

Proiect la Simularea si Optimizarea Arhitecturilor de Calcul

Autor: Barbu Paul – Gheorghe

Semigrupa: 242/1

Indrumator: Prof.univ.dr.ing. Adrian FLOREA

Enunt

Implementarea unui Automatic Design Space Exploration aferent unei arhitecturi superscalare (vezi simulatorul PSATSim – arhitectura PowerPC), pentru analiza multiobiectiv (performanță, consum de energie) folosind tehnici de optimizare Pareto: Multi Objective Particle Swarm Optimization.

Resurse necesare

Sistemul de operare: Windows, GNU/Linux

Pentru a rula aplicatia avem nevoie de:

- JRE 8.0
- simulatorul PSATSim, acesta necesita la randul sau bibliotecile:
 - libgtk+
 - zlib1
 - iconv
 - libxml2

Pentru a modifica aplicatia avem nevoie de:

- JDK SE 8.0
- Limbajul Scala 2.11.8
- Mediul de dezvoltare Scala IDE
- Biblioteca Scala XML 1.0.3
- Biblioteca JFreeChart 1.0.19
- Codul sursa al bibliotecii jMetal 4.5 (inclus deja in proiect)

Particle Swarm Optimization

Algoritmul de Particle Swarm Optimization este o tehnica din categoria “Artificial Life” prin care o populatie de indivizi urmareste optimul prin experienta colectiva cat si experienta individuala.

In cazul optimizarii unui procesor, dorim sa optimizam numarul de instructiuni rulate raportate la ciclul de procesor si energia consumata, deci dorim performanta maxima cu energie minima, asadar ne trebuie un algoritm pentru optimizare multiobiectiv (doua obiective in acest caz). Algoritmul PSO se aplica foarte bine pe probleme de optimizare multiobiectiv de minimizare.

Asadar primul nostru obiectiv, va trebui inversat, in loc de maximizarea IPC-ului (instructions per cycle), va trebui sa minimizam CPI (cycles per instruction). Acest lucru nu reprezinta o problema, deoarece $CPI = \frac{1}{IPC}$.

In linii mari algoritmul PSO, modeleaza miscarea unui stol de pasari sau al unui banc de pesti care urmaresc sa ajunga la o destinatie.

In mod plastic, in cazul optimizarii procesoarelor, vom defini “destinatia” ca fiind punctul din planul XOY unde ambele obiective, CPI si energia, sunt minime. Asadar, punctul (0, 0).

In pseudocod algoritmul ar fi:

```
initializare aleatoare a particulelor din populatie

cat-timp nr. maxim de iteratii nu a fost atins sau hipervolumul nu s-a modificat
semnificativ

    pentru fiecare particula
        evaluare particula
        daca locatia particulei este cea mai apropiata de obiectiv
            retinere configuratie particula ca personal-best
        sf-daca
    sf-pentru
    global-best = maxim personal-best
    pentru fiecare particula
        schimbare viteza in functie de personal-best si global-best
        schimbare locatie in functie de locatia anterioara si noua viteza
    sf-pentru
sf-cat-timp
```

Se poate observa ca fiecare particula tine cont atat de experienta celorlalte particule din populatie cat si de experienta proprie, datorita faptului ca viteza se schimba atat in functie de cea mai buna configuratie gasita la nivel global cat si personal.

In realitate, programul foloseste un algoritm PSO usor modificat: SMP SO, detaliat de autorii jMetal pe pagina: <http://jmetal.sourceforge.net/smpso.html>.

Pe scurt, pseudocodul pentru acest algoritm este:

```
1: initializeSwarm()
2: initializeLeadersArchive()
3: generation = 0
4: while generation < maxGenerations do
5:     computeSpeed()
6:     updatePosition()
7:     mutation() // Turbulence
8:     evaluation()
9:     updateLeadersArchive()
10:    updateParticlesMemory()
11:    generation ++
12: end while
13: returnLeadersArchive()
```

In urma studierii pseudocodului apare necesitatea de a echivala configuratia unei particule cu configuratia unui procesor. Acest lucru va fi descris in ghidul de dezvoltare, luand in considerare si notiunile din sectiunea urmatoare.

Despre simulatorul PSATSim

Fiind dificil sa implementam fizic sute, chiar mii de procesoare evaluate de-a lungul rularii algoritmului SMP SO se foloseste un simulator care ne ofera posibilitatea configurarii parametrilor procesorului si evaluarea acestuia din punct de vedere al IPC-ului si al energiei.

Dintre toti parametrii pe care ii ofera simulatorul exploratorul nostru va modifica doar:

Nume	Domeniu	Descriere
superscalar	Integers 1-16	Superscalar factor
rename	Integers 1-512	Register renaming table size
reorder	Integers 1-512	Reorder buffer size
RSB	centralized or hybrid or distributed	Reservation station architecture
Separate dispatch	true or false	True = decode and dispatch stages are separated.
vdd	Floating point greater than zero	Processor supply voltage. Should kept in the range of 1.8-3.3.
freq	Floating point greater than zero	Processor clock frequency, in MHz
integer	Integers 1-8	The number of integer functional units
floating	Integers 1-8	The number of floating point functional units
branch	Integers 1-8	The number of branch functional units
memory	Integers 1-8	The number of memory functional units

Restul de parametrii vor ramane constanti, precum se va arata in ghidul de dezvoltare.

Ghid utilizare

În urma compilării software-ului utilizatorul va trebui să pornească o fereastră de consolă, de unde va rula programul.

Comanda care trebuie folosită este:

```
java -jar cpuPso.jar --psatsim-name psatsim_con --psatsim-path /cale/spre/psatsim/ --swarm-size 5 --max-iterations 10 --archive-size 100
```

Aplicația ar trebui să fie utilizabilă atât pe Windows cât și pe GNU/Linux, iar semnificația parametrilor este următoarea:

1. psatsim-name este numele executabilului aferent simulatorului PSatSim, pe Windows “psatsim_con.exe”, iar pe GNU/Linux, “psatsim_con” (fără extensie).
2. psatsim-path este calea pe hard disk unde se găsește executabilul indicat cu parametrul anterior, exemplu: "E:/localhost/CpuParticleSwarmOptimization/psatsim/PSATSim".
3. swarm-size este numărul de indivizi ai populației de particule (cu alte cuvinte nr. de procesoare de simulat), pe procesoare mai slabe un număr mic este ideal, de exemplu 10, pe procesoare mai bune, putem mari acest număr chiar și la 50.
4. max-iterations este numărul maxim de populații pe care programul are voie să le genereze, acesta rulează oricum până când hipervolumul populațiilor nu mai evoluează semnificativ. Totuși dacă numărul maxim de iterații este atins, atunci explorarea se oprește. Dacă rulăm pe un sistem constrans dpdv. al resurselor și dorim să ne oprim după ce s-au generat câteva populații, atunci acest număr poate fi mic (exemplu: 50), dar dacă dorim să păstrăm condiția de oprire pe modificarea nesemnificativă a hipervolumului atunci este bine să folosim un număr mare, 1000, pentru a evita atingerea lui.
5. archive-size este un parametru specific jMetal și reprezintă numărul maxim de indivizi non-dominasiți de pe frontul Pareto care trebuie păstrați în timpul simulărilor, adică fișierele de output descrise mai jos nu vor putea conține mai mulți indivizi față de cât indică acest parametru. Un maxim pentru acest parametru îl reprezintă $\text{swarm-size} \times \text{max-iterations}$, deși este recomandat să fie păstrat la 100.

Output

În urma rularii cu succes a aplicației, se va crea un director cu data și ora curentă în formatul YYYY-MM-DD HH.MM.SS (ex. “2016-12-17 10.15.49”). S-a ales acest format datorită ușurinței sortării lui și a garanției unicității numelui, deoarece o simulare putem fi siguri că durează mai mult de o secundă. Acest director nu va fi creat în cazul unei rulari cu erori!

Conținutul directorului este:

- fișierul FUN – care reprezintă obiectivele atinse de cei mai buni indivizi la finalul explorării. În acest caz o linie din fișier corespunde unui individ non-dominat la finalul execuției. O linie constă din două numere de tip double care reprezintă CPI-ul unui procesor și energia

consumata de acesta. Aceste numere sunt de fapt agregate ca media aritmetica pe cele 10 trace-uri livrate cu simulatorul PSATSim.

- fisierul plot-hv.png reprezinta evolutia hipervolumului de-a lungul celor max-iterations sau pana cand acesta nu a mai evoluat semnificativ (la a 3a zecimala)
- fisierul plot-solutions.png este reprezentarea grafica a indivizilor non-dominati (frontul Pareto)
- SOMPSO.log este output-ul din program, care tine logul celor mai importante evenimente, cum ar fi valoarea parametrilor, timpul de rulare, valorile hipervolumelor de-a lungul explorarii, si motivul pentru care programul s-a oprit (fie ca a atins numarul maxim de iteratii, fie ca hipervolumul nu s-a mai modificat considerabil)
- fisierul VAR arata configuratia fiecarui individ non-dominat. Cu alte cuvinte, daca luam fiecare linie din acest fisier si folosim valorile in simulatorul PSatSim, acestea vor duce la obtinerea rezultatelor de pe aceeasi linie din fisierul FUN.

Fisierele FUN si VAR ar putea arata dupa cum urmeaza:

#	Continutul fisierului FUN		Continutul fisierului VAR										
	CPI	Energy	Superscalar	Rename	Reorder	RSB	Separate dispatch	VDD	Freq	Integer FU	Floating FU	Branch FU	Memory FU
1	1.040	2.480	14	473	3	2	1	1.929	1459	3	6	4	3
2	0.286	12.455	16	85	448	2	1	2.582	1403	4	5	6	5
3	0.565	4.395	13	27	17	2	1	3.116	1469	7	5	6	1
4	0.743	2.853	13	282	6	2	1	2.366	1461	3	5	4	2
5	0.323	7.738	15	164	261	2	1	2.448	1418	3	5	5	4
6	2.486	2.274	6	512	1	2	1	1.800	1570	1	6	1	1
7	0.323	8.283	15	152	286	2	1	2.456	1413	3	5	5	4
8	2.486	1.808	1	512	1	2	1	1.800	1646	1	6	1	1
9	0.351	7.421	14	175	266	2	1	2.416	1416	3	5	4	3

Ghid de dezvoltare

În ceea ce privește dezvoltarea aplicației de explorare folosind algoritmul Particle Swarm Optimization s-a folosit limbajul Scala (un dialect al Java) cât și limbajul de programare Java. Scala a fost ales din considerente de productivitate, acesta oferind un mod foarte ușor pentru lucrul cu fișiere XML (mai ales scrierea acestora).

Lucrul cu fișiere XML este important, deoarece pentru a automatiza simulatorul PSATSim sunt necesare fișiere de configuratie. Aceste fișiere de configuratie sunt în format XML. Scala ne permite să scriem și să parametrizăm fișiere XML similar cu scrierea codului normal, programatorul neobservând vreo diferență între codul care aplică logica aplicației, algoritmul, și partea de XML.

De exemplu:

```
private def getXml = {
  def generalNode(trace: String) =
    <general
      superscalar={_superscalar.toString}
      rename={_rename.toString}
      reorder={_reorder.toString}
      rsb_architecture={_rsb_architecture.toString}
      separate_dispatch={_separate_dispatch.toString}
      seed="0"
      trace={trace}
      output={output}
      vdd={"%1.2f".format(vdd)}
      frequency={freq.toString}
    />;

  // this XML will contain 10 general nodes, one for every trace file
  <psatsim>
    <config name={name}>
      {traces map(t => generalNode(t))}
      <execution
        architecture="standard"
        integer={integerFu.toString}
        floating={floatingFu.toString}
        branch={branchFu.toString}
        memory={memoryFu.toString}
      />
      <memory architecture="l2">
        <l1_code hitrate="0.990" latency="1" />
        <l1_data hitrate="0.970" latency="1" />
        <l2 hitrate="0.990" latency="3" />
        <system latency="20" />
      </memory>
    </config>
  </psatsim>
}
```

Această metodă privată a clasei PSATSimCfg se ocupă cu crearea (în memorie) a întregului fișier de configuratie necesar rulării simulatorului. Observăm în prima jumătate a metodei, o funcție

imbricata, care genereaza o configuratie generala pentru fiecare fisier de trace in parte. Aceasta functie este folosita in a doua parte a metodei pentru a popula nodul “config” al XML-ului.

Se poate observa modul in care parametrii precizati la sectiunea “Despre simulatorul PSATSim” sunt configurabili din exterior, iar restul, sunt constanti, acestia nefiind influentati de algoritm.

In urma generarii continutului acesta va fi scris pe disc de metoda “save” a aceleasi clase:

```
def save(path: String) = {  
    XML.save(Paths.get(path, name + ".xml").toString, getXml)  
}
```

O particula sau un individ creat de algoritmul PSO va fi o instanța a clasei CpuSolutionType. Instanța face maparea efectiva a unui procesor în reprezentarea specifica PSO.

```
override def createVariables(): Array[Variable] = {  
    val variables = new Array[Variable](11)  
    variables(0) = new Int(1, 16) // superscalar  
    variables(1) = new Int(1, 512) // rename  
    variables(2) = new Int(1, 512) // reorder  
    variables(3) = new Int(1, 2) // RSB  
    variables(4) = new Int(0, 1) // separate dispatch  
    variables(5) = new Real(1.8, 3.3) // vdd  
    variables(6) = new Int(10, 5000) // freq  
    variables(7) = new Int(1, 8) // interger FU  
    variables(8) = new Int(1, 8) // floating FU  
    variables(9) = new Int(1, 8) // branch FU  
    variables(10) = new Int(1, 8) // memory FU  
  
    variables  
}
```

In acest caz, o soluție este compusa din mai multe variabile, variabilele corespunzand caracteristicilor procesorului cu constrangerile date de domeniul lor de valori. Valoarea acestor variabile va fi folosită mai apoi pentru a calcula viteza și poziția indivizilor (prin simulare).

Este totusi necesar si limbajul Java deoarece biblioteca jMetal este scrisa in Java si a fost necesara modificarea acestei biblioteci pentru a satisface cerintele proiectului. Datorita faptului ca Scala este un dialect al Java, interactiunea cu cod Java din interiorul codului Scala nu a fost o problema. Motivul pentru care a fost necesara modificarea jMetal este faptul ca, in urma implementarii clasei CpuSolutionType, clasa XReal din pachetul jmetal.util.wrapper nu este capabila de a prelua informatia din noul tip de solutie. Clasa XReal stie sa citeasca doar dintr-un numar limitat de tipuri de solutii, acestea fiind hard-codate in interiorul clasei XReal. Acest lucru este probabil un defect al bibliotecii jMetal, dar nu reprezinta totusi o problema deoarece, avand sursele bibliotecii, defectul a fost remediat astfel:

```
public double getValue(int index) throws JMException {  
    if ((type_.getClass() == CpuSolutionType.class) ||  
        (type_.getClass() == RealSolutionType.class) ||  
        (type_.getClass() == BinaryRealSolutionType.class)) {  
        return solution_.getDecisionVariables()[index].getValue();  
    }  
}
```

```

        else if //...
        //...
        return 0.0 ;
    }

```

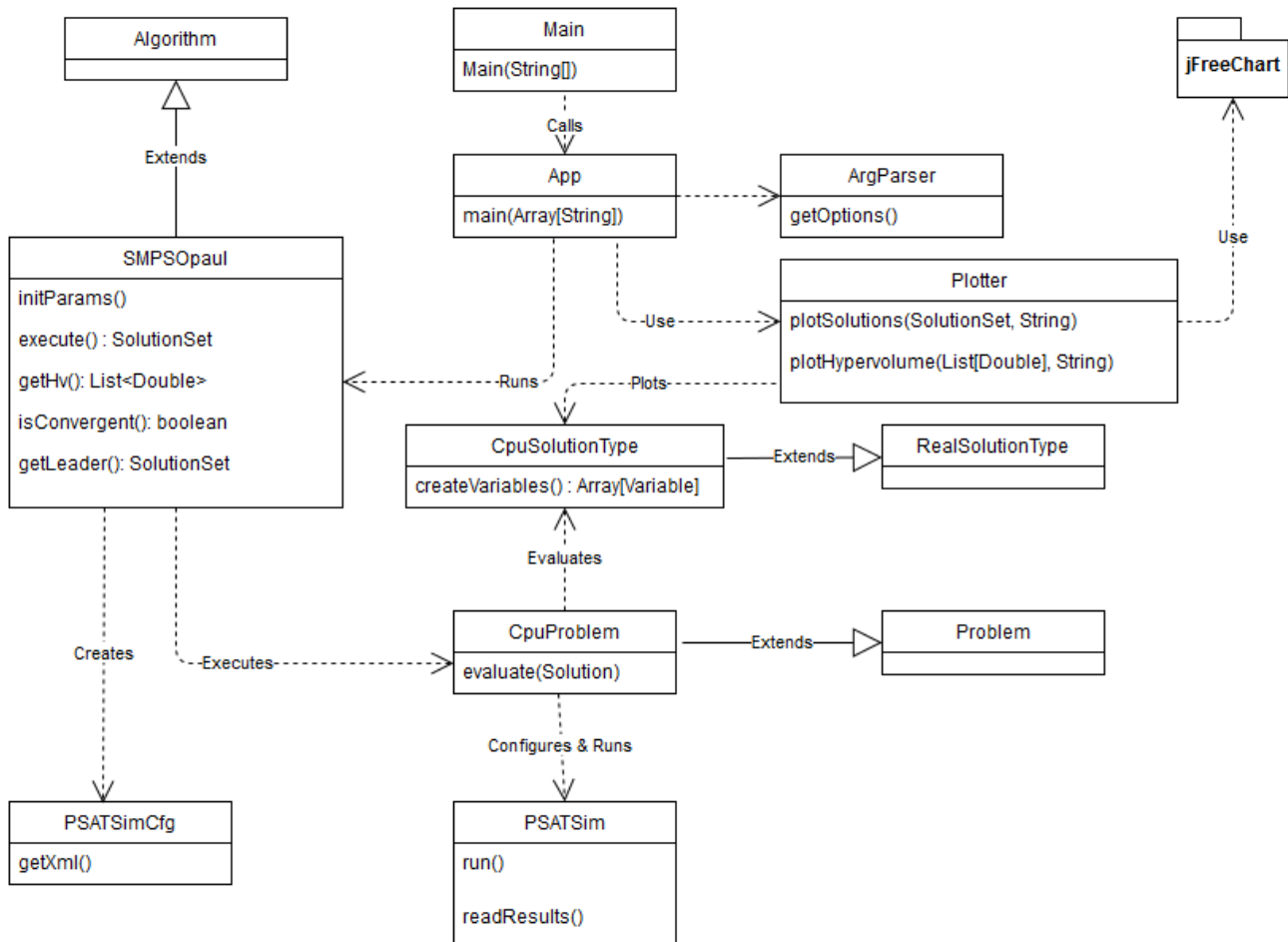
De asemenea tot la categoria “modificari aduse bibliotecii jMetal” ar putea intra si implementarea clasei „SMPSOpaul”, care aduce în plus conditia de oprire atunci când hipervolumul unei populatii nu mai difera semnificativ fata de hipervolumul anterior. În acest caz „nesemnificativ” reprezintă schimbari ale ariei dincolo de a treia zecimala ($\epsilon = 0.001$).

Pentru a desena graficele (plot-solutions.png și plot-hv.png) s-a folosit biblioteca jFreeChart, și aceasta fiind implementata tot în Java. Aceasta biblioteca este folosită în clasa Plotter. În aceasta clasa se creaza și se salvează graficele pe disc. Tot aici se normalizeaza datele pentru graficul hipervolumului, toate ariile fiind raportate la aria maxima.

Așadar rezultatele sunt generate și salvate automat în urma procesului de explorare prin intermediul clasei Plotter (pentru grafice) și a clasei SMPSOpaul (care este doar clasa SMPSO din jMetal cu adaugirile mentionate mai sus).

Paralelismul în aceasta aplicație este exploatat la nivelul simulatorului PSATSim. Acesta este capabil de a rula doua sau mai multe configuratii de procesor în paralel. Așadar, când se evalueaza un individ, simulatorul ruleaza trace-urile în paralel, reducandu-se astfel timpul de rulare și exploatandu-se la maxim resursele hardware ale sistemului de calcul.

Pentru a aduce împreuna conceptele prezentate sumar anterior se prezinta în cele ce urmează diagrama UML a proiectului (clasele goale apartin jMetal):



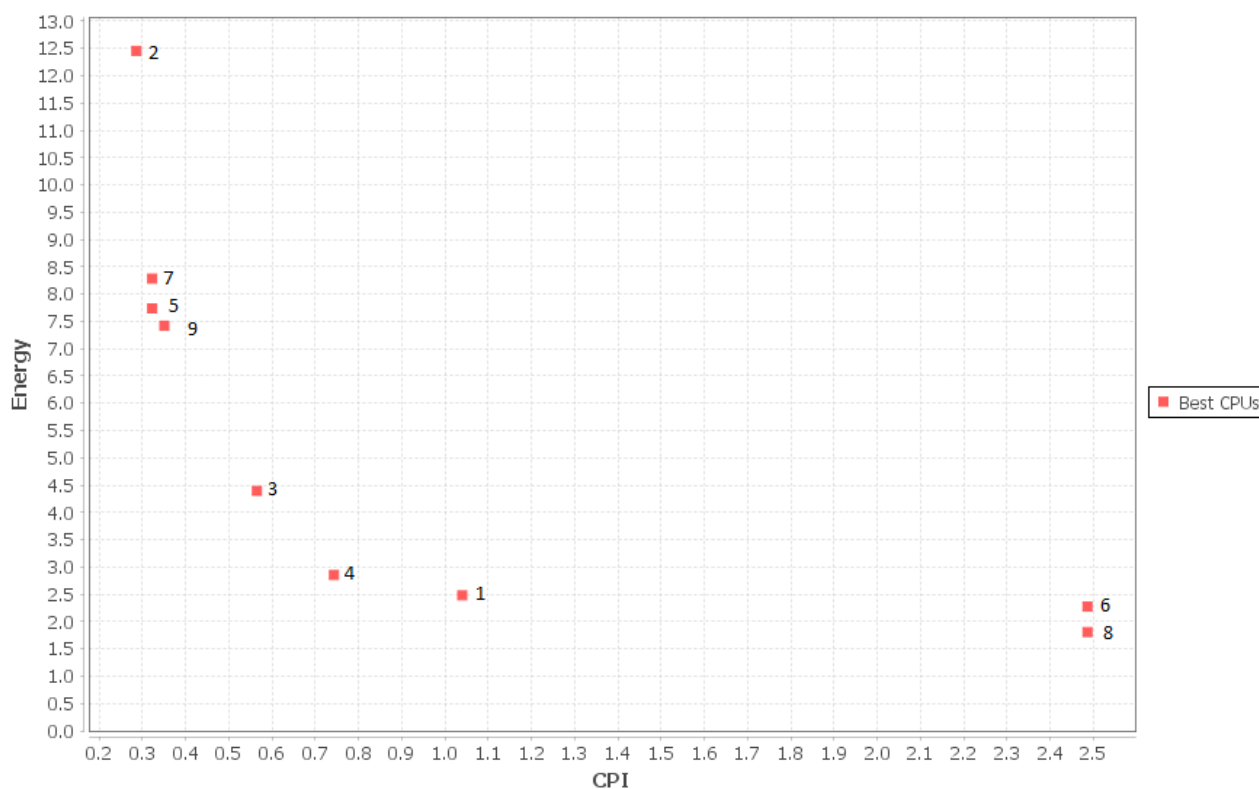
Se poate vedea în diagrama de mai sus că există câteva puncte cheie în arhitectura exploratorului automat care pot fi modificate sau înlocuite, în funcție de nevoi:

- Dacă se dorește schimbarea algoritmului, atunci putem înlocui clasa SMPSOpaul cu o altă clasă care implementează alt algoritm, este important ca această nouă clasă să mostenească clasa Algorithm din jMetal.
- Pentru a modifica problema careia să îi găsim optimul, trebuie înlocuite clasele CpuSolutionType și CpuProblem, care trebuie să extindă obligatoriu o clasă din pachetul jmetal.encodings.solutionType și respectiv clasa Problem (din pachetul jmetal.core). Eventual modificarea problemei poate atrage după sine și un nou simulator, caz în care trebuie înlocuite clasele PSATSim și PSATSimCfg.
- În cazul în care se doresc alte tipuri de grafice, trebuie modificată clasa Plotter.

Concluzii și interpretari

În urma unor versiuni initiale ale programului (înainte de implementarea condiției pe hipervolum), după 18 minute de rulare, 11 generații și 11 interații ale algoritmului PSO s-a obținut acest rezultat:

Pareto front



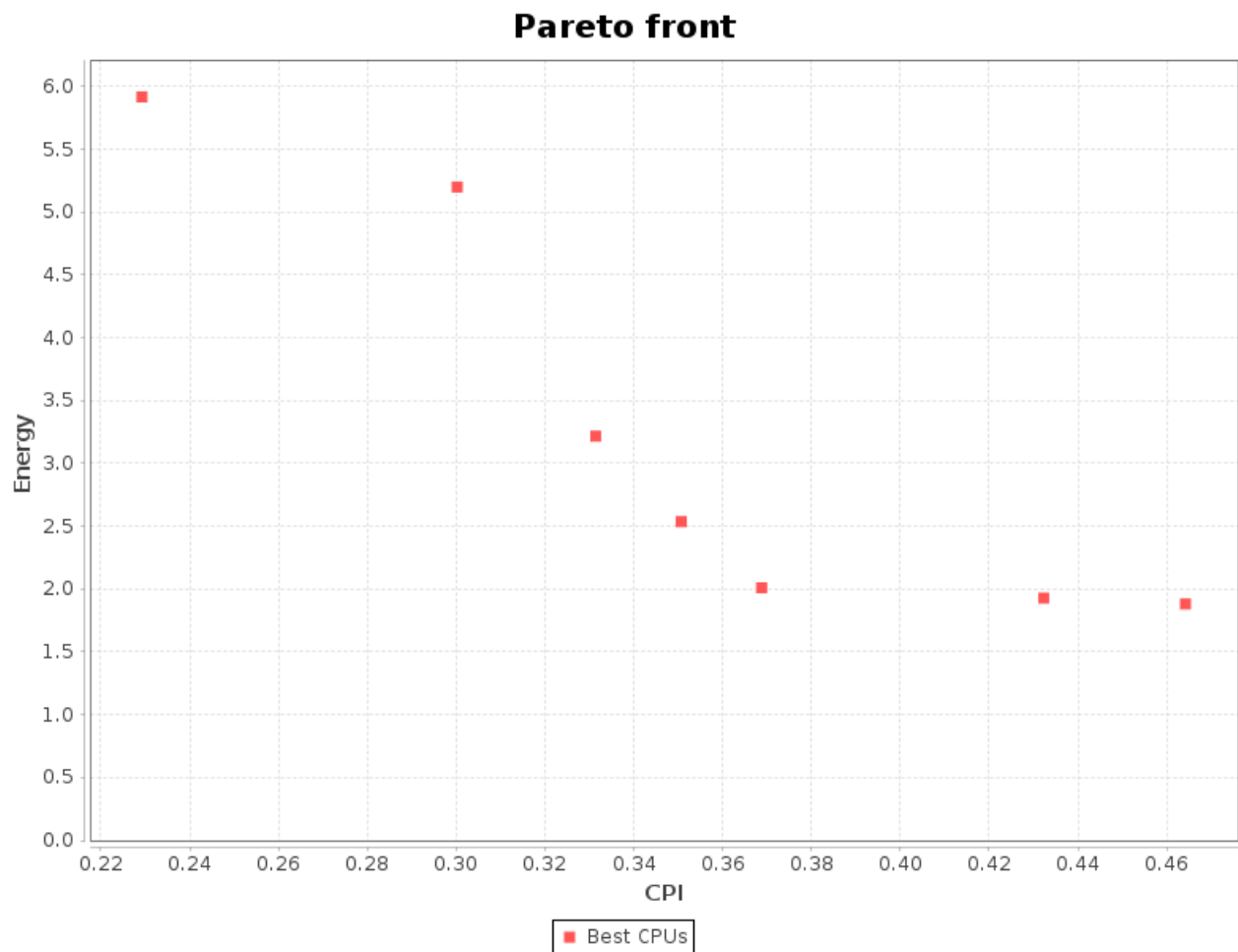
Verificand fișierul VAR, deducem ca procesoarele noastre, găsite în urma explorării automate folosind particle Swarm Optimization au următoarele configurații:

#	Superscalar	Rename	Reorder	RSB	Separate dispatch	VDD	Freq	Integer FU	Floating FU	Branch FU	Memory FU
1	14	473	3	2	1	1.929	1459	3	6	4	3
2	16	85	448	2	1	2.582	1403	4	5	6	5
3	13	27	17	2	1	3.116	1469	7	5	6	1
4	13	282	6	2	1	2.366	1461	3	5	4	2
5	15	164	261	2	1	2.448	1418	3	5	5	4
6	6	512	1	2	1	1.800	1570	1	6	1	1
7	15	152	286	2	1	2.456	1413	3	5	5	4
8	1	512	1	2	1	1.800	1646	1	6	1	1
9	14	175	266	2	1	2.416	1416	3	5	4	3

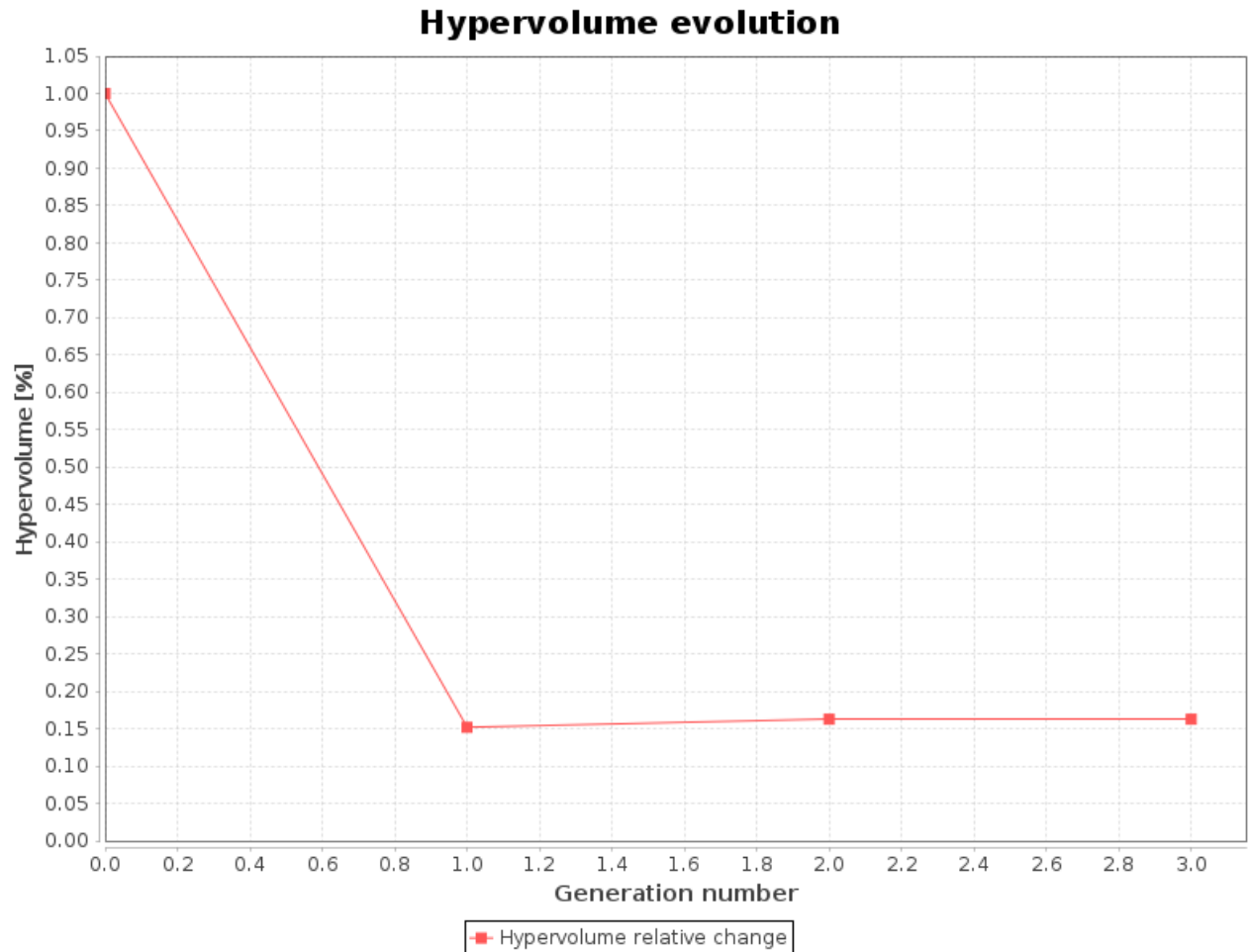
Urmărind liniile din fișierul FUN vedem următoarele valori pentru cele doua obiective optimizate:

#	CPI	Energy
1	1.03971206	2.480243
2	0.2860141291	12.45488
3	0.5652037391	4.394594
4	0.7433339221	2.853436
5	0.3228312836	7.73752
6	2.4861783131	2.273825
7	0.3227228404	8.283152
8	2.4861891932	1.808303
9	0.3507120139	7.421373

Mai jos putem vedea frontul Pareto, cât și evoluția hipervolumului în ultima versiune a programului:



TODO: dif cu și fără hipervolum



Deși inițial programul a început pe picior greșit, aria hipervolumului este extrem de mare (ca urmare a generării unui individ departe de optimul dorit), a recuperat apoi dezavantajul, în doar 3 generații reușind să creeze indivizi care nu mai îmbunătățesc semnificativ aria hipervolumului.

TODO: HV ca log, PyGMO

Alte documente:

- Documentatia jMetal se poate gasi la adresa:
<http://jmetal.sourceforge.net/javadoc/index.html>
- Ghid jMetal:
<https://sourceforge.net/projects/jmetal/files/jmetal4.5/jmetal4.5.userManual.pdf/download>
- PSATSim:
<http://homepages.udayton.edu/~ttaha1/psatsim/>
- Documentatie PSATSim:
[/calea/catre/directorul/PSATSim/cli_usage.html](http://calea/catre/directorul/PSATSim/cli_usage.html)
- Tutorial PSO:
<http://www.swarmintelligence.org/tutorials.php>
- PSO interactiv:
<https://ccl.northwestern.edu/netlogo/models/ParticleSwarmOptimization>