UDC: 004.43 INFO M: str. 18-22

# JAVA FORK/JOIN PARALELNI GAUSS-SEIDEL U AMAZON EC2 OBLAKU (JAVA Fork/Join PARALLEL GAUSS-SEIDEL IN AMAZON EC2 CLOUD)

Miloš Čubrilo, Slobodan Jovanović
Univerzitet Metropolitan, Fakultet informacionih tehnologija, Beograd,
www.metropolitan.edu.rs, milos.cubrilo@gmail.com, slobodan.jovanovic@metropolitan.ac.rs

REZIME: Od svoje prve verzije Java podržava paralelno programiranje uporebom „niti" (engl. threads) koje su deo Java jezika od njegovog nastanka. Na ovom polju, Java Standard Edition 7 (Java SE 7) donosi novine u vidu novog razvojnog okvira Java Fork/Join (Java F/J) visokog nivoa, za jednostavniju implementaciju performantnih paralelnih programa. U radu je prikazana implementacija paralelnog Gauss-Seidel algoritma, uz pomoć modernog programskog jezika Java SE 7 i novog razvojnog okvira Java Fork/Join. Pored konstruisanja paralelnog Gauss-Seiel algoritma, prikazana je i upotreba računarskog oblaka Amazon EC2 za izvršavanje paralelnog Gauss-Seidel programa. Izvršeno je merenje performansi paralelnog Gauss-Seidel algoritma, uz pomoć generisanih linearnih sistema različitih veličina i osobina upotrebom MATLAB softvera. Testirani su sistemi veoma velikih dimenzija, 10,000x10,000, i dobijena su odlična vremena izvršenja softvera (manjim od 1 sekunde) u računarskom oblaku Amazon EC2, što preporučuje računarski oblik za paralelno procesiranje linearnih sistema velikih dimenzija.
KLJUČNE REČI: Niti, Java DCL, Java SE 7, MATLAB
ABSTRACT: From its 1st version, the Javalanguage supportsparallel programming using „ threads", which are the part of Java language from the beginning. In this field, Java Standard Edition 7 (Java SE 7) brings new opportunities, like the new high-level framework Java Fork/Join, for simpler implementations of high-performance parallel programs. This paper presents an implementation of the parallel Gauss-Seidel algorithm, using of a modern programming language Java SE 7 and a new framework Java Fork/Join. Besides conscruncting the parallel Gauss-Seidel algorithm, the papers explains using the Amazon EC2 cloud for the execution of parallel Gauss-Seidel program. Performancecharacteristics of the parallel Gauss-Seidel algorithm have been easured, while the MATLAB software have been exploited to generatelinear sistems of different size and properties. Big linear sistems (dimension 10,000x10,000) have been tested, and good execution times obtained (less than 1 second) in Amazon EC2 cloud, which reccomends cloud computing forparallel processing of big linear sistems.
KEY WORDS:Threads, Java DCL, Java SE 7, MATLAB

## 1. UVOD

U vreme kada gotovo svaki računar poseduje višejezgarni procesor, paralelno programiranje postaje ključna tehnika za poboljšanje performansi mnogih aplikacija **[1-4].** Moderni programski jezici koji prate ovaj trend, npr. Java SE 7, poseduju niz biblioteka nižeg i višeg nivoa koje omogućavaju implementaciju efikasnih paralelnih programa. Od svoje prve verzije Java podržava paralelno programiranje uporebom „niti" (engl. *threads*) koje su deo Java jezika od njegovog nastanka. Na ovom polju, Java Standard Edition 7 (Java SE 7) donosi novine u vidu novog razvojnog okvira Java Fork/Join (Java F/J) visokog nivoa, za jednostavniju implementaciju performantnih paralelnih programa **[5].** U poslednje vreme, za potrebe paralelnih programa sve više se koriste računarski oblaci, pružajući prednosti koje nudi procesiranje u oblaku i nudeći nove izazove **[6,7].**

U ovom radu, problem koji je potrebno rešiti je paralelizacija Gauss-Seidel iterativnog numeričkog metoda, uz pomoć najnovijih alata koje nudi Java SE 7 za kreiranje paralelnih programa. Dakle, tema rada je konstukcija i implementacija paralelnog Gauss-Seidel iterativnog metoda u Javi SE 7, uz upotebu Java F/J okvira. I pri tome, za izvršavanje paralelizovanog softvera koristi se Amazon EC2 oblak, i analiziraju se linearni sistemi veoma velikih dimenzija.

## 2. PARALELNI GAUSS-SEIDEL ALGORITAM

Gauss-Seidel metoda je dobro poznata iterativna numerička metoda za rešavanje sistema linearnih jednačina:

$$Ax = b, \quad (1)$$

koja glasi:

$$x^{(k+1)} = -(D+L)^{-1}Ux^{(k)} + (D+L)b, \quad (2)$$

gde je: A matrica, a D i L i U su komponente matrice A (A=D+L+U), i to D je dijagonalna matrica, a L i U su donja i gornja trougaona matrica, respektivno. Pri tome, je "b" zadati vektor b=[b1, b2.....], a „x" je nepoznat vektor x=[x1, x2, .....].

Na osnovu jednačine (2), može se konstruisati paralelni Gauss-Seidel algoritam, koji se pomoću pseudo-koda može predstaviti na sledeći način**:**

```
SIZE: Velicina matrice
A[SIZE][SIZE]: Matrica koeficijenata
B[SIZE]: Vektor desne strane
X[SIZE]: Privremeni vektor resenja
SOLUTION[SIZE]: Vektor resenja
ERROR: Indikator greske
EPSILON: Zahtevana preciznost
WORKERS: Broj procesorskih jedinica
BATCHSIZE: Velicina pojedinacnog segmenta za obradu
LOCALERROR: Lokalna greska
```

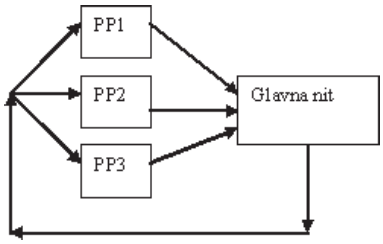
```
GLOBALERROR[WORKERS]: Globalni niz gresaka za sve pro-
cesore

BATCHSIZE ← SIZE/WORKERS
repeat
for all k=0 to k=WORKERS-1 do
if k=WORKERS-1 then
upper ← SIZE
else
upper ← (k+1)*BATCHSIZE
end if
for i=k*BATCHSIZE to i=upper-1 do
sum ← 0
for j=0 to j=SIZE-1 do
if i!=j then
sum ← A[i][j]*X[j]
end if
SOLUTION[i] ← (B[i]-sum)/A[i][i]
End for
LOCALERROR ← max(LOCALERROR,abs(X[i]-SOLUTION[i]))
X[i] ← SOLUTION[i]
end for

GLOBALERROR[k] ← LOCALERROR
end for all
for i=0 to i=WORKERS-1 do
ERROR ← max(ERROR,GLOBALERROR[i]))
end for
until (ERROR<EPSILON)
```

Pri tome, algoritam uzima kao ulazne podatke matricu koeficijenata A, vektor desne strane „b“, i zahtevanu preciznost rešenja EPSILON. Sukcesivne aproksimacije rešenja se obavljaju dok se ne zadovolji zahtevana preciznost EPSILON. Rezultat se čita iz vektora SOLUTION.

Takodje, ulazni podatak je parametar WORKERS, koji predstavlja broj procesora koji će paralelno izvršavati algoritam. U zavisnosti od vrednosti ulaznog parametra WORKERS, paralelni Gauss-Seidel algoritam deli linearni sistem na blokove koji se šalju na procesiranje pojedinim procesorima. Svaki od paralelnih procesora nakon izvršenja proračuna ažurira svoj deo vektora rešenja, i računa iznos greške aproksimacije LOCALERROR za svoj pripadsajući blok. Za izračunavanje „globalne“ greške ERROR, potrebno je da svi paralelni procesori prvo završe svoje proračune i zatim objave svoje lokalne greške preko niza GLOBALERROR[k]. Proračun ERROR se implementira sekvencijalno a ne paralelno, i odvija se pošto svi paralelni procesori završe svoje proračune. Glavna „nit“ programa pronalazi maksimum iz niza GLOBALERROR[k], i pokreće novu iteraciju ukoliko nije zadovoljena tražena preciznost. Na slici 1 je prikazana organizacija rada algoritma preko paralelnih procesora i glavne „niti“ programa.




**Slika 1.** – *Paralelni procesori PP1, PP2,PP3,...., i glavna „nit“ programa*

### 3. JAVA FORK/JOIN IMPLEMENTACIJA PARALELNOG GAUSS-SEIDEL ALGORITMA

Paralelni Gauss-Seidel algoritam implementiran je koristeći programski jezik Java 7 i novi Java Fork/Join razvojni okvir, koji je dostupan od Java 7 verzije Jave jezika. Od prve vezije Java jezika (Java 1) postoje klasične paralelne „niti“, dok u Java 5 verziji imamo sinhronizacione tehnike (Java CDL tj. Java CountDownLatch). Klase koje su korišćene u Java 7 F/J implementaciji paralelnog Gauss-Seidel algoritma su novina u Java 7 API**.**

Java Fork/Join razvojni okvir umesto direktnog rada sa „nitima“ koristi apstrakcije „akcije“ (*action*) i „zadatka“ (*task*). Prvi korak je kreiranje tzv. Pool-a (objekat tipa ForkJoinPool) odredjene veličine, gde ova veličina odredjuje koliko je moguće „zadataka“ izvršavati istovremeno, a u našem slučaju je to broj raspoloživih paralelnih procesora na sistemu (parametar WORKERS). Program zatim kreira „akciju“, koja rekurzivno deli problem na manje delove („zadatke“) i prosledjuje ih na izvršenje Pool objektu.

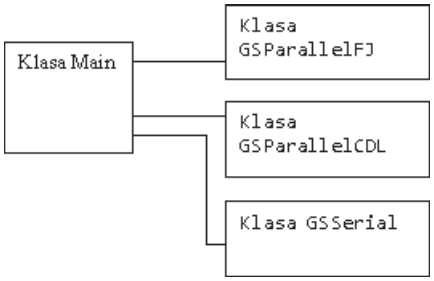
Dole je dat listing Java 7 F/J implementacije paralelnog Gauss-Seidel algoritma (klasa `GSParallelFJ`):

```
package rs.edu.fit.si.parallel.impl;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;
import rs.edu.fit.si.parallel.IterativeMethod;
public class GSParallelFJ extends IterativeMethod{
private final int workers;
private float[] globalNorm;
public GSParallelFJ(String inputMatrix, int workers) {
super(inputMatrix);
this.workers = workers;
this.globalNorm = new float[workers];
java.util.Arrays.fill(globalNorm, 0);
}
public synchronized void updateNorm(int idx, float val)
{
this.globalNorm[idx] = val;
}
public void solve() {
ForkJoinPool resourcePool = new ForkJoinPool(workers);
GSWorker worker;
double norm;
do{
```

```
worker = new GSWorker(--1,0);
// Perform parallel calculations
resourcePool.invoke(worker);
worker.join();
//At this point all worker calculations are done!
norm = 0;
for (float d: globalNorm) if (d>norm) norm = d;
if (norm<epsilon)
errorFlag = false; // Converged
else
errorFlag = true; //No desired convergence
} while (norm > epsilon && ++iter <= maxIter);
}
}

class GSWorker extends RecursiveAction {
private static final long serialVersionUID = 1L;
private final int idx;
private final int batchSize;
GSWorker(int idx, int batchSize){
this.idx = idx;
this.batchSize = batchSize;
}
@Override
protected void compute() {
if(idx == -1) { //Divide job to number of available
cores
int batchSize = data.size/workers;
List<GSWorker> tasks = new ArrayList<GSWorker>();
for(int i=0;i<workers;i++) tasks.add(new
GSWorker(i,batchSize));
invokeAll(tasks);
} else { // Worker computation
int upper = idx == workers-1? data.size :
(idx+1)*batchSize;
float localNorm = 0, diff = 0;
// For every equation in batch
for(int j=idx*batchSize;j<upper;j++) {
float s=0; // For every variable in equation
for(int i=0;i<data.size;i++) {
if (I != j) s+= data.A[j][i] * data.x[i];
data.solution[j]=(data.b[j]--s)/data.A[j][j];
}
diff = Math.abs(data.solution[j] - data.x[j]);
if (diff > localNorm) localNorm = diff;
data.x[j] = data.solution[j];
}
globalNorm[idx] =localNorm;
}
}
}
```

Na donjoj slici (slika 2) prikazan je Java 7 program, pomoću kojeg je implementiran paralelni Gauss-Seidel algoritam. Ovj program obuhvata glavnu klasu, klasu Main.java, koja poziva po potrebi klasu GSParallelFJ.java (paralelni Gauss-Seidel napisan pomoću Java 7 F/J). Takodje, program obuhvata i klasu GSSerial.java koji je implementacija sekvencijalnog (tj. klasičnog „serijskog“ ) Gauss-Seidel algoritma, koji se koristi za potrebe poredjenja brzine izvršavanja sekvencijalnog i paralelnog GS (Gauss-Seidel) algoritma. Isto tako, implementirana je i klasa GSParallelCDL.java, koja predstavlja implementaciju paralelnog Gauss-Seidel algoritma pomoću Java CDL tehnike.




**Slika 2.** – *Java SE 7 program za paralelni Gauss-Seidel algoritam*

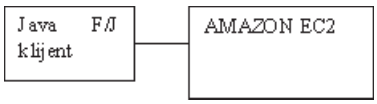
## 4. JAVA F/J PARALELNI GAUSS-SEIDEL U AMAZON OBLAKU

Za izvršavanje Java F/J paralelnog Gauss-Seidel programa, upotrebljen je Amazon EC2 računarski oblak (slika 3). Uz pomoć obezbeđenog menadžment interfejsa, kreirana je virtuelna „instanca“ tipa *High-CPU extra large,* sa sledećim karakteristikama:

- 2xQuadCore Intel Xeon CPU E5506 @ 2.13GHz (8 logičkih procesora)
- 7GB RAM
- 1690 GB Storage
- RHEL(RedHat Enterprise Linux) 6.2 64-bit OS
- JRE 1.7.0, Java HotSpot 64-Bit VM

Korišćenje" instance" se tarifira samo kada je ista u upotrebi, i košta 0.78$ po satu za „instancu" sa navedenim osobinama. Prijava i kreiranje „instance" se ne naplaćuje.

Ovo čini Amazon EC2 veoma pogodnom platformom za izvršavanje paralelnih izračunavanja na zahev. U roku od nekoliko časova od prijave na platformu, dobija se pristup virtuelnoj mašini sa željenim hardverskim resursima i operativnim sistemom. Nakon izvšenih proračuna, mašina se može stopirati, i plaćaju se samo sati potrošeni tokom izvršenja programa. Program je kompajliran i zapakovan u datoteku parallel.jar, i zatim izvršen uz pomoć skripte run.sh.




**Slika 3.** – *Računarski oblak AMAZON EC2 povezan sa Java F/J klijentom*

## 5. EVALUACIJA REZULTATA

Kao pokazatelj peformansi uzeto je vreme izvršenja svake od paralelnih implementacija Gauss-Seidel algoritma (Java F/J i Jva CDL), kao i „ubrzanje“ gde „ubrzanje“ predstavlja odnos vremena izvršavanja paralelnog Gauss-Seidel algoritma u odnosu na sekvencijalni Gsuss-Seidel algoritam. Treba napomenuti da su vremena izvršenja primetno varirala od izvršenja do izvršenja, zbog variranja u raspoloživosti resursa

Amazon platforme. U donjoj tabeli dat je primer izmerenih rezultata za matricu dimenzije 10,000x10,000, gde je primenjen broj niti jednak 6 (6 paralelnih procesora).

**Tabela 1:** *Primer izmerenih rezultata za sistem 10,000x10,000 i broj niti je 6*

| Implementacija | Br. Niti | N | Diag. dom. | Gustina | Br. Iter. | Vreme (ms) | Ubrzanje |
|---|---|---|---|---|---|---|---|
| Cdl | 6 | 10000 | 1.01 | 0.1 | 7 | 772.20 | 2.41 |
| Cdl | 6 | 10000 | 1.01 | 0.5 | 7 | 836.94 | 2.22 |
| Cdl | 6 | 10000 | 1.1 | 0.1 | 6 | 884.52 | 2.10 |
| Cdl | 6 | 10000 | 1.1 | 0.5 | 7 | 821.62 | 2.26 |
| Cdl | 6 | 10000 | 2 | 0.1 | 5 | 599.23 | 2.71 |
| Cdl | 6 | 10000 | 2 | 0.5 | 5 | 601.56 | 2.33 |
| **Java CDL srednje ubrzanje** | | | | | | | **2.34** |
| Fj | 6 | 10000 | 1.01 | 0.1 | 7 | 761.79 | 2.44 |
| Fj | 6 | 10000 | 1.01 | 0.5 | 6 | 740.81 | 2.50 |
| Fj | 6 | 10000 | 1.1 | 0.1 | 7 | 741.52 | 2.50 |
| Fj | 6 | 10000 | 1.1 | 0.5 | 7 | 728.16 | 2.55 |
| Fj | 6 | 10000 | 2 | 0.1 | 6 | 657.69 | 2.47 |
| Fj | 6 | 10000 | 2 | 0.5 | 5 | 555.16 | 2.52 |
| **Java F/J srednje ubrzanje** | | | | | | | **2.50** |

Izračunavanja su vršena na sistemima linearnih jednačina veličine 1000x1000, 5000x5000 i 10000x10000 koji zadovoljavaju uslove konvergencije Gauss-Seidel algoritma. Svaka od navedenih veličina sistema generisana je u više verzija, sa različitim stepenom dominantnosti dijagonale (1.01, 1.1 i 2). Takođe, svaki sistem je generisan i u dve različite gustine kako bi se ispitalo ponašanje implementacija kada je različit procenat ne-dijagonalnih elemenata jednak nuli. Korektnost rešenja sistema je potvrđena upoređivanjem dobijenih rezultata sa rezultatima MATLAB funkcije *linsolve* sačuvanim u datotekama sa sufiksom *sol.txt,* kreiranim pri generisanju testiranih linearnih sistema. Sistemi su generisani uz pomoć programskog paketa *MATLAB,* i sačuvani u tekstualne datoteke čiji su nazivi prosleđivani programu kao argument. U tabeli 2 je dat izvorni kod u MATLAB-u, za generisanje testiranih sistema. Variranjem promenljivih *n*, *scale i density* generisane su datoteke sa različitom veličinom sistema, stepenom dominantnosti elemenata na glavnoj dijagonali i gustinom. Program je konfigurisan tako da prekine izvršenje u sledećim slučajevima:

a) Kada dostigne maksimalan broj iteracija (maxIter=1000)
b) Kada dostigne željena preciznost rešenja (epsilon=1.0E-4)

MATLAB je dobro poznat program, koji se puno koristi kod raznih naučnih proračuna, i vrlo je koristan kod analiza linearnih sistema velikh dimenzija **[8].**

**Tabela 2:** *Izvorni kod u MATLAB-u za generisanje testiranih sistema:*

```
sizes = [1000, 5000, 10000];
scales = [1.01, 1.1, 2];
densities = [0.1, 0.5];

for i=1:numel(sizes)
    size = sizes(i);

    for j=1:numel(scales)

        scale = scales(j);

        for k=1:numel(densities)

            density = densities(k);

            Atmp = full(sprand(size,size,density));
            A = Atmp + diag(sum(abs(Atmp)*scale,2));
            b = rand(size,1);
            x = linsolve(A,b);

sys = [A b];

filename =
strcat('data_',num2str(size)
,'_',num2str(scale)
,'_',num2str(density));

            filename = strcat('X:\tmp\',filename,'_.txt');

dlmwrite(filename, sys, 'delimiter','\t','newline', 'pc');
            dlmwrite(strcat(filename,'-sol.txt'), x
                ,'delimiter','\t','newline', 'pc');

        end
    end
end
```

Ako se prouče rezultati u tabeli 1, može se uočiti da je primenom 6 paralelnih procesora postignuto ubrzanje oko 2.5. Što pokazuje da se paralelizacija pokazala vrlo korisnom, tj. da primenom paralelnog algoritma mogu se postići značajna ubrzanja u odnosu na sekvencijalni algoritam. S obzirom da se upotreba oblaka EC2 (Elastic Compute Cloud) plaća po vremenu upotrebe, npr. 0.78$ po satu, jasno je da što je veća brzina programa to se smanjuju troškovi upotrebe oblaka. U tom smislu, dalji razvoj programa koji bi omogućio upotrebu većeg broja procesora je interesantan. Medjutim, treba uzeti u obzir da upotreba većeg broja procesora i više košta, tako da ekonomičnost rešenja zavisi ne samo od brzine procesiranja već i od broja primenjenih procesora, tako da efekat upotrebe paralelnih procesora na ekonomičnost je znatno manji. Pretragom na Web-u nije nadjen nijedan rad koji analizira primenu paralelne Gauss-Seidel metode uz istovremenu upotrebu EC2 oblaka, tako da je ovaj rad gde se opisuju iskustva primene Java Fork/Join tehologije uz primenu EC2 oblaka koristan, jer opisuje upravo takvo iskustvo. U radu [6] se primenjuje paralelno procesiranje u oblaku, i dolazi se do zaključka da se postignute veće brzine primenom paralelnih procesora, ali da ekonomičnost nije bitno povećana. Ipak u radu [6] se ne analizira paralelni Gaus-Seidel. U radu [3] je analizirana upotreba paralelnih procesora kod Gaus-Seidel metode , ali bez upotrebe oblaka EC2, i npr. postignuto je za 4 paralelna procesora da je ubrzanje oko 1.8, što je slično sa

rezultatima koji su dobijeni ovde pomoću Java Fork-Join tehnologije i EC2 oblaka (ubrzanje 2.5 za 6 paralelnih procesora), ali pri čemu treba napomenuti da ubrzanje značajno zavisi od dimenzije sistema, i od zahtevane tačnosti, i od detalja samog algoritma, kao i od primenjene tehnologije (Java, itd.).

## 6. ZAKLJUČAK

U radu je prikazana implementacija paralelnog Gauss-Seidel algoritma, uz pomoć modernog programskog jezika Java SE 7 i novog razvojnog okvira Java Fork/Join. Pored konstruisanja i implementacije paralelnog Gauss-Seiel algoritma, prikazana je i upotreba računarskog oblaka Amazon EC2 za izvršavanje paralelnog Gauss-Seidel programa. Izvršeno je merenje performansi paralelnog Gauss-Seidel algoritma, uz pomoć generisanih linearnih sistema različitih veličina i osobina upotrebom MATLAB softvera. Testirani su sistemi veoma velikih dimenzija, 10,000x10,000, i dobijena su odlična vremena izvršenja softvera (manjim od 1 sekunde), i ubrzanja oko 2.5 u odnosu na sekvencijalni algoritam, što preporučuje računarski oblik za paralelno procesiranje linearnih sistema velikih dimenzija.

Doprinos ovog rada je što detaljno opisuje i objašnjava kako se može primeniti Java Fork/Join tehnologija kod problema rešavanja velikih sistema linaernih jednačina pomoću paralelne Gauss-Seidel metode, i takodje što rad iznosi rezultate i iskustva upotrebe oblaka EC2 u ovom slučaju. Tako da ovaj rad daje korisna uputstva i rezultate, korisna za razvoj ili primenu sličnih aplikacija, jer naime, pretragom na Web-u, nije nadjen rad koji analizira primenu paralelne Gauss-Seidel metode uz istovremenu upotrebu EC2 oblaka, niti nisu nadjeni na Web-u rezultati primene Java Fork/Join kod paralelne Gauss-Seidel metode.

## ZAHVALNICA:

Ovaj rad podržan je od strane Ministarstva za nauku i obrazovanje Srbije (Projekat III44006).

## LITERATURA:

[1] Jacques Mohcine Bahi, Sylvain Contassot-Vivier, Raphaël Couturier -"Parallel Iterative Algorithms – from sequential to grid computing", Chapman & Hall 2008.
[2] David Ronald Kincaid, Elliott Ward Cheney - "Numerical analysis - mathematics of scientific computing", American Matehematical Society 2001.
[3] H. Courtecuisse, J. Allard, "Parallel Dense Gauss-Seidel Algorithm on Many-Core Processors", http://codrt.free.fr/allardj/pub/
[4] R.Alqadi, M. Khammash, "An Efficient Paralle Gauss-Seidel Algorithm for the Solution of Load Flow Problems", http://www.ccis2k.org/iait/pdf
[5] Doug Lea, "Java Fork/Join Framework", http://gee.cs.oswego.edu/dl/papers/fj.pdf
[6] C. Evangelinos, C. N. Hill, "Cloud Computing for parallel Scientific HPC Applications", http://www.cca08.org/papers/
[7] C. Bunch, B. Drawert, M. Norman, "MapScale: A cloud environment for scientific computing", http://www.cs.ucsb.edu/
[8] "MATLAB and Simulink for Technical Computing" , http://www.mathworks.com/

Miloš Čubrilo, diplomirao je na fakultetu informacionih tehnologija na Metropolitan univerzitetu u Beogradu. Zaposlen je u Telenoru od 2007 kao Customer Sevice Systems Expert. Pre toga radio je u Mobtelu kao Analyst in customer service. Specijalista je za Java programski jezik, baze podataka, i integraciju računarskih sistema.

Slobodan Jovanović, Prof. Dr. Inž., predaje na Univerzitetu Metropolitan, Fakultetu za informacione tehnologije, od 2008. g. Bavi se razvojem Web aplikacija, i veštačkom inteligencijom, i „pametnim" električnim mrežama („smart" electric grids). Ima veliki boj objavljenih nučnih radova u vodećim internacionalnim časopisima. Predavao je na Strathclyde University, Glasgow, Scotland (UK), u periodu 1993.-2008. U Institutu N.Tesla radio je od 1977. do 1990., a na Queen's University of Belfast od 1990. do 1993. g. Učestvovao je u nizu naučnih i razvojnih, domaćih i inostranih projekata. Na Elektrotehničkom fakultetu u Beogradu (Beogradski univerzitet) diplomirao je 1977. i doktorirao 1984. g.