UNIVERSITATEA "ŞTEFAN CEL MARE" SUCEAVA
FACULTATEA DE INGINERIE ELECTRICĂ ŞI ŞTIINŢA CALCULATOARELOR
DOMENIUL: CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

TEZĂ DE DOCTORAT

CONTRIBUŢII LA PARALELIZAREA ALGORITMILOR DE RECUNOASTERE A FORMELOR

Ing. ISAC Ştefania – Iuliana (ŞOIMAN)

Conducător ştiinţific,

prof. univ. dr. ing. Ştefan-Gheorghe PENTIUC

Suceava 2015

Universitatea
Ştefan cel Mare
Suceava

DECIZIA D 66 / 7.05 2015
privind constituirea comisiei pentru susţinerea publică a unei teze de doctorat

În conformitate cu prevederile H.G. nr. 26 din 14.01.2015 privind organizarea şi funcţionarea Ministerului Educaţiei şi Cercetării Ştiinţifice, Anexa 3, punctul 39, prin care se instituţionalizează Universitatea Stefan cel Mare din Suceava;
În conformitate cu HG 681 din 29.06.2011 - Codul Studiilor universitare de doctorat;
Având în vedere aprobarea conducerii Universităţii;

Rectorul Universităţii "Ştefan cel Mare" din Suceava emite prezenta decizie:

Art. 1 - Se numeşte comisia de doctorat pentru susţinerea publică a tezei de doctorat în data de **22.05.2015**, cu titlul: **„CONTRIBUŢII LA PARALELIZAREA ALGORITMILOR DE RECUNOAŞTERE A FORMELOR"**,

elaborată de **ISAC A. Ştefania Iuliana (ŞOIMAN)**,
înmatriculată la data de 1.10.2009,
forma de învăţământ: fără frecvenţă, cu taxă
DOMENIUL: **Calculatoare şi tehnologia informaţiei**

PREŞEDINTE:
Prof. univ. dr. ing. Adrian GRAUR, reprezentant IOSUD, Universitatea „Ştefan cel Mare" din Suceava;

CONDUCĂTOR ŞTIINŢIFIC:
Prof. univ. dr. ing. Ştefan Gheorghe PENTIUC, Universitatea „Ştefan cel Mare" din Suceava;

REFERENŢI:
1. Prof. univ. dr. ing. Sergiu NEDEVSCHI, Universitatea Tehnică Cluj Napoca;
2. Prof. univ. dr. ing. Petru CAŞCAVAL, Universitatea Tehnică „Gh. Asachi" Iaşi;
3. Prof. univ. dr. ing. Mitică CRAUS, Universitatea Tehnică „Gh. Asachi" Iaşi;
4. Prof. univ.dr. ing. Vasile Gheorghiţă GĂITAN, Universitatea „Ştefan cel Mare" din Suceava.

Art. 2. – Direcţia Economică şi Resurse Umane şi serviciul Secretariat - doctorat vor duce la îndeplinire dispoziţiile prezentei decizii.

RECTOR,
Prof. univ. dr. ing. Valentin POPA

SECRETAR ŞEF UNIV.,
Jurist Maria MUSCĂ

Universitatea "Ştefan cel Mare" Suceava, Str. Universităţii nr. 13, 720 229 Suceava, România
Tel: +40 230 520 081, Fax: +40 230 520 080, Web: www.usv.ro

CUPRINS

C a p i t o l u l 1

Introducere

1.1 Motivație

Odată cu evoluția tehnologiilor informaționale a crescut posibilitatea stocării unei cantități importante de informații digitale. Bazele de date masive sunt utilizate în cele mai diverse domenii: educație, știință sau aplicații comerciale. Simultan a apărut și necesitatea de a le prelucra în vedearea extragerii de noi cunoștințe, însă această prelucrare este mare consumatoare de timp de calcul. Algoritmii populari de clasificare supravegheată, bazați pe Hidden Markov Models (HMM) și Support Vector Machine (SVM), necesită resurse mari de calcul atunci când procesează un volum mare de date. În cazul problemelor de recunoaştere avem un set de modele Markov ascunse şi o secvenţă de observaţii. Pentru a decide care model a generat secvenţa de observaţii, se va evalua probabilitatea acesteia în fiecare model, şi se va alege acel model, care a furnizat probabilitatea maximă. Modelele Markov cu un număr mare de stări (Beal et al., 2001), prezintă dezavantajul utilizarii unei cantități mari de memorie și timpul mare de execuție, ceea ce face ca cercetările întreprinse să fie necesare și actuale în domeniul recunoașterii vorbirii (Taylor, 2006), analiza traficului web (Felzenszwalb et al., 2003) sau în problemele care implică dimensiuni mari ale vocabularului LVCSR - Large Vocabulary Continuous Speech Recognition (Silingas and Telksnys, 2004). Pentru aceste aplicații, modelele Markov ascunse sunt construite cu sute sau chiar mii de stări. Calculul probabilității fiecărui model implică construirea matricei probabilităților de tranziție dintre stările modelului, ceea ce presupune manipularea unei matrici de dimensiuni foarte mari având dezavantajul utilizarii unei cantități mari de memorie și timpi mari de execuție. Din aceste considerente, astfel de aplicații devin pretabile pentru dezvoltarea lor pe mașini paralele, timpul de calcul și resursele folosite reducându-se semnificativ. Acest proiect de cercetare urmăreşte proiectarea și evaluarea algoritmilor paraleli de clasificare a volumelor mari de date, bazați pe HMM și SVM, folosind resurse de calcul de înaltă performanţă.

Dezvoltarea rapidă a resurselor hardware, memorii mai mari combinate cu procesoare mai rapide, a implicat progresul aplicațiilor software. Astfel, a apărut o colecție de algoritmi și implementări în diverse domenii unde este necesară o putere mare de calcul. Clusterele de calculatoare, o combinație de calculatoare interconectate prin rețele de mare viteză, oferă o putere mare de calcul prin executarea task-urilor de prelucrare pe mai multe procesoare simultan. Aplicațiile ce rulează pe clusterul USV Roadrunner (IBM) utilizează arhitectura hibridă a sistemului Cell BE formată din două tipuri de procesoare. Această arhitectură hibridă a făcut ca alegerea strategiilor optime de paralelizare și distribuire a datelor să fie dificilă. În această teză sunt studiate instrumentele necesare și condițiile în care se poate realiza dezvoltarea

aplicațiilor de recunoaștere a formelor pe procesoare cu mai multe nuclee (folosind arhitectura sistemului Cell BE). Utilizarea arhitecturii paralele aduce noi probleme, cum ar fi distribuția task-urilor, distribuirea datelor și utilizarea eficientă a ierarhiilor de memorie.

Scopul tezei este dezvoltarea algoritmului paralel Forward, pe baza modelului Markov ascuns, și optimizarea algoritmului Sequential Minimal Optimization pe baza vectorilor suport, ce vor folosi potențialul și avantajele arhitecturii clusterului USV Roadrunner (IBM) din Laboratorul de Calcul de Înaltă Performanță al Universității „Ștefan cel Mare" din Suceava. Clusterul USV Roadrunner (IBM) este construit cu procesoare PowerXCell 8i, cu o performanță de 6.53 TFlops și are o arhitectură asemănătoare cu cea a clusterului Roadrunner, proiectat de firma IBM pentru Laboratorul National Los Almos (LANL) din New Mexico, SUA. Supercalculatorul Roadrunner este primul calculator care a depășit bariera de 1 petaflops în anul 2008 (Barker et al., 2008).

1.2 Obiectivele și structura tezei

Obiectivul principal al tezei este concepția, proiectarea şi implementarea algoritmilor paraleli de clasificare folosind calculul de înaltă performanţă. Cercetarea se va face folosind cluster-ul de calculatoare USV Roadrunner (IBM), cu o putere de calcul sporită.

În realizarea acestui obiectiv am parcurs următoarele etape:

– analiza stadiului actual al cercetării în domeniul clasificatoarelor bazate pe tehnica vectorilor suport şi a Modelului Markov Ascuns;

– studiul arhitecturii sistemului de calcul de înaltă performanță USV Roadrunner (IBM) și modul în care se va face evaluarea algoritmilor paraleli de recunoaștere a formelor;

– utilizarea conceptelor şi a metodelor din domeniul calculului de înaltă performanță *HPC* (High Performance Computer) în probleme ce implică clasificarea datelor;

– proiectarea și implementarea tehnicilor de paralelizare a algoritmului Forward HMM pe două nivele;

– evaluarea algoritmului de clasificare SVM paralelizat pe două nivele și analiza rezultatelor experimentale obținute;

– aplicarea unor tehnici de reducere a timpului de antrenare a algoritmului paralel de clasificare SVM.

Teza este structurată sub forma a șapte capitole și bibliografie. Primul capitol cuprinde obiectivul principal și structura tezei, precum și prezentarea stadiului actual al cercetărilor în domeniul abordat.

În al doilea capitol sunt prezentate o serie de noțiuni generale din domeniul recunoașterii supravegheate și nesupravegheate a formelor, sunt descrise aspecte teoretice privind algoritmii de recunoaștere a formelor puși în discuție: HMM și SVM. Pentru început se face o comparare a

metodelor generative și discriminante de învățare supravegheată. Parametrii modelului probabilistic Markov ce stau la baza algoritmilor HMM sunt prezentați în acest capitol împreună cu soluții pentru rezolvarea problemelor fundamentale ale modelelor. SVM ca o metodă de clasificare supravegheată, optimizează o frontieră de decizie ce separă formele în două clase. Cele două metode de clasificare a datelor propuse necesită resurse de calcul intensiv atunci când numărul de stări ale modelului HMM este foarte mare și lungimea secvenței de observații este mare. În clasificarea binară, necesarul de memorie pentru antrenarea clasificatorului este foarte ridicat atunci când se lucrează cu seturi mari de date caracterizate printr-un număr mare de forme și/sau caracteristici.

Pentru proiectarea și implementarea algoritmilor paraleli pe clusterul de calculatoare USV Roadrunner (IBM), în capitolul III a fost studiată arhitectura sistemului paralel de calcul precum și o serie de tehnici de proiectare a algoritmilor paraleli. Capitolul prezintă pe lângă modalitățile de calcul a performanțelor aplicațiilor proiectate pe sisteme paralele de calcul și infrastuctura software a clusterului USV Roadrunner (IBM).

Capitolele IV și V descriu contribuțiile de natură teoretică. În aceste capitole sunt analizate tehnicile de paralelizare a algoritmilor HMM și SVM.

În capitolul IV sunt definite două metode de paralelizare multinivel a algoritmului Forward pe clusterul USV Roadrunner (IBM).

Capitolul V este dedicat prezentării algoritmilor de clasificare construiți cu ajutorul vectorilor suport. Algoritmul Sequential Minimal Optimization (SMO) face obiectul de studiu în acest capitol. LibSVM_CBE, ce pornește de la biblioteca LIBSVM, este o implementare paralelă a algoritmului SMO, utilizat pentru antrenarea clasificatorului SVM. Programul este executat pe procesoarele PowerXCell8i cu activarea a 16 nuclee *Synergistic Processor Element* (SPE). Datele de test folosite în antrenarea clasificatorului sunt analizate înainte de utilizarea lor în obținerea rezultatelor exerimentale a algoritmului paralel SMO.

În analiza algoritmului Forward s-au identificat punctele în care se poate face paralelizarea la nivel MPI și la nivelul nucleelor SPE. În capitolul Capitolul VI sunt prezentate rezultatele experimentale obținute la testarea algoritmilor paraleli ce au la bază modele Markov ascunse, propuși în capitolul IV. Prin testele efectuate a fost puse în evidență câștigul de viteză obținut prin activarea acceleratoarelor SPE a procesoarelor PowerXCell8i. Contribuțiile aduse în acest capitol au fost valorificate prin publicarea a două articole indexate ISI.

În ultimul capitol sunt prezentate concluziile, propunerile de direcții ulterioare și au fost evidențiate contribuțiile aduse de teză la rezolvarea problemelor propuse.

C a p i t o l u l 4

Paralelizarea algoritmilor de recunoaștere a formelor utilizând modelul HMM

4.1 Introducere

Modelele Markov ascunse sunt utilizate într-o varietate de probleme de recunoaștere a formelor, cum ar fi recunoașterea: vorbirii, a scrisului de mână, a gesturilor, a imaginilor și în domeniul bioinformaticii. Modelul matematic a fost dezvoltat de Baum și colegii săi în Princeton, la sfârșitul anilor 1960. J Baker este cunoscut pentru activitatea sa de pionierat în domeniul recunoașterii vorbirii cu HMM (Baker, 1975). Până la sfârșitul anilor 1970, HMM a rămas un model utilizat în recunoașterea vorbirii (ASR – Automat Speech Recognition), rezultatele nefiind diseminate. La mijlocului anilor '80, Rabiner și Juang, cercetători la Laboratoarele Bell, au publicat articole privind caracteristicile HMM și aplicațiile lor [4]. Tutorialul publicat de Rabiner la sfârșitul anilor 1980 (Rabiner, 1989), a depășit cu mult așteptările, el devenind extrem popular, fiind adaptat la o gamă largă de aplicații.

Modelul HMM de ordinul I pornește de la setul de N stări și secvența de observații de lungime T, fiecare observație are M posibile valori (simboluri observabile). Fiecare stare $q(t)$, are asociată probabilitățile de tranziție dintre stări a_{ij}, iar fiecare simbol o_t al secvenței de observații $O(t)$, are asociată o probabilitate $b_j(o_t)$de observare a simbolului din fiecare stare j. Conceptul de model Markov ascuns (HMM) este definit astfel după faptul că stecvența de stări, care generează date observabile, rămâne ascunsă.

Probabilitatea unei secvențe de observații, având dat un model $\lambda=(A,B,\pi)$, poate fi calculată cu algoritmul Forward. Algoritmul Forward (AF), aplică principiul programării dinamice, care este mult mai eficient decât algoritmul simplu de evaluare a probabilității unei secvențe de observații (Rabiner, 1989). Precizia cu care este calculată probabilitatea este mică pe măsură ce numărul de stări crește. În scopul evitării pierderii acesteia pot fi aplicate normalizări în procesul recursiv de calcul. În cazul de față, soluția preciziei de calcul a probabilității secvenței de observații este rezolvată prin folosirea logaritmului în reprezentarea probabilității (Lin and Dyer, 2010). Probabilitatea secvenței de observații se calculează recursiv, întreg procesul este consumator de timp și resurse de calcul pentru valori ale lui N și T mari.

Modelul probabilistic Markov cu stări ascunse poate fi utilizat în domeniul recunoașterii formelor pentru a afla clasa de apartenență a modelului, atunci când se dorește aflarea Modelului Markov Ascuns care a produs o anumită secvență de observații.

4.2 Tehnici de paralelizare a algoritmilor HMM

În lucrarea (Sand et al., 2010) autorii propun o implementare paralelă HMMlib, a algoritmilor HMM pe 16 noduri. Accelerarea obținută la testarea algoritmului Forward tinde spre valoarea 2 la tipul de data double și aproape 4 la tipul float pentru modele Markov cu mai mult de 100 de stări și 16 simboluri observabile iar lungimea secvenței de simboluri observabile 10000. Algoritmul propus a fost testat cu ajutorul aplicatiei OpenMP și instrucțiunilor SSE pe un sistem cu două calculatoare Intel quad-core, procesoare Xeon – 256kB, L2cache, 8MB L3 cache, 2.26 GHz şi 8GB RAM. HMMlib distribuie sarcinile la nivelul SPE-urilor (8 nuclee la fiecare CPU) și efectuează calcule cu tipuri de date numerice în virgulă mobilă pe 32 şi 64 biţi. A doua implementare, parredHMMlib, a autorilor (Nielsen and Sand, 2011) este practică pentru modele Markov cu spațiu mic N, al stărilor și lungimea secvenței de simboluri observabile T, mare. Algoritmul Forward paralel este de 6 ori mai rapid comparativ cu cel precedent din biblioteca HMMlib.

În articolul (Sand et al., 2013), sunt detaliate ultimele cercetări ale autorilor în domeniul bioinformaticii a celei mai importante aplicații – HMM. Performanțele implementării paralele a AF, zipHMMlib au fost înregistrate pe două procesoare Intel Sandy Bridge E5-2670 cu 8 nuclee, comparativ cu versiunile anterioare ale aceloraşi autori HMMlib și parredHMMlib.

O analiză comparativă realizată de (Meng and Ji, 2013) a algoritmului HMMER rulat pe diferite arhitecturi a sistemelor paralele de calcul arată potențialul diferitelor tehnologii hardware de accelerare. Cu soluția de accelerare a algoritmilor HMM propusă de (Lifshits et al., 2009) se obțin rezultate satisfăcătoare în practică. Sunt propuși patru algoritmi de accelerare a decodării și antrenării modelelelor HMM și implementările paralele ale acestora.

TEHNICI DE PARALELIZARE A ALGORITMULUI HMM PE UN SINGUR NIVEL

Paralelizarea datelor la calculatoarele cu memorie distribuită utilizează mesajele pentru transmiterea datelor. Prima metodă de paralelizare a algoritmului HMM utilizează paralelismul datelor prin distribuirea volumului datelor de intrare (secvențele de observații) la procesoarele PPE ale procesoarelor PowerXCell 8i. Pe fiecare procesor rulează algoritmul HMM pentru subsetul de date asignat.

În cadrul primei metode de paralelizare a AF se distribuie secvențele de observații pe fiecare procesor PPE, care va rula simultan AF pentru subsecvența asociată. După finalizarea calculelor este construită probabilitatea totală P (linia 10 a **Error! Reference source not found.**) prin însumarea probabilităților parțiale calculate pe fiecare procesor PPE. Pentru sincronizarea calculelor parțiale de pe fiecare procesor se va folosi funcția *MPI_barrier()*.

Prin împărțirea volumului de date ce trebuie procesat, complexitatea algoritmului se reduce la $O(N^2 * (nseq * len)/nproc)$, unde $nproc$ reprezintă numărul de procesoare PowerXCell8i.

4.3 Optimizarea algoritmilor HMM paraleli

Al doilea algoritm paralel Forward propune distribuirea calculului variabilei forward α fiecărui procesor SPE. Profitând de avantajul arhitecturii procesoarelor PowerXCell8i, prin activarea nucleelor acceleratoare SPE se poate crește performanța algoritmului, cu aproximativ două ordine de magnitudine (Arevalo et al., 2008). Fiecare nucelu SPE are 256 KB memorie locală folosită atât pentru date cât și pentru aplicație. Calculul variabilei *Forward*, necesită parametrii modelului HMM: *N*, *M*, *A*, *B* precum și secvența de observații. Pentru modele cu un număr mare de stări, capacitatea memoriei locale a fiecărui SPE este depășită. Astfel sunt necesare tehnici și strategii de programare paralelă pentru a putea folosi capacitatea limitată a resurselor procesoarelor SPE.

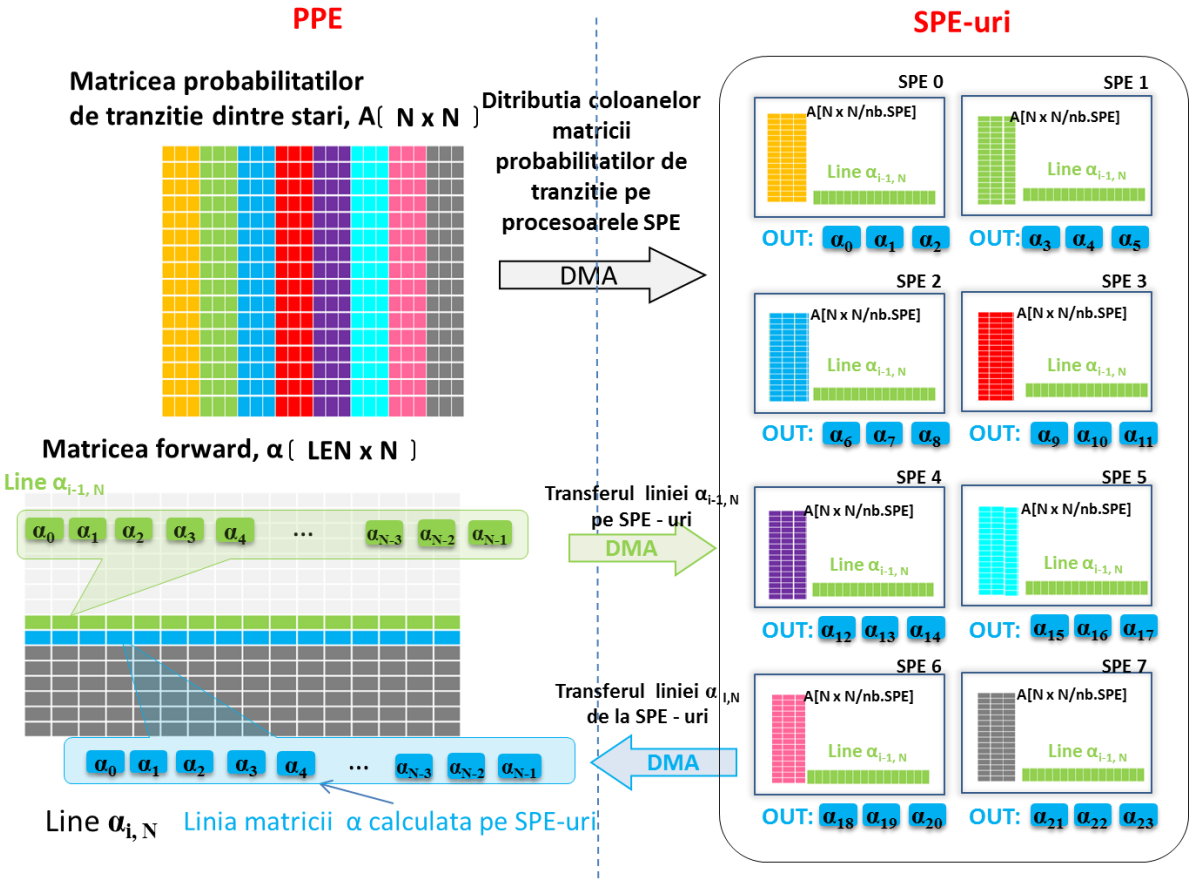

Figura 1 Calculul variabilei forward α pe procesoarele SPE (exemplu pentru un model HMM cu 24 stări)

În Figura 1 s-a exemplificat comunicațiea și transferul datelor dintre PPE și SPE pentru un model HMM cu 24 de stări. La nivelul procesoarelor PPE se construiește linia matricei α (Linieα_{i-1}), ce va fi transmisă ulterior acceleratoarelor SPE pentru calculul liniei următoare (Linieα_i). Procesul continuă pentru blocul de secvențe de observații asignate fiecărui procesor

PPE. Prima linie a matricii α este calculată pe procesoarele PPE în funcție de vectorul probabilităților inițiale (Ec.2.3.5) pentru fiecare stare, calculul celorlalte *len-1* linii efectuându-se pe cele 8 acceleratoarele SPE aferente unității PPE. Pentru modelul cu 24 de stări, fiecărui procesor SPE îi revine calculul unui bloc de trei valori a probabilității α (α_i α_{i+1} α_{i+2}), pentru cele trei coloane din matricea probabilităților de tranziție dintre stări, *A*, asociate. Fiecare bloc de coloane este colorat diferit după cum este distribuit SPE-ului. Calculul unei linii din matricea α, $Line_{\alpha i,N}$ este dependent de valoarea liniei precedente $Line_{\alpha i-1,N}$, ceea ce necesită actualizarea liniei din matricea *α* pe fiecare procesor PPE, după finalizarea construirii valorilor acesteia pe SPE-uri.

După finalizarea calculelor pe fiecare nucleu SPE, blocul de valori α_i α_{i+1} α_{i+2} se transferă prin DMA, urmând ca pe PPE să se construiască linia. La nivelul procesoarelor PPE se construiește linia matricei α_i ($Line_{\alpha i-1,N}$), ce va fi transmisă ulterior acceleratoarelor SPE pentru calculul liniei următoare ($Line_{\alpha i,N}$). Procesul continuă pentru blocul de secvențele de observații asignate fiecărui procesor PPE. Transferul datelor între PPE și SPE, transferul parametrilor modelului HMM de la PPE la SPE și transferul rezultatelor parțiale ale calculului probabilității Forward de la SPE la PPE, se realizează prin DMA.

4.4 Schema de paralelizare

Fiecare procesor SPE, la primirea comenzilor, realizează următoarele operaţii:

- transferă din memoria principală în memoria locală prin DMA structura de date ce conţine: numărul total de stări ale modelului HMM, numărul de observații, adresa matricei probabilităților simbolurilor observabile, numărul de secvențe de observații, lungimea secvenței, adresa vectorului unde sunt memorate secvențele de observații, adresa liniei din matricea α - Lineα, adresa coloanei din matricea probabilităților de tranziție dintre stări;
- transferă din memoria principală în memoria locală prin DMA, linia din matricea α;
- transferă din memoria principală în memoria locală prin DMA, blocul de coloane din matricea probabilităților de tranziție dintre stări, *A*;
- transferă din memoria locală prin DMA din memoria locala in memoria principala, blocul de valori αi calculate pentru stările asociate nucleului SPE;
- trimite prin mailbox un semnal ce arată că operaţiile au fost terminate şi valorile αi au fost transferate în memoria principală pentru a putea fi construite noile linii alfa pe PPE- uri.

Caclulul probabilității se face la nivelul acceleratoarelor SPE, procesoarele PPE colectează rezultatele parțiale și le prelucrează pentru obținterea rezultatelor finale.

Capitolul 5

Paralelizarea algoritmilor de clasificare utilizând modelul SVM

5.1 Introducere

Algoritmul de clasificare construit cu ajutorul vectorilor suport este aplicat cu succes în învățarea supravegheată pe mulțimi de date de dimensiuni mari (atât numărul formelor de clasificat cât și numărul de caracteristici ale acestora). Tehnicile de programare urmăresc îmbunătățirea acurateței clasificării formelor și reducerea timpului de antrenare a clasificatorului.

Unul din dezavantajele algoritmului SVM este cantitatea mare de memorie necesară pentru clasificarea seturilor mari de date. Motivul este rezolvarea problemei pătratice (QP) adică separarea vectorilor suport de restul datelor de antrenare. Două abordări de rezolvare a problemelor de optimizare a algoritmilor SVM au apărut în ultimii 10 ani, prima se referă la algoritmii care exploatează stuctura specială a problemei, în timp ce a doua se referă la diferite tehnici de stabilire a suprafeței de decizie ce separă formele în două clase. Din prima categorie algoritmii propuși de (Joachims, 1999) și (Chang and Lin, 2001) folosesc metoda împărțirii setului de date de antrenare (*decomposition technique*).

Dezvoltată începând cu anii 2000, biblioteca LibSVM [65] ajuns la versiunea 3.20 (noiembrie 2014). Detaliile implementării pachetului LibSVM (Chang and Lin, 2001) dovedesc faptul că biblioteca este un open source pentru probemele de optimizare, clasificare binară și multi-clasă, estimarea probabilității și selecția parametrilor modelului SVM.

5.2 Algoritmul SMO (Platt, 1999)

Algoritmul SMO împarte problema pătratică într-o serie de subprobleme ușor de gestionat. Cantitatea de memorie necesară algoritmului crește liniar cu dimnesiunea setului de date ceea ce permite lucrul cu seturi mari de date(Padron, 2007). Cea mai mare consumatoare de memorie este matricea Kernel, cu dimensiunea egală cu pătratul numărului formelor de antrenare. La fiecare pas, algoritmul SMO rezolvă analitic subproblema pătratică pentru cei doi multiplicatori și actualizează modelul SVM corespunzător. Algoritmul SMO, bazat pe metoda euristică, împarte problema pătratică QP în sub-probleme cu două dimensiuni, care pot fi rezolvate analitic.

Algoritmul SMO implică analiza a doi mutiplicatori Lagrange α_i la momentul de timp respectiv, fiind cea mai mică cantitate posibilă față de $\sum_{i=1}^{N} y_i \alpha_i$ din problema duală (Ecuația 5.1). Metoda de alegere a celor doi multiplicatori este euristică. Ideea principală a algoritmului este ajustarea a câte doi multiplicatori Lagrange și actualizarea corespunzătoare a modelului

SVM la fiecare pas. Fără memorarea matricii Kernel, algoritmul SMO scalează liniar sau pătratic *(~N* și ~$N^{2.2}$) cu dimensiunea setului de date de antrenare pentru diverse probleme.

5.3 Analiza seturilor de date

Necesarul de memorie pentru antrenarea clasificatorului este foarte ridicat atunci când se lucrează cu seturi mari de date (atât numărul formelor de clasificat cât și numărul de caracteristici ale acestora).

Pentru analiza performantelor algoritmilor paraleli SVM, de antrenare a clasificatorului, s-au folosit seturi de date din lumea reală, disponibile pe Internet [65-66]. Necesarul de memorie pentru generarea modelului după care se va face clasificarea cu SVM este foarte ridicat atunci când se lucrează cu seturi mari de date mari. Seturile binare de date (Tabelul 1) alese în evaluarea algoritmilor SVM paraleli propuși sunt: *web* (web page classification), *real-sim* (real vs simulated), *RCV1-binar*, *url* și *mnist8*-10k. Datele de antrenare pe baza cărora se face analiza algoritmilor SVM paraleli, sunt reprezentate prin numărul de forme și caracteristicile acestora (Tabelul 1).

Tabelul 1 Date de antrenare a clasificatorului SVM

Set date	Număr forme	Caracteristici
reuters corpus volume 1 – binary	20242	47236
web	49749	300
real vs simulated	72309	20958
url	20000	155216
mnist8-10k	10000	779

5.4 Algoritmi SVM paraleli pe un singur nivel

Unul din dezavantajele algoritmului SVM este necesarul mare de memorie și timpul mare de execuție pentru seturi de date foarte mari. Problema principală este rezolvarea problemei pătratice care separă vectorii suport din setul de date de antrenare. Partea de program consumatoare de timp de execuție, este găsirea hiperplanului de separare a formelor în două clase.

Agoritmul HyParSVM propus de autorii (Eitrich et al., 2006) are un nivel crescut de paralelism implementat MPI prin metoda cross validation. Metoda de paralelizare hibridă include, la antrenarea clasificatorului, metoda rapidă de calcul (en. *Gradient projection method*) pentru seturi mari de date (Serafini et al., 2005). Algorimtul SMO paralelizat pe un singur nivel, propus de autoarele (Cao et al., 2006), folosește biblioteca MPI și modelul SPMD. Programul distribuie în mod egal datele de antrenare în funcție de numărul de procesoare disponibile. Rezultatele obținute arată o accelerare încurajatoare, dar reușește paralelizarea doar la nivel de date, nu și de calcule. O abordare evolutivă EASVM (en. *evolutionary approach to support vector machines*) bazată pe SVM, propusă de (Radu et al.,

2011), demostrează utilitatea paralelizării algoritmilor genetici în rezolvarea clasificării seturilor mari de date. Rezultatele obținute la antrenarea SVM cu algoritmul paralel PCPSVM pe 10 procesoare s-au dovedit a fi încurajatoare (Soiman et al., 2015a).

Algoritmul PGPDT

Metoda de paralelizare GPM (en. *Gradient Projection Method*), a algoritmului SVM, propusă de (Zanni et al., 2006) foloseşte tehnica de descompunere a problemei pătratice de programare QP, în subprobleme QP mai mici. PGPDT (Parallel Gradient Projection-based Decomposition Technique) necesită produsul matrice – vector de dimensiune mare, ce poate fi paralelizat la fiecare iterație. La testarea performanțelor, am folosit un număr de 8 servere blade QS22, cu câte două procesoare PowerXCell8i, care rulează la o frecvență de 3.2 GHz fiecare.

Pentru evaluarea performanțelor, s-au înregistrat timpii de execuție obținuți la rularea programului PGPDT pe 1, 2, 4, 8 și 16 procesoare PowerXCell 8i, cu activarea nucleelor PPE, folosind cinci seturi de date. Distribuirea sarcinilor pe sistemul paralel de calcul se face atât pentru rezolvarea subproblemei cât și pentru update-ul matricii Kernel. O îmbunătățire a timpului de execuție se observă pentru setul de date *url*, de la 2 ore la 3,3 minute, la execuția algoritmului paralel PGPDT pe 16 procesoare PPE (Figura 2). Testul cu setul de date *mnist8* arată că partea dominantă a antrenării SVM este rezolvarea subproblemei și nu update-ul matricii Kernel, un factor de accelerare mai mic fiind obținut în acet caz, la rularea algoritmului paralel pe 16 procesoare.

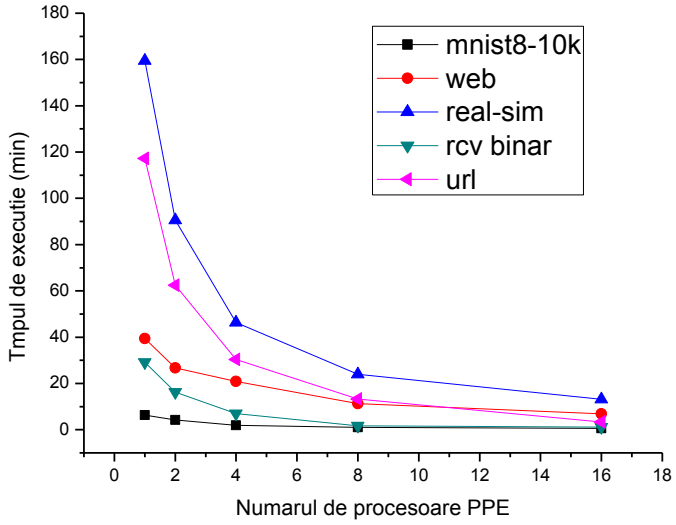

Figura 2 Timpul de execuție obținut la antrenarea clasificatorului SVM cu algoritmul paralel PGDT pe clusterul USV_Roadrunner

Testele demonstrează o scădere a timpului de execuție a programului PGPDT, privind dimensiunea setului de date sau numărul de procesoare implicate în antrenarea clasificatorului pe platforma MPI. Rezultatele din Figura 3, arată un factor de accelerare 35, obținut la execuția programului paralel propus de (Zanni et al., 2006) pe 16 procesoare PPE, la antrenarea clasificatorului SVM cu setul de date *url*.

Rezultatele au fost obținute la paralelizarea algoritmului SVM pe un singur nivel, utilizând protocolul comunicației prin mesaje. Paralelizarea algoritmilor SVM se dovedește a fi promițătoare în termenii reducerii timpului de antrenare a clasificatorului, în următorul subcapitol este studiată a doua tehnică de paralelizare a algoritmului, prin activarea acceleratoarelor SPE ale procesorului PowerXCell 8i.

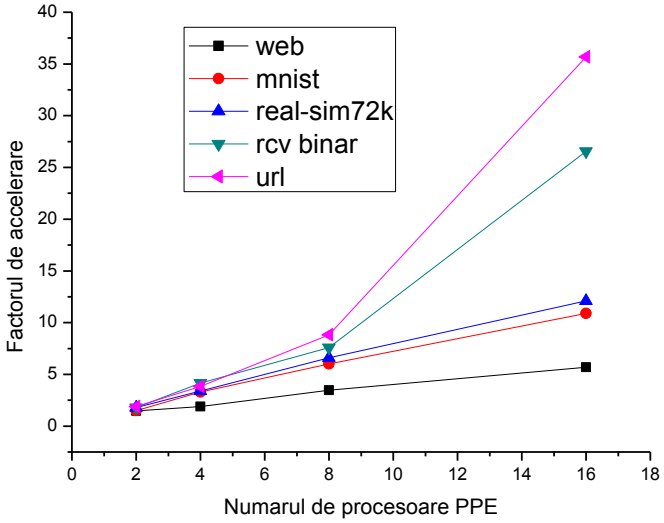

Figura 3 Factorul de accelerare obținut la execuția algoritmului paralel PGDT pe clusterul USV_Roadrunner folosind 1 până la 16 procesoare PPE

5.5 Algoritmi SVM paraleli multi-nivel

Algoritmul LibSVM$_{CBE}$, proiectat de M Marzolla (Marzolla, 2010), folosește biblioteca LibSVM, pe structuri paralele Cell/B.E., a cărei arhitectură a fost prezentată în Capitolul 3. Algoritmul paralel SMO a redus timpul de execuție considerabil, prin trimiterea calculelor cele mai consumatoare de timp (90%), a evaluării matricei Kernel, la procesoarele SPE ale procesorului Cell. LibSVM$_{CBE}$ distribuie evaluarea matricii kernel procesoarelor SPE, calculul produsului scalar a doi vectori este realizat cu instrucțiuni SIMD. Dimensiunea matricii kernel crește pătratic odată cu numărul de date de antrenare. Având un număr de peste sute de mii de date de antrenare necesarul de memorie pentru antrenarea clasificatorului devine foarte ridicat. Algoritmul SMO poate reduce necesarul de memorie prin analizarea a

câte doi vectori de antrenare (VS) și îi găsește următorii doi vectori (*multiplicatori Lagrange*) care satisfac anumite constrângeri. LIBSVM folosește metoda din lucrarea (Chang and Lin, 2001) pentru găsirea indicilor celor doi multiplicatori ce satisfac constrângerile.

Rezultate experimentale

Rezultatele rulării algoritmului paralel $\text{LibSVM}_{\text{CBE}}$ prezentate în (Marzolla, 2011) au fost obținute pe un PS3 folosind 6 seturi de date. Pentru realizarea testelor am optimizat algoritmul paralel și am folosit un singur blade QS22 a cluster-ului USV_RoadRunner, având două procesoare Cell/B.E., cu acces la 16 SPE-uri. Rezultatele obținute la antrenarea clasificatorului SVM pornind de la biblioteca LibSVM implementată pe Cell, are ca rezultat modelul SVM (numărul de vectori suport, parametrii nucleului). Performanțele algoritmului paralel LibSVM_CBE au fost evaluate cu 5 seturi de date și nucleu RFB. Datele de antrenare a clasificatorului sunt prezentate prin numărul de forme, numărul de caracteristici și densitatea datelor. Datele sunt similare cu cele alese de (Marzolla, 2011), cu completarea setului *url*, disponibile pe Internet.

Timpul de execuție reprezintă un prim indicator al performanței rulării pe clusterul USV_RoadRunner a algoritmului paralel LibSVM_CBE pe două procesoare PowerXCell8i, cu activarea celor 16 acceleratoare. Coloana *Densitate* reprezintă procentajul numărului de elemente nenule (caracteristici ale fiecărei forme) din setul de date de antrenare, 100% este densitatea maximă, toate elementele sunt diferite de zero.

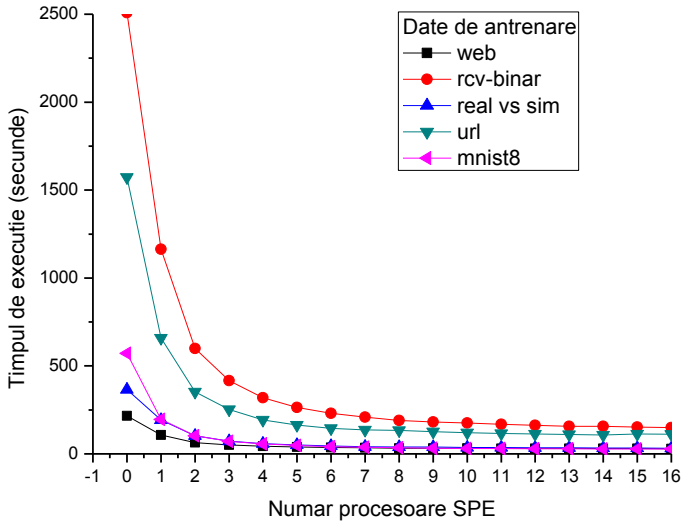

Figura 4 Timpii de excuție obținuți la testarea algoritmului paralel LibSVM_CBE pe clusterul USV_Roadrunner cu cinci seturi de date

Cel mai mare câștig de accelerare obținut la antrenarea clasificatorului SVM, este 22,89, la activarea a 16 nuclee SPE pe un nod de calcul, cu setul de date *mnist8-10k*. În acest

caz, timpul de comunicație între procesoarele SPE este mult mai mic decât timpul necesar calculelor (matricii kernel) pe fiecare SPE. Din rezultatele obținute la antrenarea clasificatorului SVM cu setul de date *rcv-binar* se observă cea mai bună scalabilitate a algoritmului paralel rulat pe un procesor PPE și pe 1 până la 16 procesoare SPE, ale unui nod QS22 cu două procesoare PowerXCell 8i. Pentru setul de date *url*, timpul de execuție la antrenarea clasificatorului SVM cu algoritmul paralel LibSVM_CBE, scade de la 26 minute (pe un procesor PPE cu un singur fir de execuție), la mai puțin de 2 minute prin distribuirea calculelor celor 16 procesoare (Figura 4).

Factorul de accelerare (Ec. 3.1), este calculat în funcție de timul de execuție obținut pe un singur procesor (PPE) a algoritmului secvențial LIBSVM, și timpul de execuție a algoritmul paralel LibSVM_CBE pe clusterul USV_Roadrunner cu activarea celor 16 nuclee SPE.

Cel mai mic factor de accelerare este obținut cu setul de date *web* (5) cu densitatea datelor 3,88%. Acest lucru se datorează faptului că pentru un număr mic de date, timpul de antrenare a clasificatorului SVM nu scade semnificativ deoarece timpul de comunicație este mai mare decât timpul necesar realizării calculelor.

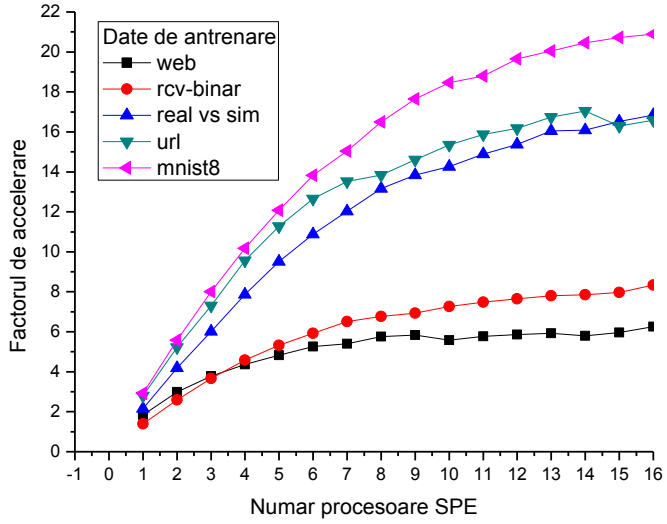

Figura 5 Accelerarea obținută la testarea algoritmului LibSVM_CBE pe clusterul USV_Roadrunner cu 5 seturi de date de antrenare

5.6 Concluzii

Obiectivul acestui capitol a fost de a determina performanța a doi algoritmi paraleli utilizați în construirea modelului cu clasificatorul SVM. În acest sens, s-au ales seturi diferite de date cu un număr de forme și caracteristici foarte mare. Având un număr de peste sute de mii de date de antrenare, cu un număr de peste 150 mii de caracteristici, necesarul de memorie și timpul folosit pentru antrenarea clasificatorului devine foarte ridicat, de aici nevoia de a utiliza

algoritmi paraleli. S-au înregistrat timpii de execuție a algoritmilor paraleli: PGPDT la nivel MPI și LibSVM_CBE la nivelul nucleelor SPE ale procesoarelor cu arhitectură hibridă Cell/B.E.

Rezultatele obținute cu algoritmul paralel PGPDT (Zanni et al., 2006), la execuția pe 1 până la 16 procesoare PPE (Tabelul 2), arată o îmbunătățire a timpului privind dimensiunea setului de date sau numărul de procesoare implicate în antrenarea clasificatorului pe platforma MPI. Factorul de accelerare a algoritmului paralel PGPDT (Figura 3), este calculat conform (Ec. 3.1), prin raporatrea timpului de execuție pe 16 procesoare PPE la timpul de execuție pe un singur procesor. Pentru setul de date *reuters corpus volume – binary*, cu o densitate mică 0,15%, și un număr de $2 \cdot 10^4$ forme cu $4{,}7 \cdot 10^4$ caracteristici s-a obținut un factor de accelerare 26,5 pe 16 procesoare PPE, iar cu setul de date *url*, cu același număr de forme, dar de trei ori mai multe caracteristici ($1{,}5 \cdot 10^5$) s-a obținut un factor de accelerare maxim, 35,5, pe același număr de procesoare. Cel mai mic câștig de accelerare a fost obținut la antrenarea SVM cu setul de date *mnist8* (10,9) și *web* (5,7), la rularea algoritmului PGPDT pe 16 procesoare PPE. Paralelizarea algoritmului SVM pe un singur nivel și-a dovedit utilitatea pentru seturile de date de antrenare mari, cu densitatea mică a datelor (<0); eficiența algoritmului paralel fiind scăzută (70%) pentru seturile de date cu o densitate mai mare, în acest caz timpul de comunicație între procesoarele PPE este mult mai mare decât timpul necesar calculelor.

Cel mai mare câștig de accelerare al construirii modelului de clasificare cu SVM, este 20,89 (Figura 5) a fost obținut cu biblioteca LibSVM_CBE, cu setul de date *mnist8*, la activarea a 16 procesoare SPE, ale unui blade QS22. Acest rezultat este promițător pentru clasificarea seturilor de date de dimensiuni foarte mari pe procesoare PowerXCell 8i cu mai multe nuclee.În urma acestor teste, s-a evidențiat faptul că performanțele obținute la evaluarea algoritmului LibSVM_CBE sunt mai bune pe seturile de date cu o densitate mare (*mnist8*), fiind rezolvată problema timpului necesar calculelor, comparativ cu soluția PGPDT, unde cele mai bune rezultate s-au obținut pe seturi de date cu o densitate mică (*url*). La antrenarea SVM cu setul de date *url*, cu algoritmul paralel PGPDT folosind platforma MPI se obține un câștig de performanță mai bun la distribuirea calculelor procesoarelor PPE, spre deosebire de algoritmul LibSVM_CBE unde timpul necesar calculelor (matricii Kernel) este mai mic decât timpul de comunicație între nucleele acceleratoare SPE. Eficiența algoritmului paralel LibSVM_CBE la antrenarea clasificatorului SVM cu setul de date *mnist8*, este mult mai bună decât a algoritmului PGPDT.

Capitolul 6

Paralelizarea algoritmului HMM Forward

6.1 Introducere

În acest capitol sunt prezentați doi algoritmi paraleli HMM Forward, propuși în capitolul IV, în scopul reducerii timpului de execuție prin utilizarea puterii de calcul a cluster-ului de calculatoare USV Roadrunner (IBM). Prima implementare a algoritmului Forward paralelizat prin transmiterea de mesaje MPI, numită FWD_MPI, distribuie sarcinile de lucru la nivelul fiecărui procesor PPE în funcție de lungimea secvenței de simboluri observabile. Nucleul PPE ce are la bază o structură PowerPC pe 64 biți, reprezintă principalul element de procesare și asigură interfața dintre celelalte nuclee subordonate și nodul de control.

A doua implementare paralelă a algoritmului Forward numită FWD_CBE, utilizează nucleele SPE ale fiecărui procesor PowerXCell 8i. La acest nivel paralelizarea s-a realizat în funcție de numărul de stări ale modelului Markov. Analiza algoritmilor paraleli propuși a fost făcută în comparație cu versiunea secvențială algoritmului Forward (**Error! Reference source not found.**). În analiza efectuată s-au avut în vedere mai mulți indici de performanță: timpul de execuție, factorul de accelerare și eficiența algoritmilor paraleli.

Rezultatele obținute la optimizarea algoritmului FWD, ce utilizează transferuri DMA și platforma MPI a cluster-ului USV Roadrunner (IBM), sunt diseminate în articolele: (Soiman et al., 2014) și (Soiman et al., 2015b). Accelerarea 38, este obținută la rularea programului FWD_MPI pe 32 procesoare PPE, iar la activarea nucleelor SPE ale procesorului Cell/B.E. cu programul FWD_CBE s-a obținut factorul de accelerare 75.8. Pe același cluster, la rularea algoritmului paralel multinivel Monte Carlo pe 92 de procesoare Cell/B.E., (Rusu et al., 2012) au redus considerabil timpul de execuție de la 237 secunde la 2.96 secunde.

Rezultatele experimentale ale celor două implementări ale algoritmului paralel Forward sunt obținute pe cluster-ul de calculatoare USV Roadrunner (IBM) din Laboratorul de Calcul de Înaltă performanță al Universității ”Ștefan cel Mare” din Suceava.

6.2 Paralelizarea algoritmului HMM Forward pe un singur nivel

Prima implementare paralelă a algoritmului Forward calculează probabilitatea secvenței de observații distribuită la procesoarele PowerXCell8i. Paralelizarea s-a realizat la nivelul nodurilor de management ale cluster-ului USVRoadRunner, ce utilizează protocolul MPI.

Pe fiecare nucleu PPE, a procesoarelor PowerXCell8i, rulează simultan algoritmul paralel Forward pentru o dimensiune mai redusă a problemei. Pentru rularea programului paralel se folosește biblioteca MPI. Se calculează probabilitatea Forward parțială pe fiecare procesor

PPE în funcție numărul de secvențe distribuit, urmând ca după finalizarea calculelor, să se colecteze aceste valori cu funcția *MPI_AllReduce()*.

Primele rezultate experimentale ale algoritmului FWD_MPI, au fost obținute pe un singur model HMM cu un număr redus de stări și secvențe de observații de lungimi diferite (de la 80 mii la peste 25 miloane de simboluri observabile). A doua observație a fost realizată pentru modele HMM diferite, cu 24 până la 1024 de stări, folosind aceeași secvență observabilă. La testarea performanțelor algoritmului FWD_MPI pe cluster-ul USV Roadrunner (IBM), s-au folosit un număr de 16 servere blade QS22, cu câte două procesoare PowerXCell8i, care rulează la frecvența 3.2 GHz fiecare. Algoritmul FWD_MPI a fost evaluat pe 32 de procesoare PowerXCell 8i, cu activarea nucleelor PPE. Pentru a pune în evidență avantajul unei abordări paralele a algoritmului propus, s-au înregistrat timpii de excuție a algoritmului secvențial Forward (Liu, 2009) pe un singur procesor PowerXCell.

REZULTATE EXPERIMENTALE OBȚINUTE LA TESTAREA ALGORITMULUI FWD_MPI PE CLUSTERUL USV ROADRUNNER (IBM)

Pentru evaluarea performanțelor algoritmului FWD_MPI s-au generat modele HMM cu un număr de stări pornind de la 24 până la 2014 stări. Rezultatele au fost înregistrate cu aceste modele HMM cu același număr de simboluri observabile (M=2)și lungimea secvenței T=3200 de simboluri observabile.

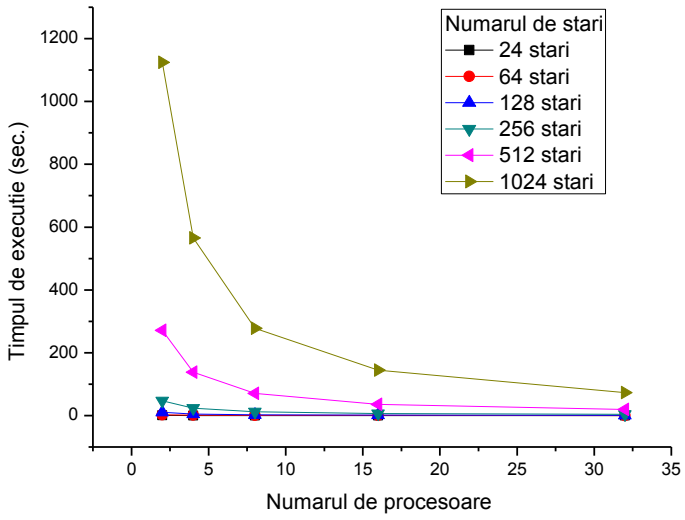

Figura 6 Timpul de execuție a algoritmului FWD_MPI pe 32 procesoare pentru diferite modele HMM

Rezultatele obținute la rularea algoritmului paralel FWD_MPI pe 2 până la 32 de procesoare PPE, arată o îmbunătățire a timpului de execuție înregistrat pentru modele cu numărul de stări mai mare de 256 (Figura 6).

Tabelul 2 Factorul de accelerare a algoritmului paralel FWD_MPI
pentru 6 modele HMM fără activarea procesoarelor SPE

Număr procesoare	24 stari	64 stari	128 stari	256 stari	512 stari	1024 stari
2 PPE	2,8	2,9	2,9	3,1	2,6	2,5
4 PPE	4,8	5,6	5,7	6,1	5,0	5,0
8 PPE	7,8	10,7	11,4	12,2	9,9	10,1
16 PPE	10,0	18,0	21,7	23,4	19,6	19,4
32 PPE	8,4	28,0	37,6	37,1	35,6	38,4

Cel mai mare câștig de accelerare al algoritmului paralel FWD_MPI (Tabelul 2), este pentru modelul HMM cu 1024 stări, cu un factor 38, nu foarte departe fiind testele cu 256 stări, la care factorul de accelerare atinge valoarea 37. Similar, (Rusu et al., 2013) au atins un factor de accelerare 40 la testarea rutinei PDSYEV, pentru calculul valorilor proprii ale unor matrici de dimensiuni foarte mari, pe 90 de procesoare ale clusterului USV Roadrunner (IBM). Odată cu creșterea numărului de procesoare PPE, crește și factorul de accelerare a algoritmului, ceea ce demonstrează o bună distribuire a sarcinilor de lucru.

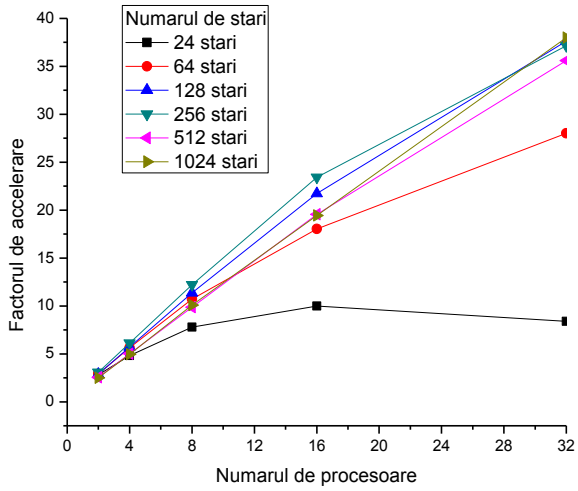

Figura 7 Accelerarea algoritmului FWD_MPI executat pe 32 de procesoare
PowerXCell8i fără activarea nucleelor SPE, pentru 6 modele HMM

Rezultatele prezentate în acest subcapitol sunt promițătoare, calculul probabilității Forward necesită resurse de calcul ridicate, pentru modele HMM cu un număr mare de stări și secvențe de observații lungi. Performanțele algoritmului Forward implementat pentru a rula în

paralel folosind biblioteca MPI, folosind 6 modele HMM și secvența de observații cu lungimea 3200 de simboluri observabile, sunt satisfăcătoare (factor de accelerare 38,4) față de rezultatele obținute cu rularea algoritmului secvențial Forward pe un singur procesor PowerXCell 8i (Figura 7). În acest sens, în următorul subcapitol a fost dezvoltată a doua tehnică de paralelizare a algoritmului, prin activarea acceleratoarelor SPE ale procesorului PowerXCell 8i și trimiterea calculului cel mai mare consumator de timp și memorie către acestea.

6.3 Paralelizarea multi – nivel a algoritmului HMM Forward

Al doilea algoritm paralel FWD_CBE, realizează distribuirea și efectuarea calculelor pe două nivele ale procesoarelor PowerXCell8i, datorită celor două tipuri de nuclee conținute, 1 PPE și 8 SPE-uri. Rezultatele obținute de (Pentiuc and Ungurean, 2012) au demonstrat eficiența paralelizării multinivel a algoritmilor de clasificare nesupravegheată a datelor pe procesoare PowerXCell8i bazate pe arhitectura hibridă. Dificultatea paralelizării algoritmului Forward la nivelul acceleratoarelor SPE constă în folosirea memoriei locale de doar 256 KB și transferul datelelor in memoria locală asociată fiecărei unități SPE în blocuri de 128 de octeți. Memoria locală de 256 KB este folosită atât pentru programul SPU cât și pentru date. SPU nu poate accesa memoria principală direct, pentru aceasta folosește comenzile DMA prin MFC pentru a aduce datele în memoria locală sau scrie rezultatele în memoria principală. Programul SPU poate continua în timp ce MFC realizează în mod independent tranzacțiile DMA. Se vor discuta si evalua doi algoritmi paraleli (**Error! Reference source not found.**) ce se vor executa pe procesoarele PPE și (**Error! Reference source not found.**) pentru nucleele SPE. Algoritmul PPE are rolul de a gestiona firele de execuție și de a controla transferurile dintre cele două tipuri de procesoare prin DMA. Programul SPE constă dintr-o buclă care așteaptă o comandă de la PPU prin intermediul mailbox. Comanda este trimisă la toate SPE-urile împreună cu EA a datelor. SPE sunt procesoare specializate cu o arhitectură SIMD. Acest tip de arhitectură este potrivită aplicațiilor ce necesită un calcul intens asupra seturilor multiple de date.

A doua implementare a algoritmului Forward, FWD_CBE, realizează distribuirea calculelor probabilităților *alpha* la nivelul fiecărui nucleu SPE în scopul de a reduce timpul de calcul a probabilității Forward pentru modele HMM cu un număr de stări mare,. La nivelul SPE-rilor se vor calcula valorile *alpha,* linia 8 a algoritmului secvențial (**Error! Reference source not found.**), pentru numărul de stări asociate *N/* nr_SPE. Primul nivel de paralelizare, se face prin distribuirea numărului de secvențe de simboluri observabile la numărul de procesoare PPE, cu offset-ul corespunzător. Numărul de secvețe de observații asociat procesului MPI curent este împărțit în mod egal la cele 8 nuclee acceleratoare SPE. Pentru fiecare procesor PPE se determină numărul de secvențe de observații asociate și offset-ul de unde încep acestea. Pentru memorarea secvențelor se folosește vectorul $data[nseq * len]$. Offset-ul secvențelor de observații $index\_data\_SPU$, asociat fiecărui procesor PPE, va fi transmis fiecărui nucelu SPE odată cu parametrii modelului HMM, necesari în calculul valorilor *alpha.* Fiecare procesor PPE

va transmite celor 8 nuclee SPE valoarea $index\_data\_SPU$ corespunzătoare, pentru calculul index-ului corespunzător numărului de secvențe de observații asociate fiecărei unități SPE.

Construirea matricii Forward α, de dimensiune $len\ x\ N$, este realizată pe procesoarele PPE, linie cu linie. Calculul fiecărei valori α_{ij} se realizează la nivelul nucleelor SPE. Calculul unei linii din matricea α, α_i este dependent de valoarea liniei precedente α_{i-1}, ceea ce necesită actualizarea liniei din matricea α pe fiecare procesor PPE, după finalizarea construirii valorilor acesteia pe SPE-uri. Prima linie a matricii α se inițializează cu probabilitatea inițială a stărilor (Π) pe fiecare procesor PPE. La următoarele $(len - 1)$ linii, pentru calculul unei singure linii este nevoie de toată matricea probabilităților de tranziție dintre stări (pătratică de ordinul N) - (A), depășind capacitatea memoriei locale fiecărei unități SPE, pentru un număr mai mare de 128 de stări.

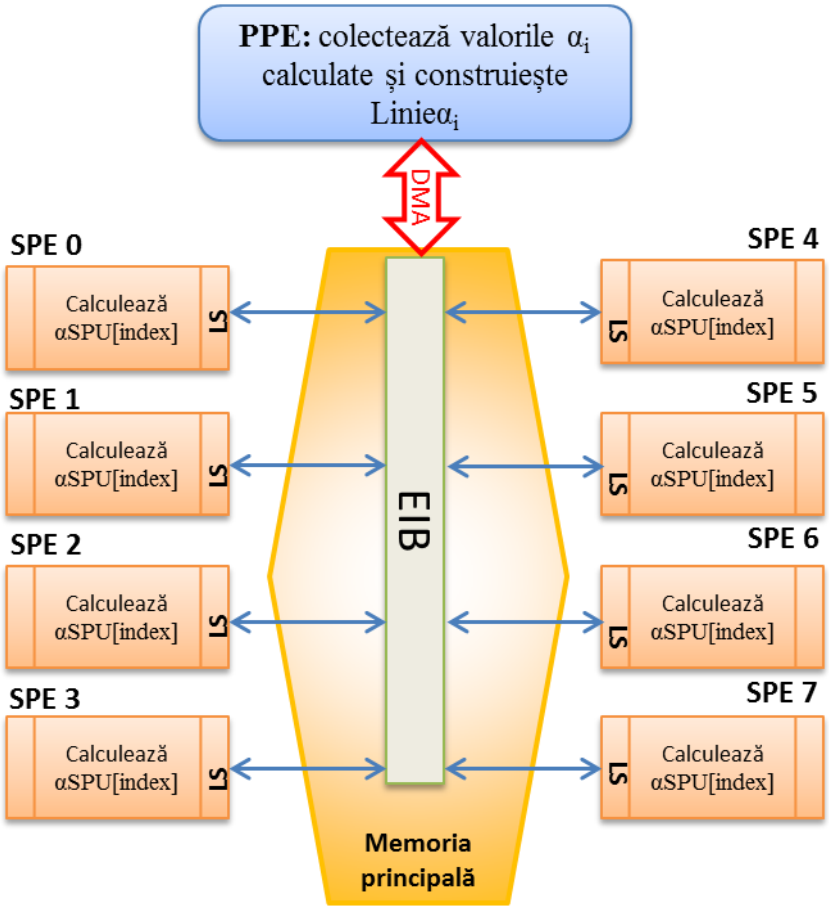

Figura 8 Calculul distribuit al valorii *alpha* pe procesoarele SPE

Paralelizarea algoritmului FA pe două nivele se face pornind pe fiecare procesor SPE un fir de execuție, realizând calculul distribuit al variabilei α_{ij} pe nucleele SPE. Pentru distribuirea egală a calculelor, se vor împărți numărul de stări a modelului HMM la numărul de procesoare

SPE disponibile. Index-ul simbolurilor secvenței de observații corespunzătoare calculului valorii $alpha$ asociate SPE-ului spu_id, se va calcula astfel:

$$index\_data\_SPU = spu\_id * (N / NR\_PROC\_SPU) \quad \text{Ec. 6.2}$$

Fiecare unitate SPE va avea alocat calculul unui bloc de N/NR_PROC_SPU valori alpha, unde N este numărul de stări ale modelului HMM și NR_PROC_SPU este 8 pentru procesoarele PowerXCell 8i. Distribuirea calculelor valorilor *alpha* la nucleele SPE poate fi cea mai bună metodă, fiecare nucleu SPE va calcula valorile pentru blocul de stări asociate (Figura 8).

O valoare *alpha* va fi calculată raportat la indexul secvenței de observații asociat SPE-ului (Ec. 6.2) și offsett-ului simbolului din secvența de observații $offset\_data\_SPU$.PPE va colecta valorile *alpha* calculate pe SPE-uri și va construi noua *Linieα*. Transferul datelor pe fiecare nucleu SPE din/în memoria locală în/din memoria principală se va face prin transfer DMA in blocuri de date de mărime multiplu de 128 octeți.

Algoritmul paralel FWD_CBE care rulează la nivelul procesoarelor PPE

```
*) citeşte din fişier parametrii modelului HMM
*) alocă memorie pentru calculul paralel a probabilității
*) calculează offset pentru repartiția secvențelor pe fiecare procesor PPE
*) trimite comandă la nucleele SPE de citire a parametrilor modelului HMM
(N, nobvs, obvs, length, data și offset_data_PPE)
pentru toate blocurile de secvente de observații repartizate fiecarui PPE
      *) inițializează prima linie a matricii α, Linieα0=prior+obvs
      *) trimite la SPE prima linie a matricii α, Linieα0
      *) calculează offset pentru calcul distribuit al valorii αi la
      procesoarele SPE
      *) trimite comandă la nucleele SPE de citire a unei coloane din
      matricea trans
      *) așteaptă de la nucleele SPE comanda de terminare a citirii
      coloanei matricii trans
      *) trimite comandă la nucleele SPE de calcul a valorii αi pentru
      stările asociate
      *) așteaptă de la nucleele SPE comanda de finalizare a calculelor
      distribuite fiecărui nucleu SPE
      *) așteaptă de la nucleele SPE semnalul că au fost realizate
      calculele și transferate în memoria principală.
      *) actualizează Linieα cu valorile calculate pe SPE-uri
```

```
      *) trimite comandă la nucleele SPE de citire a Linieα
      *) așteaptă de la nucleele SPE comanda de terminare a citirii Linieα
      *) calculează probabilitatea secvenței de observații însumând
      valorile ultimei linii din matricea α
sfârșit pentru
*) trimite comandă de oprire a execuției la nucleele SPE
*) așteaptă de la nucleele SPE să semnalizeze terminarea execuției
*) folosește funcția MPI_AllReduce() pentru colectarea probabilității
Forward parțiale de pe toate procesele MPI din sistem
*) calculează probabilitatea Forward totală P
```

Algoritmul paralel FWD_CBE executat pe nucleele SPE:

```
Repeta
*) asteapta comenzi de la nucleul PPE (asteapta primirea unui mailbox)
daca *) s-a primit comanda de citire a parametrilor modelului HMM
      *) transfera prin DMA în memoria locala structurile de date care
      contin parametrii modelului HMM
      *) trimite procesorului PPE un mailbox prin care este semnalizata
      terminarea cititrii parametrilor
daca *) s-a primit comanda de citire a unei coloane din matricea
probabilitatilor de tranzitie dintre stari
      *) transferă prin DMA din memoria principala in memoria locala
      coloana matricei trans necesara calculului unei valori αi pe SPE-uri
      *) trimite procesorului PPE un mailbox prin care este semnalizata
      terminarea cititrii
daca *) s-a primit comanda de calcul a valorii alpha
      *) calculează offset_data_SPU pentru distribuirea calculului alpha la
      toate procesoarele SPE
      *) efectuează calculul alpha pentru numărul de stări asociate
      fiecărui SPE
      *) transferă prin DMA in memoria principala valorile calculate care
      au fost asignate nucleululi SPE curent
      *) trimite procesorului PPE un mailbox prin care este semnalizata
      terminarea transferului valorilor alpha
daca *) s-a primit comanda de transfer a unei linii din matricea α
      *) transferă prin DMA din memoria principală in memoria locală
      Linieα
      *)trimite procesorului PPE un mailbox prin care este semnalizată
```

```
      terminarea transferului
pana cand *) s-a primit comanda de terminare a executiei
```

6.4 Evaluarea performanțelor obținute pe clusterul USV Roadrunner (IBM) a algoritmului paralel FWD_CBE

Platforma hardware utilizată în testarea algorimilor propuși este clusterul USV Rodrunner, cu o arhitectură hibridă. Arhitectura clusterului permite ca cele două blade-uri IBM BladeCenterQS22, IBMLS21 și blade-ul de comunicație să fie încapsulate pe un singur nod de calcul. Pe fiecare nod memoria este de 32 Gb, nodurile Opteron sunt mapate 1 – la -1 pe nodurile Cell BE cu o putere de calcul de 102.4 Gflops. Creșterea performanței aplicațiilor ce rulează pe cluster este dată de faptul că memoria este egal distribuită la toate componentele nodului de calcul. Pentru realizarea testelor am folosit 16 QS22 blade-uri cu două procesoare Cell/B.E, având acces la 256 SPE-uri.

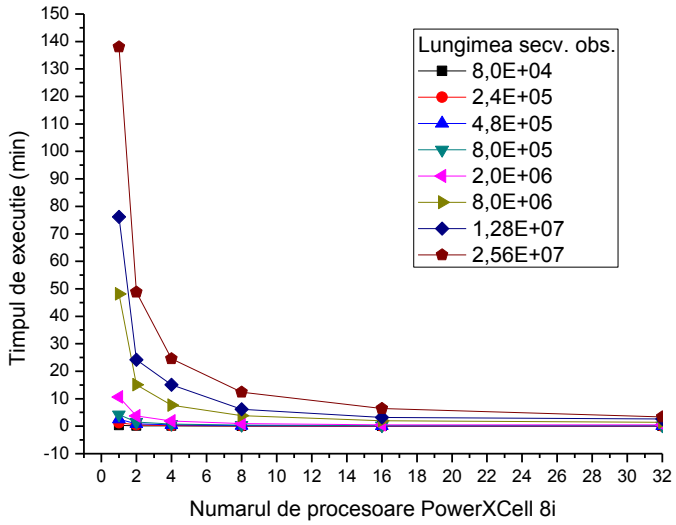

Figura 9 Timpul de execuție a algoritmului paralel FWD_CBE pentru un model HMM cu 8 secvențe obs. executat pe 1 până la 32 procesoare PPE

A doua implementare a AF pentru modele HMM cu un număr mare de stări este denumită FWD_CBE. Rezultatele algoritmul FWD_CBE rulat pe 32 procesoare PowerXCell 8i sunt prezentate în funcție de numărul de stări ale modelelor HMM și lungimea secvenței de observații.

În primă fază, am testat algoritmul paralel Forward cu modelul Markov cu N=24 stări, M=2 simboluri observabile și 8 secvențe de observații cu un număr de la 80000 până la 25 milioane de simboluri observabile.

Timpii de execuție obținuți cu algoritmul paralel Forward (FWD_CBE) în funcție de lungimea secvenței de observații, sunt prezentați în . Pe fiecare proces MPI, a fost calculată probabilitatea Forward locală/parțială aferentă blocului din secvența de observație distribuit procesului MPI. Pentru testarea performanțelor s-a înregistrat timpul de execuție pe 1 până la 32 procesoare PPE. Conform graficului din Figura 9 se observă o scădere considerabilă a timpului de execuție atunci când crește numărul de procesoare și lungimile secvenței de observații sunt de dimensiuni mari. Timpul de execuție pentru un singur procesor PPE, este timpul înregistrat la rularea algoritmului secvențial pe un singur procesor PowerXCell. Factorul de accelerare, calculat conform (Ec. 3.1), este obținut la rularea algoritmului paralel FWD_MPI pe 2 până la 32 procesoare PowerXCell8i față de execuția secvențială a AF.

Așa cum se observă în Figura 10, testele efectuate au arătat o mai bună scalabilitate pentru secvențe cu lungimi de 25 milioane de simboluri observabile. Datorită creșterii timpilor de comunicație între procesoare, la folosirea a mai mult de 20 procesoare și secvențe de observații mici, eficiența algoritmului începe să scadă (Soiman et al., 2015c). Eficiența paralelă (Ec. 3.3) cumulează cele două performanțe înregistrate: timpul de execuție și accelerarea algoritmului paralel, și arată gradul de încărcare a procesoarelor PPE. Valoarea eficienței algoritmului FWD_CBE, pentru secvențe mai mari de $2e^{+6}$simboluri observabile tinde spre valoarea 1 și chiar o depășește. Se poate observa că odată cu scăderea numărului de secvențe și creșterea numărului de procesoare, valoarea eficienței scade datorită creșterii timpilor de comunicație între procesoare.

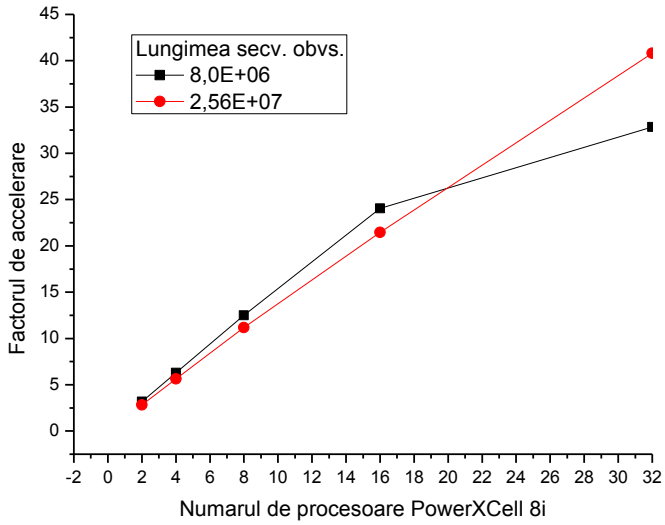

Figura 10 Accelerarea algoritmului FWD_CBE pentru un model HMM
și secvențe de observații lungi

Secvențele de observații folosite în evaluarea performanțelor algoritmului paralel FWD_CBE, de la 8e+4 până la 2.6e+7 simboluri observabile, sunt reprezentate în fișierul de intrare printr-un număr de 800 până la 256000 de secvențe, cu lungimea de 100 simboluri.

Eficiența maximă a algoritmului paralel pe 32 procesoare este, în acest caz, pentru testul cu 256000 secvențe de observații, fiecare procesor PPE calculează probabilitatea Forward pentru 8000 de secvențe.

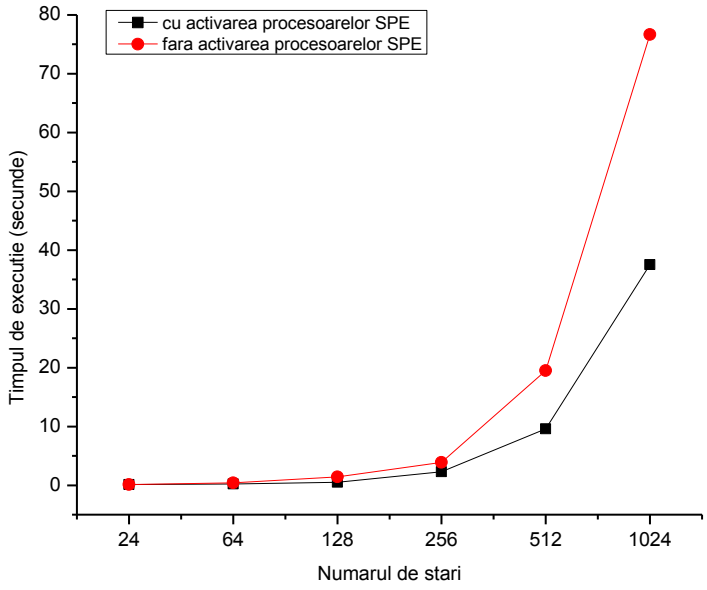

Figura 11 Timpul de execuție a algoritmului FWD_CBE în funcție de numărul de stări ale modelului HMM pe 32 procesoare PowerXCell 8i cu și fără activarea SPE-urilor.

Calculul probabilității pe un singur procesor durează peste 2 ore, în timp ce pe 32 procesoare PPE, calculul se realizează în 3 minute și jumătate. Al doilea set de experimente au fost realizate pe 6 modele HMM generate aleator cu un număr de 24 până la 1024 stări și o singură secvență de 3200 simboluri observabile, utilizând 2,4,8,16, respectiv 32 de procesoare PPE cu cele 8 nuclee SPE activate.

Pentru modele HMM cu un număr mai mare de 500 de stări se observă scăderea timpului de execuție a algoritmului FWD_CBE executat pe 32 procesoare odată cu activarea nucleelor SPE, ajungând, pentru modelul cu 1024 stări, să fie dublu față de timpul de execuție a algoritmului FWD_CBE, fără activarea nucleelor SPE (Figura 11).Din testele efectuate cu algoritmul paralel FWD_CBE, observăm o importantă accelerare (Figura 12) atunci când ambele procesoare PPE și SPE sunt utilizate. Din rezultatele înregistrate, se observă un factor de accelerare 75.8 a algoritmului FWD_CBE, o creștere semnificativă față de factorul 38, obținută la execuția algoritmului parale FWD_MPI cu același test (Soiman et al., 2015b). Acest lucru demonstrează superioritatea algoritmului FWD_CBE pentru acest tip de modele HMM.

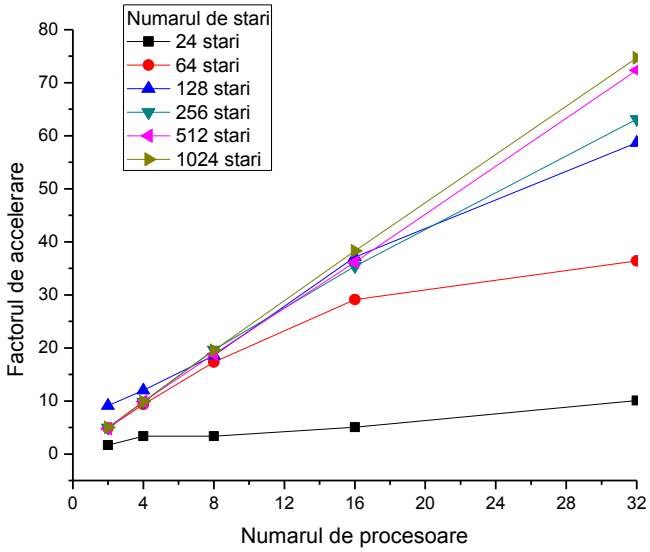

Figura 12 Accelerarea algoritmului FWD_CBE rulat pe 32 de procesoare PowerXCell8i cu activarea acceleratoarelor SPE obținută pentru 6 modele HMM

6.5 Concluzii

Algoritmul FWD_CBE propus în această lucrare a fost dezvoltat utilizând și nucleele acceleratoare SPE ale procesorului PowerXCell8i, nuclee caracteristice arhitecturii Cell/B.E. Probabilitatea FWD se calculează recursiv, întreg procesul necesitând resurse mari calcul pentru acele modele cu un număr mare de stări și secvențe de observație lungi. FWD_CBE optimizează efectuarea numeroaselor calcule de probabilități într-o manieră de paralelizare multi-nivel. Procesoarele PPE reprezintă primul nivel de paralelizare, lungimea secvenței de observație împărțindu-se între acestea. Fiecare PPE împarte numărul de stări ale modelului la cele 8 nuclee SPE, în acest fel realizându-se și cel de al doilea nivel de paralelizare.

Am obținut rezultate satisfăcătoare la paralelizarea algoritmului Forward pe cluster-ul de calculatoare cu o arhitectură IBM Roadrunner. Așa cum se observă în Figura 7, pentru modele HMM cu mai mult de 100 de stări, accelerarea este aproape liniară. În Figura 11 sunt reprezentate performanțele celor două implementări paralele diferite ale AF. Folosind avantajele procesoarelor Cell/B.E., la activarea acceleratoarelor SPE am obținut un factor de accelerare 75.8. Implementările paralele dezvoltate au demonstrat potențialul și avantajele arhitecturii clusterului USV Roadrunner (IBM) din Laboratorul de Calcul de Înaltă Performanță al Universității „Ștefan cel Mare” din Suceava.

Rezultatele obținute au demonstrat un nivel înalt de scalabilitate a clusterului USV Roadrunner (IBM) în rezolvarea problemelor ce implică modele Markov ascunse cu un număr mare de stări sau secvențe observabile lungi, utilizate în rezolvarea unor probleme complexe precum ar fi cele ce implică un vocabular de dimensiune mare.

Capitolul 7

Concluzii finale, contribuții și direcții viitoare de cercetare

7.1 Concluzii generale

Contribuțiile aduse în această teză demonstrează potențialul computațional al sistemelor paralele de calcul în domeniul inteligenței artificiale și a recunoașterii formelor. Paralelismul este exploatat pentru eficientizarea a doi algoritmi de clasificare: algoritmul paralel Forward, pentru rezolvarea unei probleme fundamentale a modelelor Markov ascunse și algoritmul paralel Sequential Minimal Optimization, circumscris formalismului vectorilor suport. Cercetările efectuate au vizat soluții pentru creșterea performanțelor aplicațiilor de calcul paralel în clasificarea supravegheată a datelor și implementarea acestora. Testele efectuate pe clusterul USV Roadrunner (IBM), din Laboratorul Calcul de Înaltă Performanță al Facultății de Inginerie Electrică și Știința Calculatoarelor al Universității "Ștefan cel Mare" Suceava, au evidențiat performanțele ridicate obținute prin creșterea numărului de noduri și activarea accelelratoarelor SPE în rezolvarea problemelor de clasificare a seturilor foarte mari de date. În studiul performanțelor algoritmilor propuși, s-au ales seturi diferite de date la antrenarea clasificatorului SVM, respectiv modele HMM cu un număr mare de stări și secvențe de simboluri observabile lungi.

Volumul de calcul este deosebit de ridicat pentru modelele Markov cu un număr mare de stări utilizate în aplicații precum recunoașterea scrisului de mână, vorbirii sau în analiza traficului web. O îmbunatațire considerabilă a acurateței recunoașterii scrisului, s-a obținut la dublarea numărului de stări ale modelului Markov ascuns Bakis, pornind de la 855 de stări ajungând până la peste 1500 de stări (Geiger et al., 2010). La aceste modele, matricea probabilităților de tranziție este foarte mare, ceea ce ridică probleme privind memoria și timpul mare de execuție serială. De aceea cercetările întreprinse în această teză privind paralelizarea acestor algoritmi sunt necesare și actuale pentru rezolvarea unor probleme reale precum cele din domeniul recunoașterii vorbirii (LVCSR – large vocabulary continous speech recognition) unde se utilizează modele Markov ascunse foarte complexe (Silingas and Telksnys, 2004). În recunoașterea vorbirii, numărul de foneme este foarte ridicat (în cazul limbii arabe, s-au identificat 27.000 de foneme), deoarece pronunția lor la începutul cuvântului, în interiorul cuvântului sau la sfârșitul cuvântului este diferită. Spre exemplu, numărul mare de stări ale modelelor Markov ascunse poate apare în analiza traficului web sau în problema detecției spam-urilor. Sunt necesare modele HMM cu sute sau chiar mai multe stări, pentru a detecta evenimente ce pot influența rata de download în analiza datelor de utilizare a siteurilor web (Felzenszwalb et al., 2003).

Contribuțiile aduse în această teză vizează doi algoritmi din domeniul recunoașterii formelor, în contextul clasificării supravegheate. Algoritmii în discuție fac referire la lanțurile Markov (HMM) și mașina cu vectori suport (SVM) utilizată în clasificarea binară.

Prin studiile și analizele efectuate asupra acestor două subiecte am prezentat provocările care se ridică în dezvoltarea algoritmilor de recunoașterea formelor utilizând resurse de calcul de înaltă performanță atunci când numărul de stări și/sau lungimea secvenței de observații ale modelului HMM este foarte mare, iar în cazul SVM, a numărului de forme din setul de date de antrenare.

Prima metodă propusă în dezvoltarea algoritmului Forward (HMM) a vizat paralelizarea pe un singur nivel, utilizând protocolul comunicației prin mesaje (MPI), algoritm identificat prin FWD_MPI. În cadrul acestui algoritm volumul de calcul a fost distribuit pe 16 noduri dual-core PowerXCell8i, adică 32 de nuclee PPE.

Al doilea algoritm propus, FWD_CBE, a fost dezvoltat utilizând și nucleele acceleratoare SPE ale procesorului PowerXCell8i, nuclee caracteristice arhitecturii CBE (Cell Broadband Engine). În studiul performanței algoritmului FWD_CBE propus, am efectuat o serie de teste cu parametri diferiți (numărul de stări ale modelului HMM și lungimea secvenței de observații), teste ce au fost rulate pe clusterul USV Roadrunner (IBM). Testele au urmărit înregistrarea atât a timpilor de execuție atunci când sunt utilizate doar procesoarele PPE folosind modele HMM cu un număr redus de stări și secvențe de observații lungi, cât și timpii obținuți când sunt activate nucleele acceleratoare ale procesorului PowerXCell 8i, folosind modele HMM cu un spațiu mare al stărilor. Pe baza acestor date au fost trasate diagrame din care pot fi identificați factorii care contribuie la creșterea sau scăderea performanțelor algoritmului. Factorul de accelerare cel mai bun obținut cu algoritmul paralel FWD_CBE a fost de 75 pe un model HMM cu 1024 de stări, pe 256 de nuclee SPE. Probabilitatea FWD a secvenței de observații se calculează recursiv, întreg procesul este consumator de timp și resurse de calcul pentru modele cu un număr mare de stări. FWD_CBE optimizează efectuarea numeroaselor calcule de probabilități, prin trimiterea calculului variabilei forward către procesoarele SPE. Primul nivel de paralelizare se face prin distribuirea volumului datelor de intrare (numărul de secvențe de observații) către procesoarele PPE, iar al doilea nivel prin realizarea calculelor de către procesoarele SPE (în funcție de numărul de stări ale modelului HMM).

Rezultatele obținute au demonstrat un nivel înalt de scalabilitate a clusterului USV Roadrunner (IBM) în rezolvarea problemelor ce implică modele Markov ascunse cu un număr mare de stări sau secvențe observabile lungi, utilizate în rezolvarea unor probleme complexe precum ar fi cele ce implică un vocabular de dimensiune mare (Lee et al., 2007).

Cea de-a doua parte dedicată algoritmilor de clasificare construiți cu ajutorul vectorilor suport (SVM) are ca obiectiv sinteza, analiza și optimizarea algoritmilor paraleli utilizați în construirea modelului cu clasificatorul SVM. Pentru a evidenția câștigul de performanță, am ales seturi diferite de date cu un număr de forme și caracteristici foarte mare. Lucrând cu seturi de

antrenare cu un număr de forme de ordinul sutelor de mii și cu un număr de peste 150 mii de caracteristici, necesarul de memorie și timpul de calcul pentru antrenarea clasificatorului devine foarte ridicat, de aici nevoia de a utiliza algoritmi paraleli. Algoritmul paralel LibSVM_CBE a fost optimizat pentru a rula pe un blade QS22 a cluster-ului USV_RoadRunner, având două procesoare Cell/B.E. Cel mai mare câștig de accelerare al construirii modelului de clasificare cu SVM, este 20,89 și a fost obținut cu biblioteca LibSVM_CBE, pe setul de date *mnist8*, la activarea a 16 procesoare SPE. Acest rezultat este promițător pentru clasificarea seturilor de date de dimensiuni foarte mari pe procesoare PowerXCell 8i cu mai multe nuclee.

Cel de al doilea algoritm analizat bazat pe SVM este PGPDT, dezvoltat de (Zanni et al., 2006). În acest caz, volumul de calcul a fost distribuit doar la nivelul procesoarelor PPE, pe aceleași seturi de date de antrenare, dar cu un număr de forme diferit. Testele efectuate la antrenarea clasificatorului SVM au urmărit înregistrarea timpilor de execuție atunci când sunt utilizate procesoarele PPE ale procesoarelor PowerXCell 8i. Astfel, algoritmul paralel PGPDT și-a dovedit utilitatea pentru seturile mari de date, cu o densitate mică (<0). Câștigul de viteză pentru setul de date *url* este 35,5 la utilizarea a 16 procesoare PPE. Eficiența algoritmului paralel scade (70%), pentru seturile de date de antrenare a clasificatorului SVM cu o densitate mare (*mnist8*), în acest caz timpul de comunicație între procesoarele PPE este mult mai mare decât timpul necesar calculelor.

7.2 Contribuțiile tezei

Pentru dezvoltarea și analiza algoritmilor propuși în cadrul acestei teze am utilizat supercalculatorul USV Roadrunner (IBM) din cadrul laboratorului de HPC al Universității noastre, sistem paralel echipat cu procesoare PowerXCell8i bazate pe arhitectura hibridă Cell/ B.E.

Principalele contribuții aduse în cadrul tezei sunt următoarele:

1. Proiectarea a două metode de paralelizare multinivel a algoritmului Forward (HMM) pe clusterul USV Roadrunner (IBM). În analiza algoritmului Forward am identificat punctele în care se poate face paralelizarea la nivel MPI și la nivelul nucleelor SPE.
2. Dezvoltarea algoritmului HMM Forward pe un singur nivel de paralelizare utilizând protocolul comunicației prin mesaje - s-a distribuit întreg volum de calcul pe fiecare nucleu PPE ale procesoarelor PowerXCell 8i, pentru calculul parțial al probabilității Forward. Soluția finală este construită la nivelul procesului master, prin însumarea soluțiilor parțiale de pe procesele slave.
3. Conceperea și dezvoltarea algoritmului paralel FWD_CBE pe două nivele de paralelizare, optimizat pentru sisteme cu procesoare hibride Cell/B.E. Primul nivel de paralelizare permite împărțirea calculului probabilității Forward la procesorele PPE prin standardul MPI. Al doilea nivel de paralelizare a algoritmului Forward implică ca fiecare

PPE să distribuie calcul variabilei forward la cele opt nuclee acceleratoare SPE. Acest lucru s-a realizat prin comenzile trimise de PPE către acceleratoarele SPE ale procesorului PowerXCell8i. La acest nivel, s-a definit o metodă prin care calculele efectuate la nivelul fiecărui procesor SPE sunt sincronizate cu transferul prin DMA a datelor între procesoare.

4. Evaluarea performanțelor algoritmului paralel FWD_MPI, paralelizat pe un singur nivel, prin utilizarea a procesoarelor PPE - s-a realizat comparația între varianta secvențială a algoritmului Forward și varianta paralelizată, s-au analizat performanțele obținute folosind mai multe modele HMM cu un număr diferit de stări pe clusterul USV Roadrunner (IBM).
5. Evaluarea rezultatelor soluției paralele FWD_CBE comparativ cu varianta secventială a algoritmului HMM Forward – s-a realizat o comparație între varianta secvențială a algoritmului Forward, varianta paralelizată pe un singur nivel și varianta paralelizată pe două nivele și s-au analizat performanțele obținute la execuția algoritmului pe clusterul USV Roadrunner (IBM).
6. Analiza perfomanțelor algoritmului PGPDT în antrenarea clasificatorului SVM prin execuția algoritmului pe clusterul USV Roadrunner (IBM), paralelizat utilizând protocolul MPI. S-a arătat că paralelizarea algoritmului SVM pe un singur nivel a fost utilă pentru seturile mari de date de antrenare a clasificatorului, cu o denistate mică (procentajul numărului de elemente nenule). Cele 5 seturi de date alese sunt disponibile pe Internet, reprezentând sisteme ce modelează probleme din lumea reală (simulatoare auto și aviație, clasificatoare de știri în funcție de subiect, recunoașterea cifrelor etc.).
7. Sinteza, analiza și optimizarea algoritmului paralel SVM pentru calculatoare cu o arhitectură CBEA - s-a realizat o comparație la rularea algoritmului paralel LibSVM_CBE pe un singur procesor PPE, utilizând până la 16 nuclee acceleratoare SPE ale procesoarelor PowerXCell 8i.

7.3 Diseminarea rezultatelor

Rezultatele și contribuțiile prezentate în această lucrare au fost concretizate prin publicarea și prezentarea următoarelor lucrări științifice:

- *Conferință internațională indexată ISI*

1. **Stefania SOIMAN**, Ionela RUSU, Ştefan-Gheorghe PENTIUC, ***Multilevel Parallelized Forward Algorithm for Hidden Markov Models on IBM Roadrunner Cluster,*** acceptată și urmează a fi prezentată la Proceedings of the 20th International Conference on Control Systems and Computer Science, May 27-29, 2015.

- *Conferință internațională indexată IEEExplore*

2. **Stefania SOIMAN**, Ionela RUSU, Ştefan-Gheorghe PENTIUC, ***A parallel accelerated approach of HMM Forward Algorithm for IBM Roadrunner clusters***, Proceedings of the 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014, ISBN: 978-1-4799-5095-9, pag. 184-189.

- *Jurnal ISI*

3. **Stefania SOIMAN**, Ionela RUSU, Ştefan-Gheorghe PENTIUC, ***Optimizing the Forward Algorithm for Hidden Markov Model on IBM Roadrunner clusters,*** acceptat spre publicare la la revista Advances in Electrical and Computer Engineering *(Factor Impact 2015: 0.642)*

- *Jurnal B+ indexat în EBSCO, DOAJ, ICAAP, Zentralblatt Math, Copernicus*

4. Ştefan Gheorghe PENTIUC, **Ştefania SOIMAN**, ***A Proposed Method for Pattern Classification with HMM in the Context of Supervised Learning,*** Journal of Applied Computer Science & Mathematics, vol. 19, pp. 50-56, 2015, ISSN: 2066-4273.
5. Ionela RUSU, Ștefan-Gheorghe PENTIUC, Elena-Gina CRĂCIUN, **Stefania SOIMAN,** ***A Performance Evaluation of QR-eigensolver on IBM Roadrunner cluster for Large Sparse Matrices***, Journal of Applied Computer Science & Mathematics, vol. 14, 2013, ISSN: 2066-4273.
6. Elena-Gina CRĂCIUN, Ștefan-Gheorghe PENTIUC, Ionela RUSU, **Stefania SOIMAN**, ***Ellipse - Novel Manipulation Technique for 3D Rotation Based on***

Continuous Axis Control, Journal of Applied Computer Science & Mathematics, vol. 14, 2013, ISSN: 2066-4273.

- *Conferință internațională indexată BDI*

7. **Ştefania ŞOIMAN**, Ştefan Gheorghe PENTIUC, ***Analysis of hybrid pattern recognition system with HMM and SVM methods,*** Proceedings of the 11th Int. Conf. on Development and Appl. Systems, May 17-19, 2012 Suceava, Romania, Abstracts Book, pp.55

8. **Stefania SOIMAN,** Radu GHEORGHE, R., Ştefan Gheorghe PENTIUC, Ionut BALAN, 2015a***. Performance Analysis of Parallel SVM Training on Cluster Having Similar IBM Roadrunner Architecture.*** International Conference of Scientific Paper AFASES 2015. Brasov, Romania.

- *Conferință națională*

9. **Stefania SOIMAN**, Stefan-Gheorghe PENTIUC **(Soiman and Pentiuc, 2011)**, ***Analiza sistemelor hybrid de recunoastere a formelor cu metodele HMM si SVM, Sisteme Distribuite, Suceava,*** Revista Sisteme Distribuite (Suceava - online), 2011, ISSN 1842-6808, pag. 15-19.

7.4 Direcții viitoare de cercetare

Algoritmii paraleli propuși în această teză au arătat potențialul ridicat a procesoarelor Cell/B.E. la clasificarea supravegheată a datelor. Se pot lua în considerare următoarele direcții viitoare de cercetare:

- Dezvoltarea algoritmilor HMM paraleli optimizați care să folosească instrucţiunile SIMD ale procesoarelor SPE.
- Folosirea potențialului cluster-ului USV Roadrunner în dezvoltarea algoritmilor SVM paraleli multinivel la clasificarea multi-clasă a seturilor mari de date.
- Utilizarea algoritmilor HMM-SVM paraleli în sisteme hibride de recunoaștere a formelor.

BIBLIOGRAFIE SELECTIVĂ

[1] AREVALO, A., MATINATA, R. M., PANDIAN, M., PERI, E., RUBY, K., THOMAS, F. & ALMOND, C. 2008. *RedBooks. Programming the Cell Broadband Engine™ Architecture Examples and Best Practices*.

[2] BAKER, J. 1975. The DRAGON system--An overview. *Acoustics, Speech and Signal Processing, IEEE Transactions on,* 23**,** 24-29.

[3] BARKER, K. J., DAVIS, K., HOISIE, A., KERBYSON, D. K., LANG, M., PAKIN, S. & SANCHO, J. C. 2008. Entering the petaflop era: The architecture and performance of Roadrunner. *International Conference for High Performance Computing, Networking, Storage and Analysis.*

[4] BEAL, M. J., GHAHRAMANI, Z. & RASMUSSEN, C. E. 2001. The infinite hidden Markov model. *Advances in neural information processing systems.*

[5] CAO, L. J., KEERTHI, S. S., CHONG-JIN, O., ZHANG, J. Q., PERIYATHAMBY, U., XIU JU, F. & LEE, H. P. 2006. Parallel sequential minimal optimization for the training of support vector machines. *Neural Networks, IEEE Transactions on,* 17**,** 1039-1049.

[6] CHANG, C.-C. & LIN, C.-J. 2001. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.,* 2**,** 1-27.

[7] EITRICH, T., FRINGS, W. & LANG, B. 2006. HyParSVM a new hybrid parallel software for support vector machine learning on SMP clusters. *In:* PROCESSING, E.-P. P. (ed.) *Proceedings of the 12th international conference on Parallel Processing.* Dresden, Germany: Springer-Verlag.

[8] FELZENSZWALB, P. F., HUTTENLOCHER, D. P. & KLEINBERG, J. M. 2003. Fast Algorithms for Large-State-Space HMMs with Applications to Web Usage Analysis. *In:* THRUN, S., SAUL, L. K. & SCHÖLKOPF, B. (eds.) *NIPS.* MIT Press.

[9] GEIGER, J., SCHENK, J., WALLHOFF, F. & RIGOLL, G. 2010. Optimizing the Number of States for HMM-Based On-line Handwritten Whiteboard Recognition. *Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on.*

[10] JOACHIMS, T. 1999. Making large-scale support vector machine learning practical. *Advances in kernel methods.* MIT Press.

[11] LEE, S., JEONG, I. & CHOI, S. 2007. Dynamically weighted hidden Markov model for spam deobfuscation. *Proceedings of the 20th international joint conference on Artifical intelligence.* Hyderabad, India: Morgan Kaufmann Publishers Inc.

[12] LIFSHITS, Y., MOZES, S., WEIMANN, O. & ZIV-UKELSON, M. 2009. Speeding up HMM decoding and training by exploiting sequence repetitions. *Algorithmica,* 54**,** 379 - 399.

[13] LIN, J. & DYER, C. (eds.) 2010. *Data-Intensive Text Processing with MapReduce*: Morgan & Claypool Publishers.

[14] LIU, C. 2009. cuHMM: a CUDA Implementation of Hidden Markov Model Training and Classification. *online*.

[15] MARZOLLA, M. 2010. Optimized Training of Support VectorMachines on the Cell Processor.

[16] MARZOLLA, M. 2011. Fast training of support vector machines on the Cell processor. *Neurocomputing,* 74**,** 3700-3707.
[17] MENG, X. & JI, Y. 2013. Modern Computational Techniques for the HMMER Sequence Analysis. *ISRN Bioinformatics,* 2013**,** 13.
[18] NIELSEN, J. & SAND, A. 2011. Algorithms for a Parallel Implementation of Hidden Markov Models with a Small State Space. *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum.* IEEE Computer Society.
[19] PADRON, R. R. 2007. A Roadmap to SVM Sequential Minimal Optimization for Classification.
[20] PENTIUC, S.-G. & UNGUREAN, I. 2012. Multilevel Parallelization of Unsupervised Learning Algorithms in Pattern Recognition on a Roadrunner Architecture. *Intelligent Distributed Computing V.* Springer.
[21] RABINER, L. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE,* 77**,** 257 - 286.
[22] RADU, G., PENTIUC, Ş. G. & BALAN, I. 2011. Paralelizarea algoritmilor evolutivi pentru instruirea maşinilor cu vectori suport. *Seminarul ştiinţific cu participare naţională Sisteme Distribuite.* Suceava.
[23] RUSU, I., PENTIUC, S. G., UNGUREAN, I. & CRACIUN, E. G. 2012. A parallel approach for Monte Carlo-based photon propagation simulation. *2012 5th Romania Tier 2 Federation Grid, Cloud & High Performance Computing Science.* Cluj Napoca, ROMANIA: Ieee.
[24] RUSU, I., PENTIUC, Ş.-G., CRĂCIUN, E.-G. & SOIMAN, S. I. 2013. A Performance Evaluation of QR-eigensolver on IBM Roadrunner cluster for Large Sparse Matrices. *Journal of Applied Computer Science & Mathematics,* 14**,** 38-41.
[25] SAND, A., MARTIN, K., CHRISTIAN NS, P. & THOMAS, M. 2013. zipHMMlib: a highly optimised HMM library exploiting repetitions in the input to speed up the forward algorithm. *BMC Bioinformatics,* 14.
[26] SAND, A., PEDERSEN, C. N. S., MAILUND, T. & BRASK, A. T. 2010. HMMlib: A C++ Library for General Hidden Markov Models Exploiting Modern CPUs. *Proceedings of the 2010 Ninth International Workshop on Parallel and Distributed Methods in Verification, and Second International Workshop on High Performance Computational Systems Biology.* IEEE Computer Society.
[27] SERAFINI, T., ZANGHIRATI, G. & ZANNI, L. 2005. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods & Software,* 20**,** 347-372.
[28] SILINGAS, D. & TELKSNYS, L. 2004. Specifics of Hidden Markov Model Modifications for Large Vocabulary Continuous Speech Recognition. *Informatica,* 15**,** 93-110.
[29] SOIMAN, S.-I., GHEORGHE, R., PENTIUC, Ş.-G. & BALAN, I. 2015a. Performance Analysis of Parallel SVM Training on Cluster Having Similar IBM Roadrunner Architecture. *International Conference of Scientific Paper AFASES 2015.* Brasov, Romania.
[30] SOIMAN, S. I. & PENTIUC, S. G. 2011. Analiza sistemelor hybrid de recunoaştere a formelor cu metodele HMM şi SVM *Seminarul ştiinţific cu participare naţională Sisteme Distribuite, Suceava*.

[31] SOIMAN, S. I., RUSU, I. & PENTIUC, Ș.-G. 2014. A parallel accelerated approach of HMM Forward Algorithm for IBM Roadrunner clusters. *12th International Conference on Development and Application Systems* Suceava, Romania.
[32] SOIMAN, S. I., RUSU, I. & PENTIUC, Ș.-G. 2015b. Multilevel Parallelized Forward Algorithm for Hidden Markov Models on IBM Roadrunner Clusters *20th International Conference On Control Systems And Computer Science (CSCS20).* Bucharest, Romania.
[33] SOIMAN, S. I., RUSU, I. & PENTIUC, Ș.-G. 2015c. Optimizing the Forward Algorithm for Hidden Markov Model on IBM Roadrunner clusters. *Advances in Electrical and Computer Engineering*
[34] TAYLOR, P. 2006. Unifying unit selection and hidden Markov model speech synthesis. *INTERSPEECH.* ISCA.
[35] ZANNI, L., SERAFINI, T. & ZANGHIRATI, G. 2006. Parallel Software for Training Large Scale Support Vector Machines on Multiprocessor Systems. *Journal of Machine Learning Research,* 7**,** 1467-1492.

[36] *LIBSVM Data: Classification (Binary Class).* http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
[37] McCallum, A.K.: Simulated/real/aviation/auto usenet data,
a. *http://people.cs.umass.edu/~mccallum/data.html*