Mașina DIALISP - o realizare cu efecte întârziate

Gheorghe Ștefan
Universitatea Politehnica
București, Bd. Iuliu Maniu 1-3, Bucharest, Romania.
E-mail: gheorghe.stefan@upb.ro

Într-o dupamiază de iarnă, la începutul anilor 80, în biroul său din ICCI, profesorul Mihai Drăgănescu mă întreba ce părere am despre posibilitatea de a realiza o mașină Lisp. În următoarele săptămâni în laboratorul din catedră[1], în care se realizase deja mașina grafică DIAGRAM, se formează un subcolectiv în care au fost incluși Aurel Păun (absolvent recent al facultații de Calculatoare), Virgil Bistriceanu și Andy Birnbaum (studenți în ultimul an ai facultății noastre). Botezăm DIALISP mașina pe care începeam să o proiectăm.

Realizarea unui proiect neuzual, în condiții anormale poate avea urmări imprevizibile. Este cazul proiectului DIALISP, finalizat în următorii ani, dar care se continuă și astăzi prin conceptele pe care le-a catalizat, însă nu le-a putut duce decât parțial la împlinire. Este vorba de:

- **microprogramarea multi-thread** implementată pentru prima dată în unitatea centrală a mașinii DIALISP
- **memoria conexă** -- inabordabilă cu tehnologia anilor 80-90 -- pentru care perspectiva este deschisă și promițătoare în contextul tehnologic actual.

La sfârșitul anilor '70 profesorul Mihai Drăgănescu se întorcea din Statele Unite cu intenția fermă a dezvoltării domeniului **inteligenței artificiale** în România. Pe lângă formarea unui colectiv în ICCI, în catedră organizează primele lecții de Lisp, ținute de Mihaela Malița în cadrul cursului de *Electronică Funcțională* - un curs deschis către subiecte de maximă actualitate. Cristian Giumale preda de mai mulți ani limbajul Lisp la facultatea de Automatică șI Calculatoare. Declanșarea proiectului DIALISP a fost astfel sprijinită din start de gândul bun al unui nucleu de cercetători pentru care o "mașină de inteligență artificială" era un instrument visat[2].

Acest text este scris cu speranța că fiecărei idei utile îi poate veni rândul la folosire. *Mașinile Lisp* nu mai sunt "la modă", chiar *inteligența artificială* a devenit o sintagmă care nu mai electrizează ca în anii 70-80, dar o mulțime de idei ce au apărut în efortul de a le impune au intrat si continuă să intre în nucleul dur al științei calculului. În acestă lumină merită să rememorăm experiența proiectului și mașinii DIALISP.

---

[1] Catedra de Dispozitive, Circuite și Aparate Electronice a Facultății de Electronică [i Telecomunicații din Institutul Politehnic București.

[2] Cele trei mașini, care s-au produs într-o mini-serie zero, au ajuns, pe căi birocratico-financiare, într-o fabric` de becuri (două dintre ele) șă în centrul de calcul al unui mic oraș de provincie. Prima lucrare in limba românî care descria DIALISP-ul și conceptul de memorie conexă [Ștefan '85] nu a putut fi publicata decât în 1991, datorită interdicției prin care "*Biroul 2*" a blocat publicarea oricărui text ce se referea la inteligența artificială.

"Efectele întârziate" ale demersului nostru din anii 80 pot acționa constructiv în contextul tehnologic al anilor 2000.

Mașina Lisp

La sfârșitul anilor 50, John McCarthy punea în valoare *lambda-calculul* lui Alonzo Church propunând limbajul Lisp (**lis**t **p**rocessing) pentru studiul problemelor teoretice legate de computabilitate. Dezvoltarea aplicațiilor de **inteligență artificială** cerea folosirea unor limbaje prin care complexitatea programării putea fi mai ușor ținută sub control. "Capacitatea expresivă" a limbajului Lisp a fost cea care l-a impus ca instrument principal al inteligenței artificiale.

Problema, care a apărut imediat, a fost cea legată de incompatibilitățile Lisp-ului cu **arhitectura von Neumann**. Arhitecturile calculatoarelor curente erau -- și sunt încă!?!? -- "derivate" din modelul **mașinii Turing**, model propus în același an (1936) cu cel al lui Church. În anii '40, când arhitectura von Neumann a fost promovată, problema *competenței* calculatoarelor era mult mai importantă decât cea a *performanței* lor. Scopul imediat urmărit era acela de a se construi o mașină. Complexitatea calculului a fost o problemă ce s-a impus mult mai apoi, când era deja prea târziu ca arhitectura curentă să mai poată fi pusă în discuție.

Lisp-ul se interpreta lent și cu mare risipă de memorie pe arhitecturile von Neumann de uz general. Chiar cele mai bune compilatoare nu permiteau o creștere suficientă a vitezei și nu scădeau dimensiunea memoriei folosite. Soluția anilor '70-'80 a fost conceperea unor arhitecturi von Neumann specializate pentru limbajul Lisp. Tehnica microprogramării dinamice era metoda cea mai simplă de adaptare arhitecturală. Ea a fost adoptată în aproape toate variantele de mașini Lisp realizate.

Mașina DIALISP a avut ca unitate centrală un **procesor microprogramat dinamic bi-thread**. Thread-ul de microprogram principal realiza funcția "eval", iar cel secundar gestiona, în paralel, stiva. Primul lucra pe liste înlănțuite iar cel de al doilea era o stivă complexă.

Execuția întrețesută a programelor a fost pentru prima dată realizată în procesorul de intrare/ieșire al mașinii *Control Data 6600* [Thornton '64[, fiind permisă de diferența mare de viteză dintre unitatea centrală și echipamentele periferice. *Conceptul de multi-thread* este introdus în [Smith '78], referindu-se la o structură paralelă de procesoare.

Execuția multi-thread prin întreteserea microprogramelor este realizată pentru prima data în mașina DIALISP [Stefan '84], în scopul fructificării la maximum a resurselor interne rapide ale unui procesor microprogramat și pentru a asigura o cuplare foarte strânsă a două procesese: cel de evaluare a *s*-expresiilor și cel de gestionare a stivei.

Creșterea performanțelor brute ale calculatoarelor (viteza de execuție și capacitatea de memorare) a adus foarte repede mașinile curente la performanțele mașinilor Lisp. Avantajele microprogramării dinamice, ale micilor trucuri pe care le permitea un set de instrucțiuni adecvat

s-au dovedit neimportante în fața evoluției explozive a tehnologiilor microelectronice, tehnologii care nu mai sprijineau structurile microprogramate dinamic (procesoarele RISC, pe bună dreptate, se situau în centrul atenției).

Mai trebuie amintit și faptul că arhitecturile propuse nu s-au distanțat definitiv de stilul von Neumann. O mașină microprogramată rămâne o mașină von Neumann, oricât de exotic este setul de instrucțiuni pe care-l are. Execuția multi-thread a două procese intim cuplate este bine venită, dar se realizează tot prin procese convenționale.

Proiectarea mașinii DIALISP este un eveniment despre care merită astazi să vorbim pentru că a prilejuit și un proces paralel de investigare a unor alternative non-standard, pentru acele probleme care se obstinau să se lase rezolvate numai cu soluții prea puțin eficiente.

Căutând soluții, eram obligați să ne limităm la ceea ce tehnologia curentă ne oferea, dar *nimeni nu ne împiedica sa visăm*. Și despre aceste vise merită să discutăm în acest moment al reamintirii.

Memoria conexă

Cât de sâcâitor este să gestionezi liste care se pot răspândi exploziv în memorie! O știu toți cei ce au implementat un Lisp într-o mașină convențională, dar mai ales cei care s-au străduit să proiecteze o mașină Lisp. Împrăștierea "necontrolată" a unei liste pe mai multe pagini pune probleme fundamentale pentru managementul memoriei.

Disperați, de imposibilitaea folosirii unui mecanism eficient de înlocuire a paginilor, am hotărât să le înlocuim dupa un algoritm de substituție aleatoare. Nu a fost deloc rău[3].

Evaluarea funcțiilor definite multiplu recursiv se face foarte comod folosind mai multe stive. Noi aveam una foarte eficientă, dar numai una singură.

Micile insatisfacții, generate de soluțiile la care restricțiile tehnologice ne obligau, incitau la visare. Ce bine ar fi dacă am putea face ... *o memorie în care reprezentarea listelor să poată fi menținută conexă! ... în care să putem organiza simplu oricâte stive dorim! ... în care să regăsim rapid listele de asociere! ... unde să putem accesa ușor definiția unei funcții! ... o memorie care să permită "creșterea" unei stive de valori chiar în sub-lista de argumente a fiecărei funcții recursiv definite!* ... .

Insatisfacțiile intense stimulează imaginația. Uneori rezultă monștri pe care-i percepem însă din ce în ce mai prietenoși pe măsură ce putințele noastre tehnologice cresc. Relaxarea indusă de forța tehnologiei este pozitivă atunci când este folosită pentru a promova în primul rând idei noi și numai în plan secund pentru a sprijini o gândire prăfuită.

[3] În deceniul următor, Hennessy și Patterson [Hennessy '92] făceau aceași recomandare, în virtutea unor măsuratori destul de riguroase.

Un astfel de "monstru" a fost, în 1985, conceptul de **memorie conexă** - MC. El a rezultat din voința de a materializa vise de genul celor anterior listate. Conceptul era nerealist la momentul apariției, dar viteza cu care tehnologiile evoluau îi dădea o șansă. Legea lui Moore a lucrat și lucrează încă șFallows '01ț în favoarea MC.

Ce este memoria conexă?

Răspuns: o posibilă soluție la "*silicon gap*", la discrepanța din ce în ce mai mare dintre posibilități tehnologice explozive și o încremenire arhitecturală cronică.

Prima caracteristică: MC este o memorie care are *elemente de procesare și evaluare atașate fiecărui cuvânt.* Este formată din celule, serial conectate, care posedă atât funcția de stocare cât și pe cele de prelucrare și evaluare elementare. Evaluarea este minimală: celulele sunt clasificate în numai două categorii - *marcate* și *nemarcate*.

A doua caracteristică: MC posedă și o *buclă globală*, folosită pentru a clasifica celulele, încă odată, în trei clase: celule ce preced prima celulă marcată, prima celulă marcată și celule ce urmează primei celule marcate.

Funcțiile executate de MC presupun același cod recepționat de fiecare celulă. Execuția codului este realizată în fiecare celulă în funcție de starea ei - marcată sau nemarcată - și în funcție de poziția în care o califică bucla globală.

Ce facilități oferă folosirea memoriei conexe?

Căutarea în timp constant este o primă facilitate, prin care poziția unui șir de simboluri în memorie este identificată, prin marcare într-un timp independent de dimensiunea memoriei și egal, în număr de cicluri de ceas, cu lungimea șirului căutat.

Fie un *text*, T, de lungime *n* și un *patern*, P, de lungime *m*. Algoritmii consacrați pe arhitecturile secvențiale de tip von Neumann identifică aparițiile lui P în T în timp $O(n^m)$ folosind un spațiu de memoie în $O(n^2)$ (prin metoda *arborelui de sufixe* [Gusfield '97]). O arhitectură centrată pe MC rezolvă problema în timp $O(m)$ și spațiu $O(n)$. Atenție! De regulă: $n>>>m$.

Mai mult! Care este constanta asociată cu $O(n^m)$ și care este cea asociată cu $O(m)$ în evaluarile anterioare. Prima constantă este mai mare de 1600, iar cea de a doua este 1 [Ștefan '01].

Menținerea conectivității structurilor de date este o a doua facilitate importantă a MC. Posibilitatea de a insera sau a extrage (delete) primul simbol marcat într-un singur ciclu de ceas asigură localitatea reprezentărilor. Funcții de genul *garbage collector* devin lipsite de sens.

Posibilitatea de a opera asupra unei subclase de celule permite realizarea unor procesări paralele deosebit de eficiente. Cea mai simplă operație de acest gen este scrierea în paralel a

aceluiași simbol în toate celulele marcate. Dar în egală masură se pot imagina operații aritmetice elementare realizate în paralel în toate celulele marcate.

Listarea funcțiilor elementare ale MC, în diverse versiuni, poate fi gasită în lucrările șȘtefan '85ț, șȘtefan '98aț.

Cât de mare este dimensiunea memoriei conexe?

Structura unei memorii DRAM presupune aproximativ un tranzistor pe bit memorat. Structura MC presupune, în funcție de setul elementar de operații, între 12 și 120 de tranzistori pe bit memorat. Dar ceea ce este important este că *dimensiunea* MC rămâne în clasa *O(n).*

În funcție de aplicațiile întrevăzute, dimensiunea celulei este mai mare sau mai mică. Spre exemplu, dacă folosim MC numai pentru aplicații legate de compresia de date în timp real, atunci se poate folosi o versiune cu aproximativ 12 tranzistoare pe bit; pentru aplicații în bioinformatică devine utilă o versiune cu 24-32 tranzistoare pe bit; pentru aplicații de baze de date începe să fie utilă o versiune ce are în jur de 64 de tranzistoare pe bit; dacă este întrevăzută o aplicație de uz general, celula ajunge să conțină în jur de 120 de tranzistoare pe bit memorat.

Sunt mari sau mici aceste constante? Ele sunt acceptabile, în primul rând, în calitatea lor de constante, deoarece accelerarea procesării se realizează de *O(n)* ori. Problema este ca *n* să poată fi suficient de mare în structura reală. Această problemă este soluționată de tehnologiile actuale, care permit, acum, în jur de 10^6 celule pe cip.

Exemplu: fie o problemă, considerată fundamentală în bioinformatică, unde $n=10^6$ este lungimea lui T iar $m=10^3$ este lungimea lui P. Folosind arborele de sufixe al șirului T se pot detecta aparițiile lui P în T într-un număr de aproximativ $1,6 \times 10^9$ cicluri de ceas. Folosind o arhitectură bazată pe MC problema se rezolvă în 10^3 cicluri de ceas. MC fiind, în accest caz, de cel mult 32 de ori mai mare, rezultă o **creștere de sute de mii de ori** a raportului *performanță/preț.*

Nu toate aplicațiile presupun o creștere la fel de spectaculoasă, dar se poate spune cu certitudine că raportul *performanță/pret* crește de *O(n)* ori.

Arhitecturi cu paralelism de profunzime

Paralelisml este o soluție în care multă lume își pune mari speranțe pentru creșterea peformanțelor sistemelor de calcul. Din păcate el este o realizare care încă nu a depășit nivelul structural. Putem concepe și construi structuri paralele (ce sunt circuitele dacă nu structuri paralele?) dar ele nu vor putea fi validate până când nu vor fi suportate de arhitecturi paralele. În domeniul sistemelor de calcul o structură neprotejată arhitectural nu are perspective de dezvoltare. Aceasta este situația în momentul de față. Cum poate fi ea depășită? Prin conceperea unor arhitecturi paralele, dacă acest lucru va fi posibil.

Este greu de imaginat o arhitectură paralelă eficientă atât timp cât modelele principale de calculabilitate, pe care se bazează încă știința calculatoarelor, sunt inerent secvențiale.

De asemenea, atât timp cât comunicarea interumană și cea dintre om și mașină se desfășoară sub dominația discursului în limbaj natural sau în limbaje formale, este aproape imposibil ca să se poată impune o modalitate pur paralelă de a concepe calculul.

Dar, așa cum în "spatele" gândului discursiv exprimat se află, de multe ori, procese mentale cu o componentă paralelă consistentă, tot așa am putea concepe și arhitecturi superficial secvențiale, dar cu o puternică componentă paralelă în profunzime.

Poate este mai natural să credem că atât mintea noastră cât și posibile sisteme de calcul se manifestă printr-un amestec de secvențial și paralel a căror ponderare ține de natura problemei gândite sau procesate.

Arhitecturile secvențiale cu paralelism de profunzime s-ar putea dovedi o soluție realistă pentru fructificarea paralelismului structural, iar MC poate fi unul dintre suporturile fizice cele mai adecvate.

Un bun exemplu de model de calculabilitate care poate combina *secvențialitatea superficială* cu *paralelismul de profunzime* îl reprezintă **algoritmii Markov**.

Prin definiție, un algoritm Markov este o mulțime **ordonată** de producții:

$$\alpha_1 \to \beta_1$$
$$\alpha_2 \to \beta_2$$
$$\dots$$
$$\alpha_n \to \beta_n$$

unde $\alpha_i, \beta_i \in A^*$, A fiind un alfabet finit. Fiind dat un șir $\gamma \in A^*$, ca dată de intrare, algoritmul Markov asociat lucrează în felul următor:

- aplică *prima* regulă pe care o poate aplica, substituind șirul α_j prin șirul β_j în șirul γ
- în șirul rezultat aplicăm iarăși prima producție ce poate fi aplicată și tot așa
- algoritmul se oprește atunci când nici una dintre producții nu mai poate fi aplicată.

Secvențialitatea algoritmilor Markov este dată de faptul că producțiile nu pot fi aplicate în paralel. Aplicarea unei producții modifică șirul γ, astfel încât următoarea regulă nu poate fi aleasă decât după ce regula curentă a realizat substituția.

Paralelismul adânc (ascuns) presupus de algoritmii Markov constă în faptul că deteminarea regulii ce se poate aplica trebuie făcută prin căutarea șirului α_i în șirul γ. Această cautare este realizabilă

printr-un proces paralel. Șirul α_i poate fi căutat în șirul γ, folosind o MC, într-un timp proporțional cu șirul căutat și independent de șirul în care se caută, care este, de regulă, mult mai mare.

Exemplul algoritmilor Markov este unul pur teoretic[4]. El este însă deosebit de semnificativ pentru posibilitatea conceperii unor arhitecturi cu paralelism de profunzime.

Până atunci, realizarea unor co-procesoare, care să acclereze funcții critice ale sistemelor de calcul în aplicații specifice, este domeniul în care CM își poate găsi cele mai rapide aplicații.

MC a fost gândită inițial pentru a sprijini realizarea unei mașini Lisp. Dar, chiar de la prima ei prezentare, din septembrie 1985 la Academie, Gheorghe Tecuci observa că o aplicație mai potrivită ar fi realizarea unei mașini Prolog. M-am declarat de acord, dar numai mult mai târziu, când am început să înțeleg ce este limbajul Prolog, i-am dat dreptate. Chiar dacă a fost gândită din perspectiva unei aplicații, se pare că utilitatea MC este mult mai largă. Poate cel mai puțin adecvată este pentru realizarea unei mașini Lisp, dând astfel dreptate lui Umberto Eco:

"Raționamentul pe care și-l plăsmuiește mintea noastră este ca o plasă, sau o scară, care se construiește ca să ajungi la ceva. Dar după aceea scara trebuie aruncată pentru că se descoperă că deși slujea, era lipsită de sens.[5]"

***

Care este perspectiva conceptelor de *execuție multi-thread întrețesută* și de *memorie conexă*? Evoluții spectaculoase în tehnologie și în tehnica programării au creat un nou context acestor concepte. De asemenea, apariția unor noi paradigme computationale -cum ar fi calculul molecular, calculul cu membrane, ș.a. - oferă cadrul unor aplicații nestandard [Ștefan '97], [Ștefan '98]. Presiunile, mai mult psihologice, la care suntem supuși de "silicon gap" fac investitorii mai sensibili la idei exotice.

Ca întotdeauna în lumea *hi-tech*-urilor, minuscule efecte imprevizibile, amplificate de reacții pozitive ale pieții șArthur '90ț vor promova idei și produse pe care nu le putem acum prevede.

Java este un limbaj care presupune programarea multi-thread. Sistemele de operare se implementează din ce în ce mai mult folosind thread-uri multiple[6]. Bazele de date presupun căutări din ce în ce mai sofisticate, în condițiile unei concurențe din ce în ce mai mari.

---

[4] Dacă ignorăm faptul că stă la baza limbajului SNOBOL.

[5] [Eco '84], pag. 488.

[6] Chiar înaintea finalizării acestui text Bogdan Mîțu mi-a transmis forma finală a tezei lui de doctorat -- *Tehnici avansate în proiectarea microcontrolerelor* - în care exploatează execuția multi-thread pentru a realiza eficient microcontrolere cu periferie virtuală, propunând un produs care combină flexibilitatea logicii programate cu utilizarea unor structuri standardizabile.

Relația dintre circuite, algoritmi și informație șȘtefan '96ț este din ce în ce mai mult reconsiderată atunci când dimensiunea circuitelor depășește cu multe ordine de mărime complexitatea pe care o putem controla eficient.

Proiectul mașinii DIALISP a fost declanșat de necesitatea de a rula eficient programe scrise într-un limbaj care permite uncontrol avansat al complexității. Conceptele pe care acest proiect le-a catalizat pot contribui la reechilibrarea relației dintre dimensiunea și complexitatea sistemelor de calcul, într-un moment în care creșterea complexități nu mai este corelată cu posibilitățile tehnologice.

Execuția sau interpretarea multi-thread, realizată la nivelul procesoarelor, este posibilă și ar putea oferi creșteri ale **performanțelor arhitecturale**. Preocupările actuale sunt concentrate prea mult asupra creșterii *performanțelor structurale* (pseudo-paralelisme de tip ILP, spre exemplu).

Creșterea dimensiunii memoriilor, păstrând actuala cărămidă de bază (celula dinamică de memorare), este exemplul cel mai flagrant de creștere dimensională la complexitate constantă. MC oferă o soluție pentru a antrena și creșterea complexității în procesul de creștere a dimensiunii memoriilor.

Silicon gap *este consecința decalajului prea mare dintre creșterea exponențială a dimensiunii*[7] *și creșterea nici macar constantă a complexității sistemelor digitale*.

Poate motivația cea mai profundă a proiectului DIALISP a fost de la început - și continuă să fie - încercarea, de ce nu și utopică, de echilibrare a dinamicii cu care evoluează dimensiunea și complexitatea sistemelor de calcul. Multă lume crede în așa ceva și există semnele unei reușite apropiate.

Bibliografie

[Arthur '90] Brian Arthur: “Positive Feedbacks in Economy" in *Scientific American*, 262, 92-99, Feb. 1990.

[Eco '84] Umberto Eco: *Numele trandafirului*, Editura Dacia, Cluj-Napoca, 1984.

[Fallows '01] James Fallow: “New Life for Moore’s Law", in *The Atlantic Monthly*, Oct., 2001.

[Gusfield '97] Dan Gusfield: *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*, Cambridge Univ. Press, 1997.

[Hennessy '92] John L. Hennessy, David A. Patterson: *Computer Architecture: A Quantitative Approach*, Second Edition, Morgan Kaufmann, 1992.

[7] “Legiferat`" \n 1965 de Gordon Moore -- cofondator al companiei *Intel*.

[Smith '78] B. J. Smith: "A Pipelined, Shared Resource MIMD Computer", in *Proc. of the 1978 Intl. Conf. on Parallel Processing*, pp. 6-8, August 1978.

[Stefan '84] Gheorghe Stefan, Aurel Paun, Andy Birnbaum, Virgil Bistriceanu: "DIALISP - a LISP Machine", in *Proceedings of the ACM Symposium on LISP and Functional Programming*, Austin, Texas, Aug. 1984. p. 123 - 128.

[Stefan '85] Stefan, G., Bistriceanu, V., Paun, A.:"Catre un mod natural de implementare a Lisp-ului", comunicare la *Al doilea Simposion Național de Inteligent Artificiala* organizat de Academia RSR, Sept. 1985; publicat in *Sisteme de Inteligenta Artificiala,* Ed. Academiei Române, 1991.

[Stefan '96] Stefan, G., Malitza, M. : "Chaitin"'s Toy-Lisp on Connex Memory Machine", in *Journal of Universal Computer Science*, vol. 2, no. 5, 1996, pp. 410-426.

[Stefan '97] Stefan, G., Malitza, M. : "The Splicing Mechanism and the Connex Memory", in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, Indianapolis, April 13 - 16, 1997. p. 225-229.

[Stefan '98] Stefan, G., Benea, R.: "Connex Memories î& Rewrieting Systems", in *MELECON '98*, Tel-Aviv, May 18 -20, 1998.

[Stefan '98a] Gheorghe Stefan: "The Connex Memory: A Physical Support for Tree / List Processing" in *The Roumanian Journal of Information Science and Technology*, Vol.1, Number 1, 1998, p. 85 - 104.

[Stefan '01] Gh. Stefan, Dominique Thiebaut: "Hardware-Assisted String-Matching Algorithms", in *WABI 2001, 1st Workshop on Algorithms in BioInformatics*, BRICS, University of Aarhaus, Danemark, August 28-31, 2001.

[Thornton '64] J. E. Thornton: "Parallel Operation in the Control Data 6600", in *AFIPS Conference Proceedings FJCC*, part 2, vol. 16, 1964.