta fiind în același timp capabil să interacționeze cu geometria de intrare și fișierul de comandă. Acest lucru ne oferă o mare flexibilitate, iar utilizatorul final poate automatiza întregul proces de simulare folosind framework-ul nostru.

Pachet de optimizare pentru LAMMPS

Simularea LAMMPS este caracterizată de două tipuri de meta-parametri, fișierul de comandă de intrare ce stabilește parametrii simulării impreună cu tipurile de operații și fișierul de date care conține geometria atomilor. Fișierul de comandă de intrare este folosit de noi

```
1
2  read_data ZnO.data
3  variable radius equal 20
4  variable dx equal 50
5  variable length equal 100
6  replicate dx dx ${length}
7  region DEL cylinder z  ${radius} ${radius} ${length}  \
8                      INF INF side out units box
9  delete_atoms region DEL
```

Listing 1 Fisier de comanda ce genereaza nanofirul de Oxid de zinc folosind LAMMPS

pentru a extrage informații specifice despre structura atomică, iar optimizarea procesului de simulare se face folosind pachetul extern Tool4LAMMPS. Pentru integrarea cu LAMMPS, ne concentrăm pe operaţiile care se ocupă de structura fișierului de date. Pentru a citi și modifica geometria de intrare, am scris un modul intern de analiză în cadrul COSMOS pentru acest tip de fișier de intrare [31]

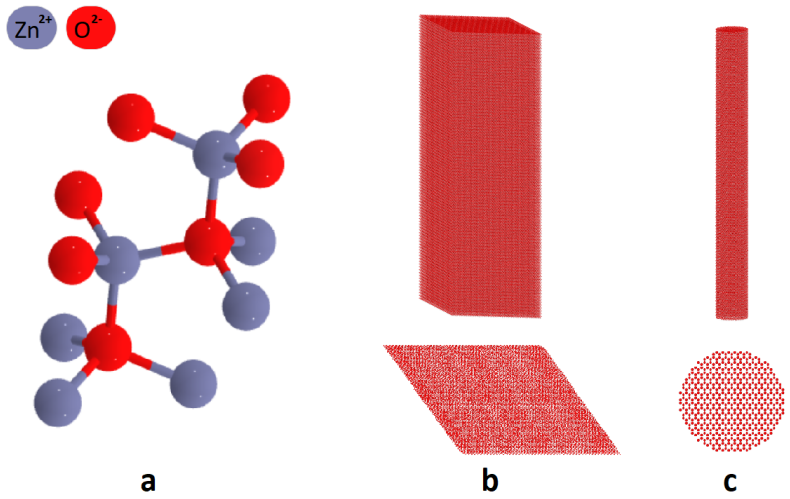

Fig. 3.7 Structură de Oxid de Zinc

Rezultate

Folosirea LAMMPS pentru a genera structurile de Oxid de Zinc

O metodă de a genera nanofirul prezentat în Figura 3.7c este de a scrie un fișier de comandă de intrare pentru LAMMPS unde putem descrie parametrii structurii și operaţiile

care pot fi făcute pentru a obține rezultatul dorit. Un exemplu de fișier de comandă de intrare este enumerat mai jos si realizează acest lucru prin următoarele acțiuni:

- citirea celulei inițiale de ZnO simetrizată simplă, constând din 4 atomi, așa cum se arată în Figura3.7a
- definirea razei și lungimei nanofirului folosind comanda *variable* (în acest exemplu special, generăm un nanofir cu un diametru de 40 nm și o lungime de 100 nm)
- generarea unui box de simulare pornind de la celula inițială având următoarele dimensiuni: dx = dy = 50nm și dz = 100nm. Pentru a face acest lucru folosim comanda *replicate*. Structura de ieșire după această comandă poate fi văzută în Figura 3.7b
- la acest pas, avem o cutie de atomi si trebuie să definim limitele structurii nanofirului necesara pentru simulare
- în cele din urmă, folosim comanda *delete_atoms* pentru a elimina toți atomii care se află în afara nanofirului

S-ar putea observa că este destul de ușor să generați o astfel de structură cilindrică din LAMMPS în sine și, pentru unele scenarii, este de dorit să urmați această procedură, decât să vă scrieți propriul generator de structuri moleculare. Cu toate acestea, principalele dezavantaje ale acestei abordări sunt:

- lipsa flexibilitaţii in definirea structurilor complexe de atomi
- este greu sa generăm imperfectiunile din nanofirele de oxid de zinc pentru a putea replica strucura celor *crescute* in laborator
- dat fiind faptul ca simularea si pasii intermediari sunt controlati prin intermediul COSMOS, este greu sa generam toate structurile de atomi pentru simulari multiple fara a folosi o metoda programatică
- desi metoda de generare folosind LAMMPS este usor de folosit, ii lipseste abilitatea de a reordona lista atomilor astfel incat sa optimizam accesele la memorie si sa limitam comunicarea intre threaduri

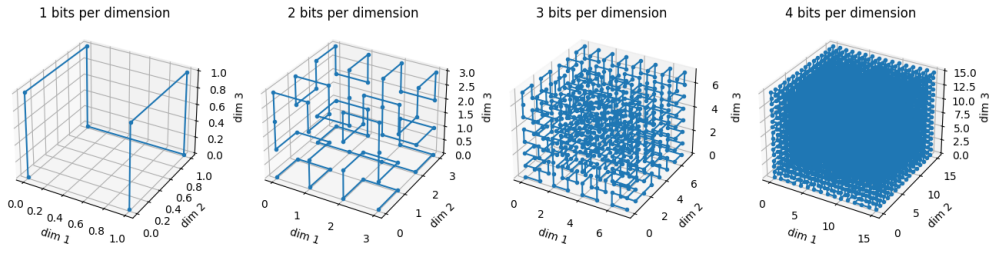

Fig. 3.8 Curba Hilbert curve in spaţiul 3D [34]

Folosirea LAMMPS pentru a genera structurile de Oxid de Zinc

Această secțiune prezintă o metodă de generare a nanofirelor ZnO folosind COSMOS care obține același rezultat ca metoda echivalentă LAMMPS descrisă în secțiunea anterioară. Generăm nanofirul de ZnO pornind de la o structură de intrare așa cum se arată în Figura 3.7a. Vom replica apoi box-ul inițial pentru a genera o supercelulă, așa cum se arată în Figura 3.7b și vom începe să eliminam atomii care se află în afara nanofirului. Pentru fiecare atom pe care îl ștergem din structura finală, eliminăm și fiecare legătură cu atomii din exteriorul nanofirului.

În pasul următor, vom genera și unele imperfecțiuni: stergerea de atomi folosind o probabilitate dată, deplasarea unor atomi pe una din direcțiile sale sau indepartarea aleatorie a unor legaturi dintre atomi. Înainte de a face scrierea finală în fișierul de date LAMMPS, putem rearanja lista de atomi folosind un algoritm pentru minimizarea comunicării între procese atunci când LAMMPS utilizează MPI pentru paralelizare. Un astfel de algoritm pentru reordonarea datelor de intrare LAMMPS este curba Hilbert pentru de umplere a spațiului. Există mai multe implementări ale acestui algoritm în majoritatea limbajelor obișnuite de programare, dar, pentru abordarea pe care am luat-o în COSMOS, putem folosi versiunea NumPy [34]. După cum se arată în Figura 3.8, putem codifica pozițiile a 16.777.216 atomi folosind doar un octet pe fiecare dimensiune. Unul dintre principalele motive pentru care am dori să rearanjam lista urmând curba Hilbert pentru un spațiu 3D este că îmbunătățește localitatea datelor noastre, reducând astfel comunicarea într-un sistem distribuit. O abordare similară, dar în domeniul randării, este folosită de Keller și colab. [25] pentru a randa de-a lungul curbei Hilbert.

În cele din urmă, rezultatul obținut este un nanofir cu o înălțime și o rază predeterminate, așa cum se arată în Figura 3.7c. S-ar putea observa că structura rezultată este identică cu cea generată în LAMMPS, dar acum avem control deplin asupra acesteia folosind framework-ul COSMOS.

Capitolul 4

Integrarea COSMOS cu alte aplicaţii si domenii conexe

Ţn capitolele anterioare, am discutat despre integrarea framework-ului cu aplicaţii de simulare și pachete de optimizare relevante pentru domeniul nostru de cercetare. După cum am afirmat anterior în capitolul 1, o listă exhaustivă a acestui tip de software nu poate fi acoperită de această teză. În acest capitol, vom prezenta alte lucrări pe care le-am realizat, care fie sunt potrivite pentru a fi integrate cu COSMOS, fie aduc o perspectivă nouă pentru calculul de înaltă performanță. Contribuțiile care merită remarcate vor acoperi:

- simulare numerică: algoritmi de sortare topologica a datelor colectate in experimentul ATLAS folosind CUDA
- optimizare numerica: optimizarea algoritmilor de căutare in spatiu 3D
- planificare de job-uri pentru aplicațiile de procesare a imaginilor medicale folosind algoritmi de ţnvaţare automată
- academic: integrarea calculului paralel în curriculumul Universității Politehnica din București

4.1 Sortarea topologica a Calorimetrului pe GP-GPU la ATLAS folosind CUDA

În această secțiune, prezentăm munca pe care am realizat-o în colaborare cu CERN privind procesarea unor cantități mari de date obținute de la Large Hadron Collider (LHC). După cum se menționează în experimentul ATLAS de la CERN Large Hadron Collider [9], o cantitate uriașă de date va fi colectată din coliziunile $4 * 10^7$ pe secundă de 10^{11} protoni (p) și ioni grei (A), în special nuclee de plumb. Energia coliziunii proton-proton (p-p) este de

14 TeV și 5,5 TeV per pereche de nucleoni (A-A). Două detectoare de uz general, ATLAS (A Toroidal LHC ApparatuS) și CMS (Compact Muon Solenoid) au fost construite pentru achizitia datelor obtinute in urma coliziunilor p-p și A-A. Structura detectorului ATLAS poate fi văzută în Figura 4.1.

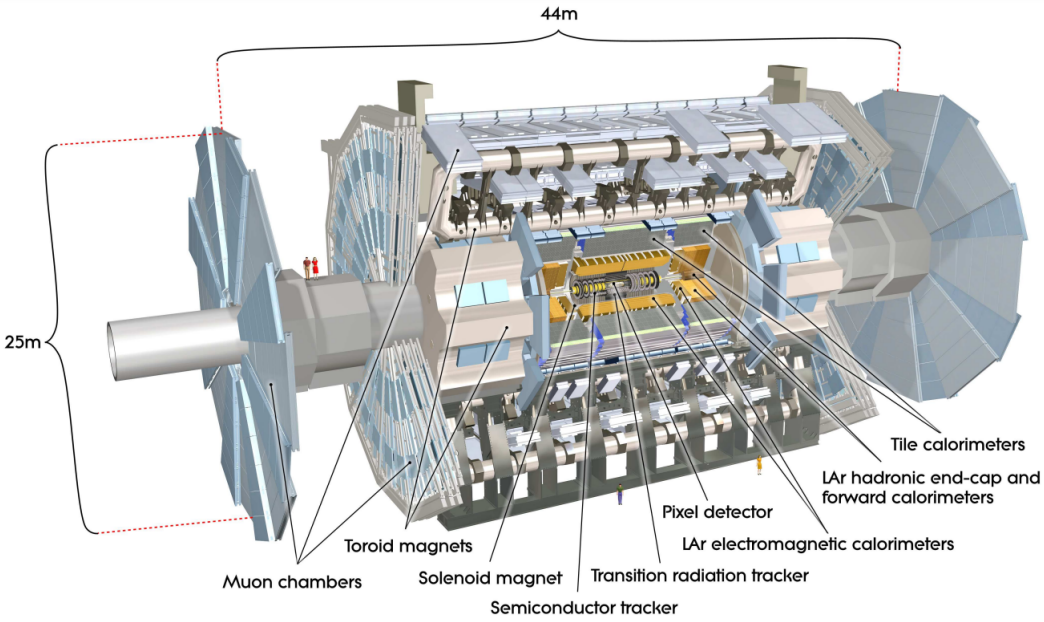

Fig. 4.1 Secţiune a detectorului ATLAS [10]

Din detectorul ATLAS, ne interesează subsistemul Calorimetru, prezentat în Figura 4.2, și algoritmii care prelucrează datele obținute din acesta. Calorimetrul ATLAS este format din celule detectoare individuale plasate într-o geometrie cilindrică. Fiecare celulă individuală va înregistra informații atunci când au loc coliziuni de particule și aceste date sunt stocate pentru a fi ulterior procesate de algoritmi furnizați de ATHENA Framework [5]. Unul dintre obiectivele principale ale acelor algoritmi este de a analiza evenimentele de coliziune care au loc în interiorul calorimetrului și de a construi clustere topologice pe baza energiei înregistrate de celule. Sortarea topologică [10] are o etapă de creștere atunci când celulele vecine sunt grupate în grupuri pe baza energiei lor înregistrate. A doua etapă împarte clusterele generate în partea în creștere numai dacă acestea conțin mai mult de o celulă având o energie locală maximă înregistrată.

Munca noastră în acest proiect a fost să analizăm algoritmii actuali de divizare a clusterelor scrise în C++ și să le portăm pe GPU folosind CUDA. În procesul de dezvoltare a codului, a trebuit să facem și conversiile necesare pentru ca structurile de date să fie folosite pe *GPU*. Am ales să prezentăm acest lucru ca o contribuție la această teză, chiar dacă, la o prima analiza, s-ar putea observa că paralelizarea algoritmilor Athena are puține lucruri in

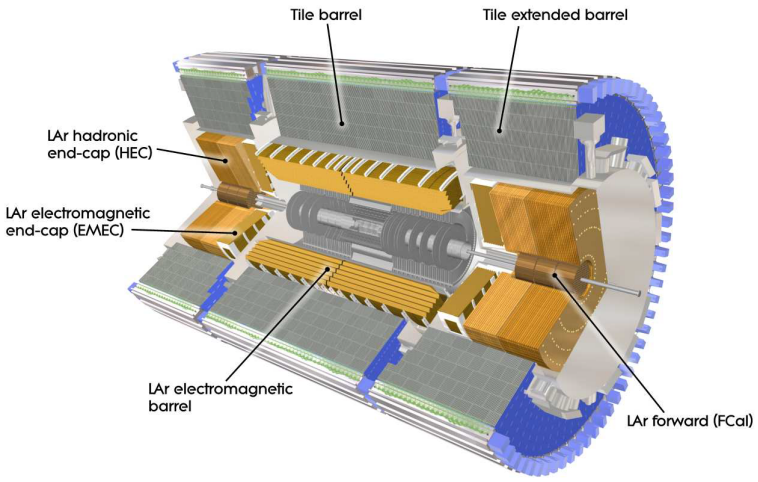

Fig. 4.2 Secţiune a Calorimetrului din ATLAS [10]

comun cu COSMOS. În realitate, ambele framework-uri integrează algoritmi de procesare și optimizare a datelor, la o scară diferită. În timp ce Athena se concentrează în principal pe algoritmi creați pentru procesarea datelor obținute în LHC, COSMOS este un framework general care poate furniza un spectru mai larg de pachete software de optimizare și simulare. Unul dintre principalele motive pentru care am ales să nu integrăm Athena în COSMOS este că dorim să păstrăm un domeniu general de aplicare pentru framework-ul nostru și intrgrarea necesita o cantitate non-triviala de resurse.

4.2 Optimizarea căutării 3D folosind COSMOS

Această secțiune prezintă o extensie a COSMOS prin adăugarea de metode de simulare și optimizare numerica specifice cautarilor in spatiu 3D continuu si finit. În consecință, este necesară introducerea de noi module care interacționează cu modulul Controller și care se concentrează pe procesul de post-optimizare pentru a determina etapele optime ale simulării numerice. Structura se bazează pe trei etape principale și anume: reconstrucția unui spațiu continuu 3D cu o cameră mobilă, colectarea informațiilor din scenă și procesarea acesteia; și procesul de optimizare care va determina următorul pas al simulării numerice, pe baza datelor colectate anterior.

Modulele principale COSMOS pot funcționa individual, deschizând posibilitatea de a aduce implementări separate și de a integra diverse alte module de simulare externe. Scopul principal este extinderea ariei de acoperire a simulărilor, prin introducerea unei noi probleme, o căutare informativă într-un spațiu 3D continuu. Având ca intrare vizualizarea inițială a scenei, se dorește pornirea căutării dintr-o poziție aleatorie pentru a determina coordonatele

camerei din perspectiva intrării, cu date colectate pe parcursul căutării. Pentru a atinge scopul, simularea ar trebui optimizată prin generarea celor mai buni pași următori ai căutării. Toate acestea vor fi sincronizate de principalele module COSMOS. În plus, va fi posibil ca, atât procesele de simulare, cât și de optimizare să fie utilizate în rezolvarea problemelor reale pentru vehiculele aeriene fără pilot. Domeniul de aplicare poate include urmărirea speciilor pe cale de dispariție [32], monitorizarea vegetației în agricultura de precizie [37] sau îmbunătățirea capacităților de căutare a sistemelor CCTV. Acest domeniu este vast și în creștere rapidă, datorită îmbunătățirilor continue și semnificative ale tehnologiei. După cum sa menționat anterior, proiectele pot fi împărțite cu ușurință în trei componente principale care vor necesita integrare:

- Reconstituirea scenei, în care se folosește Mitsuba2.0. Acest randare permite modificarea aspectului scenei, specificațiilor camerei, poziției și orientării, pentru a simula spațiul 3D continuu;
- Extragerea informațiilor din scenă, unde modulul principal introdus este *You Only Look Once: Unified, Real-Time Object Detection*, care, potrivit lui R. Girshick et al. [39], va detecta și va furniza date pentru maparea obiectului, pentru a estima coordonatele obiectului;
- Optimizarea, caz în care, pe baza rezultatelor date furnizate de a doua componentă descrisa anterior, următorul pas cel mai bun din simulare va fi calculat prin analiza obiectelor și a poziției lor în scenă.

După implementarea soluției propuse, urmeaza să fie introdus un pachet complet în COSMOS Framework, sporind capacitățile acestui framework. Va putea reda diverse scene 3D doar prin modificarea fișierului de intrare, să recunoască diferite obiecte din scenă și să estimeze poziția acestora. Mai mult, furnizȩnd un fisier de intrare (imagine initialıa) si o pozitie de pornire, prin combinarea caracteristicilor descrise mai sus, scena cautata va fi gasita intr-o maniera optimizata.

In ultimii ani s-a intensificat efortul de unificare a mediilor eterogene de calcul de înaltă performanță, care, in tradiţional erau separate. Acest lucru a oferit putere de calcul unei varietăți de aplicații de procesare a imaginilor în multe domenii, inclusiv astronomie (de exemplu, pentru analiza energiei materiei întunecate [27] sau pentru clasificarea galaxiilor [17].); bioinformatica (de exemplu, predicția la scară genomică a structurii și funcției proteinei [20]); fuziune (de exemplu, pentru reconstrucția profilului de plasmă 2D [18]), etc. În mod similar, în domeniul medical, HPC a fost utilizat pe scară largă pentru diferite sarcini de imagistică, inclusiv reconstrucție, restaurare, îmbunătățire, clasificare, segmentare sau detectare. Cu toate acestea, imagistica medicală prezintă provocări unice care se confruntă cu execuțiile la scară largă.

Procesarea imaginilor medicale folosind aplicatii dezvoltate pentru acest lucru se bazează, de obicei, pe metode de învățare automată. Datorită varietății mari a eșantioanelor utilizate pentru antrenament și a modalităților de inferență, aplicațiile sunt construite folosind o mare varietate de tipuri de metode AI/ML. În plus, faptul că studiile utilizate sunt încă experimentale având un caracter explorator, se adaugă și mai mult la natura dinamică și complexă a codurilor și optimizărilor. De exemplu, departamentul de neuroștiință de la Vanderbilt utilizează peste 50 de coduri care sunt în continuă dezvoltare și le combină în moduri diferite, în funcție de ceea ce trebuie testat fiecare studiu [23]. Optimizările pentru metodele AI ale GPU-urilor [35, 19] sau pentru procesarea datelor [49, 12] pot fi aplicate cu succes unor studii. Cu toate acestea, datorită gamei diverse de modele de comportament în procesarea imaginilor medicale, nu există o optimizare care să se potrivească tuturor scenariilor.

În această contributie, ne concentrăm pe analiza performanței unei aplicații de procesare a seturilor de imagini care utilizează un model de execuție cu date intensive, dar nu în sensul tradițional de a trata imagini de rezolutii mari. Aplicațiile cu randament ridicat mută accentul de la rulări individuale la ansamblu prin analizarea unui set de date uriaș de imagini de dimensiuni mici care trebuie să treacă printr-o serie de algoritmi și a căror debit trebuie optimizat. În acest scop, ne-am uitat la `SLANTbrainSeg` - Deep Whole Brain High-Resolution Segmentation - o serie de algoritmi de segmentare a creierului pe imagini de rezonanță magnetică structurală, care este esențială pentru înțelegerea relațiilor neuroanatomo-funcționale [48]. Obiectivul nostru principal este să aflăm cea mai bună modalitate de a rula această aplicație pe sisteme la scară largă pentru a crește randamentul acesteia, începând cu o comparație între versiunile existente ale programului și continuând cu aducerea modificărilor codului și modelului de execuție, astfel încât că utilizează la maximum arhitectura eterogenă a sistemului HPC. Contribuțiile noastre în acest domeniu sunt următoarele:

- analiza performanței aplicației de procesare a imaginilor de neuroștiință al cărei accent este pe obținerea unei utilizari eficiente a resurselor. De asemenea, identificăm limitarile atunci când implementăm un astfel de flux de lucru pe sisteme la scară largă.
- Pe baza rezultatelor propunem un nou model de execuție pentru *high-throughput* care se aplică aplicațiilor de procesare a imaginilor care au un volum mare de date, dar nu generează sau citesc imagini de dimensiuni mari.
- Comparăm rezultatele noului model pe diferite arhitecturi hibride (inclusiv Summit [29], supercomputerul de la ORNL) arătând accelerări de până la 4x și discutăm despre limitări și probleme viitoare deschise.

4.3 Scientific Computing in Higher Education

Acest capitol are scopul de a ne oferi o mai bună înțelegere a motivului pentru care am ales un anumit subset de tehnologii și limbaje de programare precum Python pentru dezvoltarea atât a COSMOS Framework, cât și a tehnicilor de optimizare prezentate anterior. Prezentăm o abordare pentru integrarea calculului paralel în curriculumul Universității Politehnica din București [3]. Unul dintre principalele motive pentru care am ales Python pentru dezvoltarea COSMOS este că are o curbă de învățare relativ mică și este accesibil studenților din întreaga lume in primii ani de studiu al programelor de învățământ superior. Un alt subiect de interes pentru calculul de înaltă performanță îl reprezintă tehnicile de optimizare, care reprezintă unul dintre subiectele de bază abordate de cursurile de licenţă, așa cum este descris în continuare. După cum se poate observa, toate aceste cunostinte dobandite intr-un program de licenţă sunt necesare atunci când se dezvoltă un proiect complex precum COSMOS Framework.

Având în vedere evoluția actuală a industriei IT, calculul paralel și distribuit este văzut ca un subiect esențial pentru orice lucrator in domeniul tehnologiei informatiei si calculatoarelor. Universitatea „Politehnica” din București este una dintre cele mai vechi și mai prestigioase școli de inginerie din România. În ultimii 20 de ani, Departamentul de Inginerie si Ştiinţa Calculatoarelor a conferit o importanță deosebită curriculei PDC. Importanța sistemelor paralele și distribuite, precum și o distincție între sistemele paralele și cele distribuite este discutată în [36] [38], iar abordarea oferită de UPB este în concordanță cu viziunea prezentată aici, întrucât programa noastră conține deja diferite cursuri pentru sisteme distribuite și paralele. Majoritatea cursurilor din aceasta arie sunt predate în primii trei ani de licenta și ating un public larg de aproximativ 400-500 de studenți pe an. Prelegeri similare sunt oferite în întreaga lume de diferite grupuri CS din Tennessee [21], Cadiz [47] sau Cluj-Napoca [47].

În programul de licență al UPB, putem găsi trei cursuri principale în care sunt prezentate studenților problemele de calcul paralel și distribuit, și anume algoritmii paraleli și distribuiti (APD), arhitectura sistemelor informatice (ASC) și arhitecturile de procesare paralelă (APP). În această contributie ne concentram pe cursurile Arhitectura sistemelor informatice și Arhitecturile de procesare paralelă. Numărul de studenți care urmează cursurile APD și ASC variază de la 350 la 450 în fiecare an - fiind obligatorii pentru toți studenții înscriși la programul de licenta mentionat mai sus. Cursul APP adună de la 130 la 150 de studenți, la specializarea Sisteme Avansate de Calculatoare a Programului nostru de licență în Calculatoare.

Echipa din Departamentul de Calculatoare și Inginerie al UPB se străduiește să îmbunătățească prezentarea conceptelor PDC în programele sale de licență. În cursuri, studenții sunt predate aspectele generale de arhitectură și design ale PDC, în timp ce în activitățile practice explorează diverse abordări software cele mai potrivite pentru a ilustra aceste concepte

generale. Exerciţiile din laboratoare și temele pentru acasă sunt apoi menite să verifice dacă abilitățile relevante dorite au fost învățate de către studenții noștri. În acest articol, schițăm conținutul prelegerilor, procesul de evaluare a studenților, precum și lecțiile învățate de-a lungul timpului și îmbunătățirile pe care le-am introdus în conținutul și abordarea noastră. Industria IT manifestă un interes deosebit pentru absolvenții noștri, iar abilitățile lor PDC sunt foarte apreciate. Acest lucru se datorează și faptului că am continuat să ne dezvoltăm materialele și metodele în mod constant, pe măsură ce apar noi tehnologii. În același timp, însă, ne propunem ca studenții noștri să aibă o înțelegere fundamentală a modului în care funcționează arhitecturile de procesare paralelă și distribuită, atât din perspectiva hardware, cât și a software-ului. Pe măsură ce noi arhitecturi apar în mod continuu, conduse acum de domenii emergente precum AI sau IoT, blocurile esențiale rămân aceleași – iar PDC este unul dintre aceste blocuri.

Ne adaptăm în mod constant curriculum-ul pe măsură ce industria evoluează. O direcție interesantă sunt limbajele intermediare cross-API, cum ar fi SPIR, care oferă timpul de rulare de bază pentru mai multe API-uri, cum ar fi OpenCL, Vulkan, SyCL, OpenMP sau OpenACC. Mai mult, luăm în considerare adăugarea unei secțiuni care exemplifica interacțiunea OpenCL cu cadre populare de analiză a datelor și de învățare automată, cum ar fi Anaconda, prin utilizarea wrapper-ului PyOpenCL [16], conectând astfel laboratoarele CSA pe Python și OpenCL.

Capitolul 5

Concluzii

High Performance Computing (HPC) este unul dintre domeniile în care am văzut cele mai multe progrese în ultimii doi ani, atât în ceea ce privește hardware-ul, cât și software-ul. Fiecare domeniu de cercetare care necesită simulări numerice, modelare software a sistemelor, optimizări numerice sau capacități de calcul a mai mult de o mașină, este cel mai probabil legat de HPC și va beneficia foarte mult de cercetarea și banii investiți in acest domeniu. Principalele contributii si rezultate pe care le-am obtinut sunt în următoarele domenii ale ingineriei software și domeniilor de scriere computațională:

- Optimizari numerice (publicaţii [40], [33], [41])
- Aplicaţii de simulare (publicaţie [33], contribuţii la proiectul open-source CORSIKA [11]))
- Task scheduling (publicaţii [40], [33])
- Tehnici de paralelizare (publicaţie [40], contribuţii la proiectul open-source CERN Athena Framework [5])
- Curricula academică (publicaţie [3])

Această secțiune prezintă rezultatele, contribuțiile pe care le-am adus prin această teză cat şi activitatea viitoare în contextul COSMOS.

5.1 Contribuţiile tezei

COSMOS Framework a fost dezvoltat pentru a face o legatura intre domeniile de simulare și optimizare cu cel al stiintei calculatoarelor. După cum am prezentat în publicația noastră **COSMOS - Framework for Combining Optimization and Simulation Software**, propunem un framework general pentru integrarea aplicatiilor de simulare cu software de optimizare în mediile HPC. Am proiectat soluția noastră pentru a oferi o interfață comună

aplicațiilor existente (atât simulări numerice, cât și pachete de optimizare), astfel încât utilizatorul să poată crea pipeline de sarcini și bucle de optimizare doar prin descrierea lanțului de aplicații într-un fișier JSON.

O contribuție importantă pe care o avem este în domeniul programării sarcinilor: algoritmi implementați precum Min-Min, Max-Min și WorkQueue [22] care folosesc metrici de încărcare a mașinii și de caracteristicile sarcinilor ce sunt programate a spre executie. Am reușit acest lucru prin implementarea unui modul independent în framework-ul care monitorizează utilizarea resurselor live pentru fiecare mașină și oferă, de asemenea, timpul estimat de finalizare pentru sarcinile care rulează. De asemenea, arătăm impactul programării atât în medii simulate, cât și în aplicațiile din lumea reală în general, prezentăm arhitectura cadrului și oferim suport cu privire la modul în care COSMOS poate fi extins pentru a integra pachete care nu sunt încă acceptate. Din punct de vedere al simulării, am integrat aplicatii utilizate pe scară largă, cum ar fi LAMMPS, GROMACS și CORSIKA. Pentru fiecare dintre aceste aplicații de simulare numerică am scris interfețele de comunicare cu COSMOS și integrarea cu metode de optimizare și pachete externe.

Pachetul de optimizare CORSIKA combină metode de optimizare numerică pentru extragerea parametrilor de intrare și reglarea lor în consecință. Folosind metadate specifice aplicației, am împărțit sarcina de simulare în mai multe altele mai mici și am eliminat calculele redundante. Procedând astfel, am reușit să obținem o accelerare de 10,6 ori (pe serverele x86-64) și de 7,2 ori (pe computerul arm64 singleboard) față de rularea în serie. De asemenea, subliniem importanța eficienței energetice în calculul nostru și capacitatea de a executa pe arhitecturi eterogene fără a fi nevoie să schimbați nimic în codul aplicației. Deoarece cadrul este scris în Python, putem folosi unele dintre caracteristicile limbajului pentru a executa cu ușurință metode de optimizare sau pentru a face post/preprocesare a datelor pe mașinile gazdă după nume. Acest lucru ne permite să distribuim sarcina pe agenții broker în loc să ne bazăm pe un singur nod de controler.

Suportul pe care l-am adăugat în COSMOS pentru LAMMPS ne permite să avem *control la nivel de atom* asupra datelor de intrare de simulare. Putem genera structuri atomice complexe de la zero sau le putem modifica pe cele existente pentru a ne îndeplini nevoile sau a simula modele specifice (de exemplu, reordonarea listei de atomi pentru a urma curba Hilbert). Am folosit framework-ul pentru a genera nanofire de oxid de zinc de diferite dimensiuni pentru a simula absorbția hidrogenului în acest tip de structuri. O altă contribuție pe care am avut-o a fost integrarea unui optimizator extern numit Tool4LAMMPS, folosit mai ales pentru controlul parametrilor de simulare și închiderea buclei de optimizare.

Oferim, de asemenea, o soluție pentru una dintre cele mai abordate probleme atunci când vine vorba de proiectarea unei simulări: care este configurația hardware optimă pe care o

puteți folosi fie pentru a maximiza eficiența, fie pentru a minimiza timpul de executie. De data aceasta, nu ne interesează controlul parametrilor de simulare, ci configurația hardware de bază: numărul de fire de executie și procese, numărul de GPU și numărul total de servere. Am folosit COSMOS pentru a programa sarcinile de simulare GROMACS și pentru a analiza automat performanța fiecărei sarcini. După ce avem rezultatele pentru un numar de configurații reprezentative, oferim automat cei mai buni candidați atât pentru utilizarea optimă a resurselor, cât și pentru cel mai rapid timp de execuție. Folosind datele culese din simulări pe serverele noastre, oferim și câteva indicii despre configurația pe care s-ar putea alege și de la furnizorii de servicii cloud publice (Amazon, Microsoft si Google).

O altă colaborare unde am obtinut rezultate interesante a fost în procesarea imaginilor medicale folosint algoritmi de invatare automata si inteligenta artificiale. Ne-am concentrat pe analiza performanței unei aplicații de procesare a imaginilor in bloc, care utilizează un model de execuție intensiv de date, dar nu în sensul tradițional de procesare a imaginilor de rezoluţii mari. Aplicațiile cu randament ridicat mută accentul de la rulări individuale la ansamblu prin analizarea unui set de date uriaș de imagini de dimensiuni mici care trebuie să treacă printr-o serie de algoritmi de procesare și a căror debit trebuie optimizat. În acest scop, ne-am uitat la SLANTbrainSeg - Deep Whole Brain High-Resolution Segmentation - o secvenţa de algorithmi de segmentare a imaginilor de rezonanță magnetică structurală ale creierului, care este esențială pentru înțelegerea relațiilor neuroanatomo-funcționale. Analizând informațiile de profilare pentru a înțelege care sunt blocajele și folosind tehnici de programare adaptate pentru acest tip de aplicație, am reușit să maximizăm volumul de procesarea a loturilor de RMN. Utilizând abordarea noastră, scalam orizontal volumul de lucru la orice configurație hardware a serverelor pe care o avem.

În colaborarea noastră cu CERN, am lucrat la procesarea unor cantități mari de date obținute de la Large Hadron Collider (LHC). Aceste date sunt obținute din diverse experimente, colectate de un număr mare de subsisteme LHC, iar munca noastră s-a limitat strict la analizarea evenimentelor care au loc în sistemul Calorimetru. Am obținut rezultate bune de performanță, reușind să descarcăm sarcinile de lucru care erau specifice procesorului pentru GPU-urile NVidia folosind CUDA.

Ca parte a contribuției noastre, am demonstrat că framework-ul COSMOS poate fi utilizat pentru a optimiza algoritmii de căutare 3D. Această lucrare reprezintă un mare pas înainte pentru urmărirea speciilor pe cale de dispariție, monitorizarea vegetației în agricultura de precizie sau îmbunătățirea capacităților de căutare a sistemelor CCTV. Folosind algoritmii noștri de căutare în spațiu 3D, se poate reduce considerabil atât timpul de căutare, cât și puterea de calcul necesară pentru acest tip de sarcină. Având acest tip de flux de lucru integrat în cadrul nostru, demonstrăm un grad ridicat de flexibilitate pentru COSMOS.

Unul dintre subiectele noastre principale de interes este menținerea la zi a curriculumului de calcul paralel și distribuit. În programul de licență al Universitatţii Politehnica din Bucuresti, putem găsi trei cursuri principale în care sunt prezentate studenților problemele de calcul paralel și distribuit, și anume Algoritmii Paraleli și Distribuiti (APD), Arhitectura Sistemelor de Calcul (ASC) și Arhitecturile si Procesari Paralele(APP). În aceste cursuri, studenților le sunt predate aspectele generale de arhitectură și design ale sistemelor paralele si distribuite, în timp ce în activitățile practice explorează diverse abordări software cele mai potrivite pentru a ilustra aceste concepte generale. Laboratoarele practice și temele pentru acasă sunt apoi menite să verifice dacă abilitățile relevante dorite au fost învățate de către studenții noștri. Prezintăm conținutul prelegerilor, procesul de evaluare a elevilor, precum și lecțiile învățate de-a lungul timpului și îmbunătățirile pe care le-am introdus în conținutul și abordarea noastră. Industria IT manifestă un interes deosebit pentru absolvenții noștri, iar abilitățile lor sunt foarte apreciate. Acest lucru se datorează și faptului că am continuat să ne dezvoltăm materialele și metodele în mod constant, pe măsură ce apar noi tehnologii. În același timp, însă, ne propunem ca studenții noștri să aibă o înțelegere fundamentală a modului în care funcționează arhitecturile de procesare paralelă și distribuită, atât din perspectiva hardware, cât și a software-ului. Pe măsură ce noi arhitecturi apar în mod continuu, conduse acum de domenii emergente, cum ar fi AI sau IoT, blocurile esențiale rămân aceleași - iar PDC este unul dintre aceste blocuri.

Publicaţii

- **Cosmin-Gabriel Samoilă**, Emil-Ioan Sluşanschi. COSMOS - Framework for Combining Optimization and Simulation Software. 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, Publisher: IEEE, ISBN: 978-1-7281-2332-5

- Sebastian Muraru, **Cosmin-Gabriel Samoilă**, Emil-Ioan Sluşanschi, Jorge S. Burns, Mariana Ionita. Molecular Dynamics Simulations of DNA Adsorption on Graphene Oxide and Reduced Graphene Oxide-PEG-NH_2 in the Presence of Mg^{2+} and Cl^{-} ions. Coatings 2020, Publisher: MDPI, ISSN: 2079-6412, DOI: 10.3390/coatings10030289

- Carabaş Mihai, Draghici Adriana, Lupescu Grigore, **Cosmin-Gabriel Samoilă**, Emil-Ioan Sluşanschi. Integrating Parallel Computing in the Curriculum of the University Politehnica of Bucharest: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers

- **Cosmin-Gabriel Samoilă**, Damian Dinoiu, Emil-Ioan Sluşanschi. COSMOS Framework: Simulation and Optimization of 3D Search, U.P.B Scientific Bulletin - Series C, Volume 85, Issue 1, 2023, Bucharest, Romania, ISSN:2286-3540

Draft tehnic

- Maria Predescu, **Cosmin-Gabriel Samoilă**, Ana Găinaru, Emil-Ioan Sluşanschi, High-Throughput Computing on HPC: a Case Study of Medical Image Processing Applications

- **Cosmin-Gabriel Samoilă**, Nuno dos Santos Fernandes, Patricia Conde Muíño, Emil-Ioan Sluşanschi. Topological Calorimeter Clustering on GP-GPU at ATLAS using CUDA. 23rd IEEE Real Time Conference - Trigger Systems, CERN

5.2 Activităţi viitoare

Din perspectiva COSMOS, ne propunem să oferim o integrare cat mai strânsă cu alte pachete de simulare externe și software de optimizare. Ne dăm seama că nu va fi posibilă furnizarea de suport pentru o listă exhaustivă de aplicații, dar ne străduim să oferim îndrumări altor persoane care sunt dispuse să colaboreze la proiectul nostru open source. Pe lângă aceasta, vom oferi integrarea cu SLANT în viitorul apropiat, astfel încât să putem beneficia de toate caracteristicile pe care framework-ul le oferă în ceea ce privește programarea sarcinilor și echilibrarea incarcarii pe sistemele de calcul. Din punct de vedere academic, ne adaptăm constant curriculumul pe măsură ce industria evoluează. O direcție interesantă sunt limbajele intermediare cross-API, cum ar fi SPIR, care oferă timpul de rulare de bază pentru mai multe API-uri, cum ar fi OpenCL, Vulkan, SyCL, OpenMP sau OpenACC. Mai mult, luăm în considerare adăugarea unei secțiuni care exemplifica interacțiunea OpenCL cu cadre populare de analiză a datelor și de învățare automată, cum ar fi Anaconda, prin utilizarea wrapper-ului PyOpenCL, conectând astfel laboratoarele ASC pe Python și OpenCL.

Bibliografie

[1] Cosmin-Gabriel Samoilă. COSMOS Framework. https://github.com/gabrielcsmo/COSMOS-Framework.git. Accessed 26 Feb 2023.

[2] H.M. Bücker A. Rasch. Efcoss: An interactive environment facilitating optimal experimental design. http://doi.acm.org/10.1145/1731022.1731023, April 2010. ACM Trans. Math. Softw. 37, 2, Article 13 (April 2010), 37 pages.

[3] Mihai Carabaş, Adriana Drăghici, Grigore Lupescu, Cosmin-Gabriel Samoilă, and Emil-Ioan Sluşanschi. Integrating parallel computing in the curriculum of the university politehnica of bucharest. In *Euro-Par 2018: Parallel Processing Workshops: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers 24*, pages 222–234. Springer, 2019.

[4] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[5] CERN. Athena framework. https://gitlab.cern.ch/atlas/athena.

[6] B. Lang C.H. Bischof, H.M. Bücker. An interactive environment for supporting the transition from simulation to optimization. Sci. Prog. 11, 4, 263–272, 2003.

[7] P.A. Cherenkov. Visible emission of clean liquids by action of γ radiation. http://iopscience.iop.org/article/10.1070/PU2007v050n04ABEH006238/pdf, 2007.

[8] Intel Co. Intel 64 and IA-32 architectures optimization reference manual. https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf.

[9] The ATLAS Collaboration. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3(08):S08003, aug 2008.

[10] The ATLAS Collaboration. Topological cell clustering in the ATLAS calorimeters and its performance in LHC run 1. *The European Physical Journal C*, 77(7), jul 2017.

[11] Air shower physics - corsika. https://gitlab.iap.kit.edu/AirShowerPhysics/corsika. Accessed 05 Mar 2023.

[12] Marco Dantas et al. Monarch: Hierarchical storage management for deep learning frameworks. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 657–663, 2021.

[13] GROMACS development team. GROMACS User Guide. https://manual.gromacs.org/current/user-guide/index.html. Accessed 26 Feb 2023.

[14] J.N. Capdevielle D.Heck, J. Knapp. Corsika: A monte carlo code to simulate extensive air showers. https://web.ikp.kit.edu/corsika/physics_description/corsika_phys.pdf, 1998.

[15] T. Pierog D.Heck. Extensive air shower simulation with corsika: A user's guide. https://web.ikp.kit.edu/corsika/usersguide/usersguide.pdf, August 2016. Institut fur Kernphysik.

[16] Massimo Di Pierro. Portable parallel programs with python and opencl. *Computing in Science & Engineering*, 16(1):34–40, 2013.

[17] Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441–1459, Apr 2015.

[18] Diogo R. Ferreira. Applications of deep learning to nuclear fusion research, 2018.

[19] Ana Gainaru et al. Understanding the impact of data staging for coupled scientific workflows. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4134–4147, 2022.

[20] Mu Gao et al. High-performance deep learning toolbox for genome-scale prediction of protein structure and function. In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*, pages 46–57, 2021.

[21] Sheikh Ghafoor, David W Brown, and Mike Rogers. Integrating parallel computing in introductory programming classes: an experience and lessons learned. In *Euro-Par 2017: Parallel Processing Workshops: Euro-Par 2017 International Workshops, Santiago de Compostela, Spain, August 28-29, 2017, Revised Selected Papers 23*, pages 216–226. Springer, 2018.

[22] V. Sn*á*šel H. Izakian, A. Abraham. Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments. Neural Network World 6/09, 695-710, 2009.

[23] Yuankai Huo et al. Towards portable large-scale image processing with high-performance computing. *Journal of Digital Imaging*, 31(3):304–314, Jun 2018.

[24] J.J. Moore J. Czyzyk, M.P. Mensier. The neos server. IEEE Computational Science & Engineering 5, 3, 68–75, 1998.

[25] Alexander Keller, Carsten Wächter, and Nikolaus Binder. Rendering along the hilbert curve. In *Advances in Modeling and Simulation: Festschrift for Pierre L'Ecuyer*, pages 319–332. Springer, 2022.

[26] Julian Keupp. Methods for fast thermal equilibration in molecular dynamics simulations, 2015. JSC Guest Student Program.

[27] Anthony Kremin et al. Rapid processing of astronomical data for the dark energy spectroscopic instrument. In *2020 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, pages 1–9, 2020.

[28] Sandia National Laboratories. LAMMPS Benchmarks. http://lammps.sandia.gov/bench.html.

[29] Oak Ridge National Laboratory. Summit. https://docs.olcf.ornl.gov/systems/summit_user_guide.html. [Online; accessed September-2022].

[30] Lammps benchmarks. http://lammps.sandia.gov/bench.html. Accessed 20 Mar 2023.

[31] Lammps data format. https://docs.lammps.org/99/data_format.html. Accessed 20 Mar 2023.

[32] Julie Linchant, Jonathan Lisein, Jean Semeki, Philippe Lejeune, and Cédric Vermeulen. Are unmanned aircraft systems (uas s) the future of wildlife monitoring? a review of accomplishments and challenges. *Mammal Review*, 45(4):239–252, 2015.

[33] Sebastian Muraru, Cosmin Gabriel Samoila, Emil I. Slusanschi, Jorge S. Burns, and Mariana Ionita. Molecular dynamics simulations of dna adsorption on graphene oxide and reduced graphene oxide-peg-nh2 in the presence of mg2+ and cl- ions. *Coatings*, 10(3), 2020.

[34] Numpy hilbert curve. https://pypi.org/project/numpy-hilbert-curve/. Accessed 20 Mar 2023.

[35] Yosuke Oyama et al. The case for strong scaling in deep learning: Training large 3d cnns with hybrid parallelism. *CoRR*, abs/2007.12856, 2020.

[36] Marcin Paprzycki, Ryszard Wasniowski, and Janusz Zalewski. Parallel and distributed computing education: A software engineering approach. In Rosalind L. Ibrahim, editor, *Software Engineering Education*, pages 187–204, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[37] Marija Popović, Teresa Vidal-Calleja, Gregory Hitz, Inkyu Sa, Roland Siegwart, and Juan Nieto. Multiresolution mapping and informative path planning for uav-based terrain monitoring. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1382–1388. IEEE, 2017.

[38] Michel Raynal. Parallel computing vs. distributed computing: a great confusion?(position paper). In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*, pages 41–53. Springer, 2015.

[39] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[40] Cosmin-Gabriel Samoila and Emil-Ioan Slusanschi. Cosmos-framework for combining optimization and simulation software. In *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, pages 162–169. IEEE, 2019.

[41] Cosmin-Gabriel Samoil˘ a, Damian Dinoiu, and Emil-Ioan Slu¸sanschi. Cosmos framework: Simulation and optimization of 3D search. *Scientific Buletin of Universitatea Politehnica of Bucharest, C Series*, 85, 2023.

[42] Klaus Schittkowski. *EASY-FIT: A software system for data fitting in dynamical systems*, volume 23. Springer, 03 2002.

[43] Fabian Schmidt. Corsika shower images: Iron showers. https://www-zeuthen.desy.de/~jknapp/fs/iron-showers.html. University of Leeds, UK.

[44] Garrick Staples. Torque resource manager. SC '06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Article No. 8, ISBN:0-7695-2700-0, November 2006. Tampa, Florida.

[45] Georgi Stoychev. Tools for preparation and analysis of molecular dynamics simulations, 2014. JSC Guest Student Program.

[46] A. Agrawal T. Goyal, A. Singh. Cloudsim: simulatior for cloud computing infrastructure and modeling. International Conference on modelling, optimisation and computing (ICMOC-2012), 2012.

[47] Antonio J Tomeu-Hardasmal, Alberto G Salguero, and Manuel I Capel. Integration of ict in concurrent and parallel programming lectures. In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*, pages 114–124. Springer, 2015.

[48] Yunxi Xiong et al. Reproducibility evaluation of SLANT whole brain segmentation across clinical magnetic resonance imaging protocols. In Elsa D. Angelini and Bennett A. Landman, editors, *Medical Imaging 2019: Image Processing*, volume 10949, pages 729 – 736. International Society for Optics and Photonics, SPIE, 2019.

[49] Yue Zhu et al. Entropy-aware i/o pipelining for large-scale deep learning on hpc systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 145–156, 2018.