UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

PARALELNI KOMBINATORNI ALGORITMI SA PRIMJENAMA U STOHASTIČKOJ SIMULACIJI

—MAGISTARSKI RAD—

Komisija:
1. Ž. Mijajlović (mentor)
2. N. Parezanović
3. D. Barjević

TANJA S. PULEVIĆ

Branjeno:
19.05.92

Beograd, 1992. godine

MENTOR RADA

Prof.dr Žarko Mijajlović

Matematički fakultet Beograd

ČLANOVI KOMISIJE

Prof.dr Nedeljko Parezanović

Matematički fakultet Beograd

Prof.dr Dragan Banjević

Matematički fakultet Beograd

Glavni dio ovog rada je urađen zahvaljujući uslovima koji su mi bili obezbijeđeni na Matematičkom institutu i Matematičkom fakultetu u Beogradu. Zahvaljujem se mojim kolegama Zoranu Ognjanoviću, Goranu Gogiću i Zolnai Zsoltu koji su mi za vrijeme boravka u Matematičkom institutu pružili veliku pomoć. Za tehničku obradu rada zahvaljujem se Bebi Pavićević.

Prateći instrukcije profesora Žarka Mijajlovića i zahvaljujući njegovoj stalnoj podršci zainteresovala sam se za probleme paralelnog programiranja i počela da se bavim naučno-istraživačkim radom. Zbog toga i privilegije što sam njegov đak profesoru Žarku Mijajloviću dugujem veliku zahvalnost.

S A D R Ž A J

U V O D

Zadnjih četrdeset godina broj operacija koje kompjuter može da uradi u sekundi duplo se povećava svake dvije godine. Stalno se javljaju aplikacije koje zahtijevaju brže računare. Proizvođači računara su zadovoljavali zahtjeve za brzinom unapređivanjem tehnoloških elemenata. Međutim, u tehnološkom smislu približava se granicama brzine svjetlosti i efektima kvantne fizike, pa je već deset godina opšte prihvaćeno mišljenje da je jedini način, koji može znatno da poveća brzinu obrade, korišćenje paralelnog kompjutera, kompjutera sa više procesorskih jedinica. Ovaj pristup kompjuterskoj obradi zahtijeva posebni hardver, operativne sisteme, paralelne algoritme, jezike i kompajlere, pa je sve to postalo predmet naučnih istraživanja.

Po svemu sudeći, ulazimo u eru paralelizma pa je ovaj rad nastao inspirisan željom da se sa tog popularnog aspekta posmatraju algoritmi koji su sastavni dio mnogih složenih aplikacija. Paralelni algoritam je metod rješavanja problema na paralelnom računaru i dizajn ovakvog algoritma mora biti prilagođen arhitekturi računara na kojem će biti implementiran. U prvom dijelu ovog rada su opisane neke od paralelnih arhitektura.

Generisanje osnovnih kombinatornih objekata je zbog svoje primjenljivosti oduvijek bio važan zadatak. Od 1983. godine dizajnirano je nekoliko paralelnih algoritama koji na različitim arhitekturama rješavaju ovaj zadatak. Mor, Frankel i Zaks su razmatrali pitanje generisanja permutacija. Gupta i Bhattacharjee su dizajnirali algoritme za generisanje varijacija, kombinacija i permutacija [10]. Chan i Chern su razmatrali paralelno generisanje

varijacije klase najviše k od n elemenata [17]. Semba je paralelno generisao sve k podskupove skupa od n elemenata [6]. U drugom dijelu ovog rada su izloženi paralelni algoritmi za generisanje osnovnih kombinatornih objekata, zasnovani na njihovim numeracijama koje je predložio S.G.Akl u radu [3], a takođe i algoritam za generisanje svih $k!$ permutacija za svako k, $1 \leq k \leq n$ koji je predložen u [7]. Izabrali smo ove algoritme jer je njihova implementacija, uz male modifikacije, jednostavna na opremi koja nam je na raspoloaganju. Uz to, procedure koje koriste ovi algoritmi koriste se za dizajn nekih algoritama u četvrtom dijelu ovog rada.

Programi su napisani na jeziku 3L Parallel Fortran za transpjutere i realizovani na transpjuterskoj ploči sa procesorima T800. Za neke algoritme iskorišćena je mogućnost simulacije dijeljenja memorije na transpjuterima. Treći dio opisuje potprograme Parallel Fortrana, koji omogućavaju ovu simulaciju, i potprograme za komunikaciju među procesorima preko kanala.

U četvrtom dijelu su dizajnirani paralelni algoritmi za generisanje slučajnih brojeva sa raznim distribucijama, slučajnih nizova bitova i slučajnih osnovnih kombinatornih objekata. Brzo generisanje slučajnih objekata je osnovna potreba prilikom simuliranja fizičkih sistema koji u sebi imaju nešto probabilističko ili statističko pa su dati i neki primjeri primjene u stohastičkoj simulaciji.

U prilogu su data objašnjenja o meta jeziku koji se koristi za opis algoritama u radu, koraci koji su potrebni da se aplikacija pripremi za izvršavanje i osnovni elementi 3L konfiguracionog jezika potrebnog za pisanje konfiguracionih datoteka.

1. ARHITEKTURE PARALELNIH RAČUNARA

Postoji više načina za klasifikaciju arhitektura paralelnih računara. Detaljni prikaz raznih arhitektura i načina klasifikacije može se naći u radu [8].

Ovdje ćemo izložiti podjelu zasnovanu na konceptima toka instrukcija (instruction stream) i toka podataka (data stream). Tok instrukcija je niz instrukcija koje računar treba da izvrši a tok podataka je niz podataka nad kojim ove instrukcije operišu.

Zavisno od toga da li postoji jedan ili više tokova instrukcija ili podataka podjela je izvršena na SISD, MISD, SIMD i MIMD računare.

SISD (Single Instruction stream, Single Data stream) računari imaju jedan procesor koji izvršava jedan tok instrukcija koje obrađuju jedan tok podataka. Većina serijskih računara je iz ove grupe.

MISD (Multiple Instruction stream, Single Data stream) računari imaju n procesora, $n>1$. Svaki procesor ima svoju kontrolnu jedinicu. Podaci su smješteni u zajedničkoj memoriji. Na svakom koraku procesor dobije podatak iz memorije i obrađuje ga instrukcijom koju je poslala njegova kontrolna jedinica. Dakle, nad jednim tokom podataka simultano se izvršava više tokova instrukcija.

SIMD (Single Instruction stream, Multiple Data stream) računari imaju n, $n>1$ procesora. Svaki procesor ima svoju memoriju. U tim lokalnim memorijama mogu biti smješteni programi i podaci. Rad procesora se odvija prema jednom toku instrukcija koje

odašilje jedna kontrolna jedinica. U jednom koraku svi procesori dobijaju istu instrukciju nad nekim podatkom iz svoje memorije. Ne moraju svi procesori da izvršavaju sve instrukcije koje dobiju. Ako neki procesor treba da preskoči neku instrukciju (to obavještenje nosi sama instrukcija) on čeka ostale da je završe. Čekanje postoji i kada neki od procesora završi neku instrukciju prije nego ostali. Ovako se postiže sinhronizovan rad procesora.

MIMD (Multiple Instruction stream, Multiple Data stream) računari imaju n, $n>1$ procesora od kojih svaki izvršava svoj tok instrukcija nad svojim tokom podataka. Dakle, rad ovih procesora nije sinhronizovan.

U većini slučajeva prilikom izvršavanja nekog algoritma na SIMD ili MIMD mašini javlja se potreba da procesori međusobno komuniciraju tj. da šalju jedni drugima podatke.

Načini komuniciranja među procesorima SIMD (MIMD) mašine daju podjelu na SM (Shared Memory) SIMD (MIMD) i mrežne modele (Interconnection-Network) SIMD (MIMD) računare.

Kod SM SIMD (MIMD) računara procesori komuniciraju preko djeljive memorije. Ako su memorijske lokacije kojima procesori prilaze, zbog upisivanja ili čitanja podataka, različite onda se dozvoljava simultani pristup. Zavisno od toga kako procesori mogu prići istoj memorijskoj lokaciji SM SIMD (MIMD) mašine se dijele na:

EREW (Exclusive-Read, Exclusive-Write) SM SIMD (MIMD) računari. Nema simultanog pristupa od različitih procesora istoj memorijskoj lokaciji ni zbog čitanja ni zbog upisivanja.

ERCW (Exclusive-Read, Concurrent-Write) SM SIMD (MIMD) raču-

nari. Može se simultano upisivati ali ne i čitati.

CREW (Concurrent-Read, Exclusive-Write) SM SIMD (MIMD) računari. Može se simultano čitati ali ne i upisivati.

CRCW (Concurrent-Read, Concurrent-Write) SM SIMD (MIMD) računari. Dozvoljeno je simultano upisivanje i čitanje.

SM MIMD modeli su poznati pod imenom multiprocesori (tightly coupled machines).

Mrežni SIMD (MIMD) model omogućava razmjenu podataka među procesorima preko komunikacionih linija kojima su procesori povezani.

Kod potpune mreže svaki procesor ima vezu sa svim ostalim. Obično, ovolika povezanost nije potrebna pa su popularniji slabije povezani modeli: linearni niz procesora, dvodimenzionalni niz procesora, drvo, kub itd.

Kod linearnog procesorskog niza procesori $P_1,\ldots,P_n$ su povezani dvosmjernim komunikacionim linijama tako da procesor P_1 ima vezu sa P_2, procesor P_n vezu sa P_{n-1} i procesor P_k, $1<k<n$, vezu sa P_{k-1} i P_{k+1}.

Dvodimenzionalni procesorski niz (mesh) se sastoji od n procesora razmještenih u obliku kxk matrice gdje je $k=\sqrt{n}$. Procesor $P_{i,j}$ se nalazi u i-toj vrsti i j-toj koloni matrice. Unutrašnji procesori imaju veze sa susjedima $P_{i+1,j}$, $P_{i-1,j}$, $P_{i,j+1}$ i $P_{i,j-1}$. Procesori na granici imaju manje veza, ugaoni po dvije a ostali po tri.

S - dimenzionalni kub, $s\geq 1$, je mreža od 2^s procesora u kojoj je svaki procesor povezan sa s susjeda.

Mrežni MIMD računari su poznati pod imenom multikompjuteri

(loosely coupled machines).

Transpjuterska ploča sa procesorima T800 funkcioniše kao MIMD mrežni model. Pošto je to jači model od SIMD mrežnog modela onda svaki algoritam dizajniran za MIMD ili SIMD mrežni model može se implementirati na ovoj ploči.

Osim toga na jednom transpjuteru može da se simulira rad MIMD (SIMD) modela sa djeljivom memorijom.

2. MODEL ZA IMPLEMENTACIJU ALGORITAMA

Opišimo elementarne stvari koje se koriste za implementaciju nekih od algoritama iz ovog rada na transpjuterskoj ploči sa procesorima T800 korišćenjem programskog jezika 3L Parallel Fortran.

Transpjuterski sistem podržava paralelnu obradu kao kolekciju sekvencijalnih procesa koji se izvršavaju simultano na različitim procesorima i komuniciraju samo preko kanala.

Svaki transpjuter (procesor) ima 4 veze za spajanje sa ostalim transpjuterima. Svaka veza ima dva kanala, po jedan za oba smjera.

Transpjuter se može posmatrati kao jednoprocesorski sistem ili kao jedan procesor u mreži povezanih transpjutera.

Paralelni Fortran za transpjuterski sistem prati njegov hardverski model. On posmatra jednu aplikaciju kao skup od jednog ili više zadataka koji se simultano izvršavaju. Zadaci su međusobno povezani preko portova. Portovi su pridruženi kanalima. Svaki zadatak ima svoj dio memorije za programski kod, podatke, vektor ulaznih i vektor izlaznih portova. Konfiguraciona datoteka vezuje sve zadatke opisujući koji su procesori u mreži, fizičke veze među procesorima, koji zadatak se nalazi na kojem procesoru i veze među ulaznim i izlaznim portovima različitih zadataka.

Za komunikaciju među zadacima koriste se potprogrami i funkcije iz paketa CHAN. Paket se uvodi rečenicom:

```
INCLUDE 'CHAN.INC'
```

Za nalaženje adresa kanala kojima su pridruženi ulazni i iz-

lazni portovi koriste se funkcije F77_CHAN_IN_PORT i F77_CHAN_OUT_PORT.

Na primjer, rečenica

ULAZNI=F77_CHAN_IN_PORT(*i*)

obezbjeđuje da se vrijednost promjenljive ULAZNI(tipa INTEGER) koristi kao adresa za primanje podataka preko ulaznog porta *i*. Analogno, rečenica

IZLAZNI=F77_CHAN_OUT_PORT(*i*)

obezbjeđuje da se vrijednost promjenljive IZLAZNI(tipa INTEGER) koristi kao adresa za slanje podataka preko izlaznog porta *i*.

Pozivima

CALL F77_CHAN_IN_BYTE(IME,ULAZNI),

CALL F77_CHAN_IN_WORD(IME,ULAZNI),

CALL F77_CHAN_IN_MESSAGE(DUŽINA,IME,ULAZNI)

obezbjeđuje se:

primanje bajta u IME preko ulazne adrese ULAZNI,

primanje riječi u IME preko ulazne adrese ULAZNI,

primanje poruke IME dužine DUŽINA (u bajtima) preko ulazne adrese ULAZNI.

Slanje podataka određene dužine, analogno, obezbjeđuje se potprogramima F77_CHAN_OUT_BYTE, F77_CHAN_OUT_WORD i F77_CHAN_OUT_MESSAGE.

Simulacija dijeljenja memorijskog prostora na transpjuteru

Transpjuterska ploča koja nam je na raspolaganju nema realnu mogućnost dijeljenja memorijskog prostora među procesorima.

Paralelni Fortaran koristi osobine arhitekture transpjutera i omogućava simulaciju paralelne mašine sa djeljivom memorijom na jednom transpjuteru. To se postiže kreiranjem podzadataka koji se konkurentno izvršavaju u sklopu jednog zadatka. Ti podzadaci se nazivaju niti. Niti komuniciraju međusobno preko zajedničke memorije i preko kanala.

Paket potprograma THREAD omogućava rad sa nitima. Datoteka paketa se uvodi rečenicom:

INCLUDE 'THREAD.INC' .

Nit se kreira pomoću opšteg potprograma F77_THREAD_START ili pomoću logičke funkcije F77_THREAD_CREATE.

Poziv

CALL F77_THREAD_START(PP, POČ_NIZ, VELIČINA, ATRIBUTI, BR_ARG, ARG1, ... ARGN)

ostvaruje kreiranje i početak izvršavanja niti.

PP je ime opšteg potprograma na kome je nit bazirana.

POČ_NIZ je tipa INTEGER i predstavlja niz kojim počinje radni prostor niti.

VELIČINA je tipa INTEGER i opisuje veličinu radnog prostora niti.

ATRIBUT je argument koji opisuje atribute niti. Osnovni atribut je prioritet niti. Prioritet može biti urgentan i nije urgentan. Konstante koje opisuje prioritet date su u datoteci THREAD.INC. To su F77_THREAD_URGENT i F77_THREAD_NOTURG. Ako nova nit treba da ima isti prioritet kao tekuća (ona u čijem sklopu se kreira ova nit) za argument ATRIBUT se uzima vrijednost F77_THREAD_PRIORITY( ) koja je tipa INTEGER.

BR_ARG opisuje broj argumenata potprograma PP.

ARG1,...,ARGN su argumenti potprograma PP. Argumenti mogu biti bilo kojeg tipa samo treba obratiti pažnju da se promjenljive tipa CHARACTER prenose preko dva argumenta i o tome treba voditi računa kada se određuje BR_ARG.

Poziv

I=F77_THREAD_CREATE(PP,VELIČINA,BR_ARG,ARG1,...,ARGN) .

kreira i startuje nit baziranu na opštem potprogramu PP. Ova nit se izvršava sa istim prioritetom kao nit koja je kreira. Argumenti imaju isto značenje kao kod pozivanja F77_THREAD_START. Ako je nit uspješno kreirana funkcija vraća vrijednost .TRUE., inače vraća vrijednost .FALSE..

Nit završava svoj rad kada se potprogram PP završi ili pomoću poziva CALL F77_THREAD_STOP.

Ako hoće da koristi Run Time Library nit je mora rezervisati pozivom CALL F77_THREAD_USE_RTL. Pozivom CALL F77_THREAD_FREE nit oslobađa RTL.

Ovo su bili osnovni pozivi omogućeni paketom THREAD.

3. PARALELNO GENERISANJE OSNOVNIH KOMBINATORNIH OBJEKATA

Opštost izlaganja se neće narušiti ako se objekti generišu nad skupom $\{1,2,\ldots,n\}$.

3.1 Paralelno generisanje varijacija

Elementi potrebni za algoritam

Algoritam koji opisujemo zasnovan je na numeraciji varijacija. Bijektivno preslikavanje, nazovimo ga *var*, obezbjeđuje ovu numeraciju:

var: $V_k^n \to \{1,2,\ldots,V_k^n\}$, V_k^n je skup svih varijacija klase k od n elemenata a $V_k^n=n!/(n-k)!$ njihov broj.

Za varijaciju $v_1v_2\ldots v_k$ pravi se niz $v_1',v_2',\ldots,v_k'$, gdje je

$$v_i'=v_i-i+\sum_{j=1}^{i-1}\delta(v_i<v_j),\quad i=\overline{1,k},\quad \delta(v_i<v_j)=\begin{cases}1, & \text{ako je } v_i<v_j\\ 0, & \text{inače}\end{cases}.$$

Primijetimo da je $0\le v_i'\le n-1$ za $i=\overline{1,k}$ pa se $v_1'v_2'\ldots v_k'$ može posmatrati kao prirodan broj u mješovitoj osnovi i njegova decimalna vrijednost predstavlja vrijednost funkcije *var* na varijaciji $v_1v_2\ldots v_k$:

$$var(v_1v_2\ldots v_k)=1+\sum_{i=1}^{k} v_i'\prod_{j=0}^{k-i-1}(n-i-1)$$

Preslikavanje *var* čuva leksikografski poredak varijacija.

Opišimo var^{-1}.

Neka je dat broj 1, element skupa $\{1,2,\ldots,V_k^n\}$. Varijacija $v_1v_2\ldots v_k=var^{-1}(1)$ se dobija kroz sljedeće korake:

Najprije se generiše niz $v_1', v_2', \ldots, v_k'$,

$$v_i' = \left\lfloor (l-1-\sum_{j=1}^{i-1} v_j \prod_{h=0}^{k-j-1} (n-j-h)) \Big/ \prod_{h=0}^{k-i-1} (n-i-h) \right\rfloor , \quad i=\overline{1,k} .$$

Onda se odrede $f_1, f_2, \ldots, f_k$, gdje je f_i najmanji prirodni broj takav da je:

$$f_i = \sum_{i=1}^{i-1} \delta(v_i'+i-f_i<v_j), \quad i=\overline{1,k} .$$

I najzad elementi tražene varijacije su:

$$v_i = v_i'+i-f_i , \quad i=\overline{1,k} . \quad [4]$$

Paralelni algoritam koristi var^{-1}. Procedura *varinv* za date n, k i l, $l \in \{1,2,\ldots,V_k^n\}$, određuje varijaciju $v_1 \ldots v_k = var^{-1}(l)$.

```
varinv (n,k,l,v1,v2,...,vk)
 1  l=l-1
 2  FOR i=1,n
 3      si=0
 4  NEXT i
 5  t=1
 6  FOR i=k-1,1,-1
 7      t=t(n-k+i)
 8  NEXT i
 9  FOR i=1,k
10      c=l/t
11      l=l-tc
12      IF (n>i) THEN
13         t=t/(n-i)
14      ENDIF
15      m=0
```

```
16      j=0
17      DO WHILE (m<c+1)
18         j=j+1
19         IF s_j=0 THEN
20            m=m+1
21         ENDIF
22      ENDDO
23      v_j=j
24      s_j=1
25  NEXT i
```

Najveći dio posla obavlja se u koracima od linije 9 do linije 25 pa je vrijeme izvršavanja ove procedure jednako $O(n \cdot k)$.

Paralelni algoritam koristi i proceduru, nazovimo je *s_var*, koja za date n, k i varijaciju $v_1 \ldots v_k$ generiše varijaciju koja se u leksikografskom poretku nalazi iza $v_1 \ldots v_k$. Ova procedura predstavlja elementarni kombinatorni zadatak pa je nećemo detaljno opisivati (vidjeti [4]). Napomenimo, zbog analize paralelnog algoritma, da je njeno vrijeme izvršavanja jednako $O(k)$.

Paralelni algoritam

Algoritam je dizajniran za SIMD mašinu.

Neka nam je na raspolaganju *nproc* procesora, $1 \le nproc \le V_k^n$. Svaki procesor generiše svoj podskup od V_k^n. P_i, $i=\overline{1, nproc}$, generiše one varijacije iz V_k^n za koje je vrijednost funkcije *var* u skupu $\{(i-1)\lceil V_k^n/nproc \rceil + 1, (i-1)\lceil V_k^n/nproc \rceil + 2, \ldots, i\lceil V_k^n/nproc \rceil\}$.

Pomoću procedure *varinv* P_i najprije nalazi varijaciju čiji je kod (u smislu preslikavanja *var*) jednak $(i-1)\ \lceil V_k^n/nproc\rceil+1$, a zatim, počevši sa tom varijacijom, upotrebom procedure *s_var* generiše, u leksikografskom poretku, sljedećih $\lceil V_k^n/nproc\rceil-1$ varijacija. Podskup generisan procesorom P_i je u leksikografskom poretku iza podskupa generisanog procesorom P_i-1.

Primijetimo da, zavisno od n i k, neki procesori ne generišu tačno $\lceil V_k^n/nproc\rceil$ varijacija.

Sve ovo realizuje se implementacijom algoritma *p_var* na svakom od procesora.

Algoritam *p_var*

(na i-tom procesoru)

```
1  j=(i-1) ⌈V_k^n/nproc⌉+1
2  IF j≤V_k^n THEN
3        varinv(n,k,j,v_1,v_2,...,v_k)
         (* nađena je v_1...v_k=var^{-1}(j) *)
4        WRITE v_1...v_k
5        FOR i=1, ⌈V_k^n/nproc⌉-1
6            s_var(n,k,v_1,v_2,...,v_k)
7        NEXT i
8  ENDIF
```

Ocijenimo vrijeme izvršavanja ovog algoritma.

Za računanje broja V_k^n potrebno je $O(k)$ operacija. Vrijeme izvršavanja za *varinv* je $O(n\cdot k)$, za *s_var* je $O(k)$ i *s_var* se izvršava $\lceil V_k^n/nproc\rceil-1$ puta. Za štampanje u liniji 4 potrebno je vrijeme $O(k)$. Dakle, dominantan član u izrazu za vrijeme izvršavanja

algoritma je $max\{O(n\cdot k),\ O(\lceil V_k^n/nproc\rceil\cdot k)\}$.

Ako je $n\leq\lceil V_k^n/nproc\rceil$ tj. $1<nproc\leq V_k^n/n$, onda je vrijeme izvršavanja algoritma $O(\lceil V_k^n/nproc\rceil\cdot k)$.

Primijetimo da procesori treba da komuniciraju samo dok svi dobiju vrijednosti n i k. Ta komunikacija može se obaviti preko djeljive memorije ili preko kanala.

3.2 Paralelno generisanje permutacija

Neka je $\mathcal{P}_n$ skup svih permutacija skupa $\{1,2,\ldots,n\}$ a $P_n=n!$ njihov broj.

Paralelni algoritam 1

U 2.1. nigdje nema pretpostavke da je $k\neq n$. Stavljajući svugdje $k=n$ dobija se algoritam za paralelno generisanje permutacija.

Paralelni algoritam 2

Algoritam koji opisujemo generiše svih $m!$ permutacija iz $\mathcal{P}_m$ za svako m, $1\leq m\leq n$.

Dizajniran je za arhitekturu procesorski niz sa n procesora, operacije koje se izvršavaju su elementarne i zahtjevi za memorijom su veoma mali.

Ideja je zasnovana na činjenici da se od permutacije $p=p_1p_2\ldots p_{m-1}$ iz $\mathcal{P}_{m-1}$ može dobiti m permutacija iz $\mathcal{P}_m$ stavljajući m na m mogućih mjesta u nizu p. Na ovaj način se od svih permutacija iz $\mathcal{P}_{m-1}$ dobijaju sve permutacije iz $\mathcal{P}_m$.

Procesor P_1 počinje sa $\mathcal{P}_1=\{1\}$.

Procesor P_m, $1<m<n$, dobija (čita) permutaciju $p=p_1\ldots p_{m-1}$ od procesora P_{m-1} i postavlja m na sve moguće pozicije u nizu p. Kada

jednom postavi m, on tu, gotovu, permutaciju štampa i šalje procesoru P_{m+1} na obradu. To se radi za svaku od $(m-1)!$ dobijenih permutacija od P_{m-1}. Na ovaj način P_m generiše cio skup $\mathcal{P}_m$. Zadnji procesor P_n završava posao tj. generiše $\mathcal{P}_n$.

Sinhronizacija rada procesora se uspostavlja preko signala za kraj rada jednog procesora. Kada jedan procesor završi rad on daje to na znanje desnom susjedu stavljajući $p_1=0$. Tako, prvi procesor šalje $P_1=\{1\}$ drugom procesoru i postavlja $p_1=0$. Naravno, zadnji procesor ovo ne radi.

Sve ovo što smo opisali ostvaruje se kroz sljedeći posao na m-tom procesoru ($m\neq 1$ i n)

```
1  RECEIVE p1...pm FROM Pm-1
   (* Pobija se permutacija od lijevog susjeda. Primijetimo
      da je ona dužine m jer sadrži dodatne informacije *)
2  DO WHILE p1≠0
3        pm+1=m
4        FOR i=o,m
5            pm-i+1 ↔pm-i
6        NEXT i
7        SEND p1...pm+1 TO Pm+1
8        WRITE p1...pm      (* generisana permutacija *)
9  ENDDO
10 SEND p1...pm+1 TO Pm+1
   (* p1=0, ovaj procesor je završio rad, generisao je i
    odštampao elemente skupa Pm *)
```

Onog momenta kada prva permutacija stigne do n-tog procesora

neki procesori su već završili svoj posao a neki rade paralelno sa P_n. Tako, vrijeme izvršavanja algoritma je vrijeme koje utroši n-ti procesor i vrijeme za koje je prva permutacija pređe put od prvog do n-tog procesora. Ako se za jedinicu vremena uzme vrijeme za jednu ulazno/izlaznu operaciju i jednu izmjenu mjesta elemenata jedne permutacije onda se vrijeme izvršavanja algoritma može izraziti kao $O(t_1+n!)$ gdje je t_1 vrijeme "putovanja" prve permutacije od P_1 do P_n.

3.3 Paralelno generisanje kombinacija

Elementi potrebni za algoritam

Algoritam koji opisujemo zasnovan je na numeraciji kombinacija. Bijekcija, nazovimo je *komb*, obezbjeđuje ovu numeraciju:

komb: $C_k^n \to \{1,2,\ldots,C_k^n\}$, C_k^n je skup svih kombinacija klase k od n elemenata a $C_k^n=n!/(n-k)!k!$ njihov broj. U C_k^n kombinacije su izlistane u leksikografskom poretku. Preslikavanje *komb* čuva taj poredak.

Neka je $c=c_1\ldots c_k$ jedna kombinacija iz C_k^n $(c_1<c_2<\ldots<c_k)$

$komb(n,c_1\ldots c_k)=C_k^n-f(h(n,c_1\ldots c_k))$. [4]

Preslikavanje h daje komplement kombinacije c u odnosu na skup $\{1,2,\ldots,n\}$ tj. $h(r,c_1\ldots c_k)=c_1'\ldots c_k'$ gdje je

$c_i'=(n+1)-c_{k-i+1}$

Preslikavanje $f:C_k^n \to \{0,1,\ldots,C_k^n-1\}$, $f(c_1\ldots c_k)=\sum_{i=1}^{k} C_i^{c_i-1}$

numeriše kombinacije u antileksikografskom poretku.

Paralelni algoritam koristi $komb^{-1}$. Neka $m \in \{1,2,\ldots,C_k^n\}$

$komb^{-1}(n,k,m)=h(n,f^{-1}(n,k,C_k^n-m))$.

Ako je $f(c_1 \ldots c_k)=t$ onda se elementi kombinacije $c_1 \ldots c_k=f^{-1}(t,n,k)$ dobijaju na sljedeći način: c_i je najveće j koje zadovoljava:

$$i \le j \le n \text{ i } t-\sum_{s=i}^{k} C_s^{s-1} \ge C_i^{j-1} .$$

Procedura *kombinv* koju će pozivati paralelni algoritmi i koja računa $komb^{-1}(n,k,m)$ izgleda:

```
kombinv (n,k,m,c1,...,ck)
(* računa se f^-1(n,k,Ck^n-m)=r1...rk *)
 1  t1=Ck^n-m
 2  FOR i=k,1,-1
 3      ri=0
 4      j=n
 5      s=Ci^n-1
 6      DO WHILE ri=0
 7          IF (t1≥s) THEN
 8              ri=j
 9          ENDIF
10          s=s(j-i)/j
11          j=j-1
12      ENDDO
13      t1=t1-Ci^ci-1
14  NEXT i
    (* računa se h(n,r1,...,rk)=c1...ck *)
```

15 FOR $i=1,k$

16 $c_i=(n+1)-c_{k-i+1}$

17 NEXT i

Vrijeme izvršavanja ove procedure je $O(n\cdot k)$.

Algoritam koristi i proceduru, nazovimo je *s_komb*, koja za date n,k i kombinaciju $c_1\ldots c_k$ generiše kombinaciju koja se u leksikografskom poretku nalazi iza $c_1\ldots c_k$ (vidjeti [4]). Složenost ove procedure je $O(k)$.

Paralelni algoritam

Algoritam je dizajniran za SIMD mašinu.

Procesori komuniciraju međusobno samo dok svi dobiju vrijednosti za n i k.

Neka nam je na raspolaganju *nproc* procesora, $1\le nproc\le C_k^n$. Svaki procesor generiše svoj poskup od C_k^n. P_i, $i=\overline{1,nproc}$, generiše one kombinacije iz C_k^n za koje je vrijednost funkcije *komb* u skupu $\{(i-1)\lceil C_k^n/nproc\rceil+1,\ (i-1)\lceil C_k^n/nproc\rceil+2,\ldots,i\lceil C_k^n/nproc\rceil\}$.

Pomoću procedure *kombinv* P_i najprije nalazi kombinaciju za koju je vrijednost funkcije *komb* jednaka $(i-1)\lceil C_k^n/nproc\rceil+1$, a zatim, počevši sa tom kombinacijom, upotrebom procedure *s_komb* generiše, u leksikografskom poretku, sljedećih $\lceil C_k^n/nproc\rceil-1$ kombinacija. Podskup generisan procesorom P_i je u leksikografskom poretku iza podskupa generisanog procesorom P_{i-1}.

U zavisnosti od n i k, neki procesori ne generišu tačno $\lceil C_k^n/nproc\rceil$ kombinacija.

Sve ovo realizuje se implementacijom algoritma *p_komb* na

svakom od procesora.

Algoritam *p_komb*

(* na *i*-tom procesoru *)

1 $j=(i-1)\lceil C_k^n/nproc\rceil+1$

2 IF ($j\leq C_k^n$) THEN

3 $kombinv(n,k,j,c_1\ldots c_k)$

4 (* nađena je kombinacija $c=komb^{-1}(n,k,j)$)

5 WRITE $c_1,\ldots,c_k$

6 FOR $i=1$, $\lceil C_k^n/nproc\rceil-1$

7 $s\_komb(n,k,c_1,\ldots,c_k)$

8 NEXT *i*

9 ENDIF

Ocijenimo vrijeme izvršavanja ovog algoritma.

Za računanje broja C_k^n potrebno je $O(k)$ operacija. Za izvršavanje procedure *kombinv* potrebno je vrijeme $O(n\cdot k)$. Vrijeme izvršavanja za *s_komb* je $O(k)$ i *s_komb* se izvršava $\lceil C_k^n/nproc\rceil-1$ puta.

Za štampanje u liniji 4 potrebno je vrijeme $O(k)$.

Dakle, dominantan član u izrazu za vrijeme izvršavanja algoritma je $max\ \{O(n\cdot k),\ O(\lceil C_k^n/nproc\rceil k)\}$.

Za $n\leq\lceil C_k^n/nproc\rceil$, tj. $1<nproc\leq C_k^n/n$ vrijeme izvršavanja algoritma je $O(\lceil C_k^n/nproc\rceil k)$.

4. PARALELNO GENERISANJE PSEUDO-SLUČAJNIH OBJEKATA

Prva stvar potrebna stohastičkoj simulaciji je izvor slučajnosti.

Razmatraćemo neke izvore slučajnosti, paralelizovati njihov rad i kroz primjere vidjeti njihovu primjenu u stohastičkoj simulaciji.

Dakle, posmatramo generatore pseudo-slučajnih objekata. Kažemo pseudo-slučajnih jer svi ovi, algoritamski dizajnirani, generatori rade po determinističkom principu i pri istim početnim uslovima svako novo aktiviranje generatora daje iste nizove objekata.

Kod svih algoritama koji imaju svoje generatore slučajnih objekata i koji treba da se adaptiraju za paralelni kompjuter odmah se javljaju pitanja vezana za korelacije između nizova slučajnih objekata dobijenih na različitim čvorovima kompjutera. Ako postoje korelacije, onda može doći do nagomilavanja istih informacija sa različitih čvorova, što ne poboljšava tačnost tražene informacije. Može da se desi još gora stvar, da korelacije dovedu do pogrešnih rezultata.

Kod sekvencijalnog slučaja, pronađeni su načini generisanja slučajnih objekata koji su oslobođeni korelacije. Dakle, zadovoljavajuće rješenje bi bio paralelni metod koji bi imitirao sekvencijalni, tj. metod koji daje iste nizove slučajnih objekata kao i sekvencijalnih.

Za najpopularniji generator pseudo-slučajnih brojeva, linearni kongruentni generator, pronađen je paralelni algoritam

koji ga imitira. Taj algoritam je izložen u [9].

Opisaćemo ga u 4.1. i primijeniti na paralelno generisanje slučajnih brojeva sa $\mathcal{U}(0,1)$ raspodjelom. Brzo generisanje brojeva sa $\mathcal{U}(0,1)$ raspodjelom je veoma važno, između ostalog, zbog toga što se mnoge raspodjele interesantne za stohastičku simulaciju modeliraju pomoću nje.

Svi paralelni algoritmi dizajnirani u ovom dijelu imitiraju zadovoljavajuće sekvencijalne verzije.

4.1 Linearni kongruentni generator

Linearni kongruentni generator, za izabrane "magične" brojeve Z_0, a, c, m, $Z_0 \geq 0$, $a \geq 0$, $m \geq Z_0$, $m > a$, $m > c$ generiše niz brojeva po principu

$$Z_{n+1} = (aZ_n + c) \bmod m \qquad (1)$$

Z_0 se naziva sjeme generatora a niz (Z_n) linearni kongruentni niz. Ovaj generator je uveo Lehmer 1948. g.

Biranje "magičnih" brojeva je detaljno opisano u [11]. Nas interesuje paralelizacija rada ovog generatora pri već odabranim Z_0, m, c i a.

Paralelni algoritam

Algoritam je dizajniran za SIMD mašinu. Neka je *nproc* broj procesora. Komunikacija među procesorima nije potrebna.

Ideja paralelnog algoritma leži u vezi između n-tog i $(n+k)$-tog elementa niza (Z_n). Veza se dobija iz (1) i izgleda:

$$Z_{n+k}=(AZ_n+C) \bmod m \qquad (2)$$

gdje je $A=a^k$, $C=1+a+a^2+\ldots+a^{k-1}$.

Svaki procesor radi po principu (2) za $k=nproc$ počevši sa svojom "pomjerenom" vrijednošću za sjeme.

Rastojanje između brojeva generisanih na jednom procesoru je $nproc-1$. Rastojanje određujemo u odnosu na sekvencijalno generisan niz. Ovo znači da ako se u sekvencijalno generisanom nizu uoče dva broja koja se paralelnim algoritmom generišu na jednom procesoru jedan iza drugog onda između njih postoji $nproc-1$ broj.

Ovo se postiže sljedećim radom na i-tom, $i=0,\overline{nproc-1}$, procesoru:

Algoritam *lkg*

```
 1  Postaviti vrijednosti za Z0, a, c i m
 2  A=1, C=0
 3  FOR k=0,nproc-1
 4       A=(aA)mod m
 5       C=(aC+c) THEN
 6       IF (k=i) THEN
 7            Z=(AZ0+c)mod m
 8            (* ovo Z je sjeme na ovom generatoru *)
 9       ENDIF
10  NEXT k
11  Z=(AZ+C)mod m
```

Koraci u linijama od 1 do 9 definišu generator koji će raditi na i-tom procesoru tj. određuju A,C i sjeme. A i C su na svakom procesoru isti a sjeme nije.

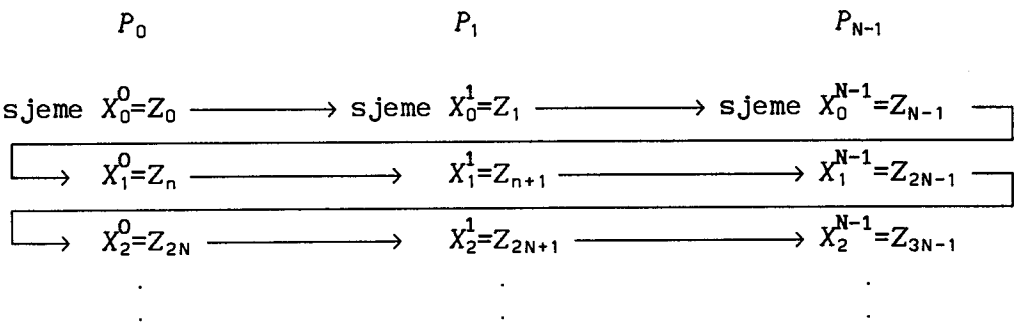
Korak u liniji 10 se ponavlja onoliko puta koliko slučajnih

brojeva treba da se generiše na i-tom procesoru.

Sa Z_i označimo i-ti broj generisan generatorom (1) a sa X_k^l k-ti broj generisan na l-tom procesoru pomoću algoritma koji opisujemo.

Generisanje po procesorima izgleda:

Označimo *nproc*=N



Vidimo, slijedeći strelice tj. uzimajući brojeve sa P_0, P_1, P_{N-1}, P_0, ... procesora da se dobija isti niz brojeva kao u sekvencijalnom slučaju tj. niz Z_0, Z_1, Z_2,...

4.2 Generisanje pseudo-slučajnih brojeva sa $\mathcal{U}(0,1)$ raspodjela

Pomoću niza (Z_n) dobijenog generatorom (1) dobija se niz pseudo-slučajnih brojeva (U_n), $U_n=X_n/m$ uniformno distribuiranih na $(0,1)$.

Sekvencijalni algoritam

1 Postaviti vrijednosti za Z_0, a, c i m

2 $Z=Z_0$

3 $Z=(aZ+c) \bmod m$ (3)

4 $U=Z/m$

Koraci u linijama 3 i 4 se ponavljaju zavisno od toga koliko je potrebno generisati brojeva.

Paralelni algoritam

Algoritam je dizajniran za SIMD računar.

Zbog potpune zavisnosti generatora (3) od linearnog kongruentnog generatora, ideja je ista kao kod algoritma *lkg* jedino treba dodati korak koji obavlja transformaciju $U=Z/m$.

Dakle, na i-tom, $i=\overline{0,nproc-1}$, procesoru obavlja se posao:

Algoritam *unif*

```
 1  postaviti vrijednosti za Z0, a, c i m
 2  A=1, C=0
 3  FOR k=0, nproc-1
 4      A=(aA)mod m
 5      C=(aC+c) THEN
 6      IF (k=i) THEN
 7          Z=(AZ0+c)mod m
            (* ovo Z je sjeme na ovom procesoru *)
 8      ENDIF
 9  NEXT k
10  U=Z/m
11  Z=(AZ+C)mod m
```

Koraci u linijama 10 i 11 se ponavljaju.

Primijetimo da je redosljed ovih koraka obrnut u odnosu na sekvencijalni slučaj zbog potpune imitacije.

Primjer

Posmatrajmo paralelni algoritam za računanje $I=\int_0^1 f(x)dx$ Monte Carlo metodom. Ako su r_i slučajni brojevi sa $\mathcal{U}(0,1)$ raspodjelom

onda je

$$I=\lim_{k\to\infty}\frac{1}{k}\sum_{1}^{k}f(r_i).$$

Paralelni algoritam se svodi na prethodno opisani *unif*. Svaki procesor pravi svoju sumu vrijednosti funkcije *f* na slučajno odabranim brojevima na tom procesoru i na kraju, poslije željenog broja biranja brojeva, suma se objedinjuje u jednu i ta jedna se dijeli ukupnim brojem biranja na svim procesorima.

Ovaj algoritam je implementiran na transpjuterskoj ploči pomoću fortranskih datoteka *int1*, *int2*, *int3*, *int4* i konfiguracionom *int.cfg*.

Monte Carlo metode za računanje integrala dolaze do izražaja u statističkoj mehanici i teoriji kvantnih polja gdje se računaju integrali sa hiljadama promjenljivih. [9]

4.3 Generisanje pseudo-slučajnih brojeva sa *N*(0,1) raspodjelom

Najpoznatiji "tačni metod" za generisanje brojeva sa normalnom raspodjelom čiji su parametri 0 i 1 je metod Boxa i Mullera uveden 1958.g. Često se upotrebljava, prilično je spor pa je paralelizacija njegovog rada opravdana.

Kod ovog metoda par (X,Y) brojeva sa $N(0,1)$ raspodjelom generiše se pomoću para (U_1,U_2) brojeva sa $\mathcal{U}(0,1)$ raspodjelom na sljedeći način:

$$X=\sqrt{-2\ln U_2}\cos 2\pi U_1$$
$$Y=\sqrt{-2\ln U_2}\sin 2\pi U_1 \qquad (4)$$

Paralelni algoritam

Algoritam se dizajnira za SIMD mašinu.

Na svakom od *nproc* procesora generišu se parovi slučajnih brojeva sa $N(0,1)$ raspodjelom. U paru su dva broja dobijena vezama (4) iz para "uzastopnih" U_1 i U_2.

Rastojanje između parova generisanih na jednom procesoru je $2nproc-1$. Ovo znači da ako u sekvencijalno generisanom nizu uočimo dva para koji su generisani na istom procesoru onda između prvog elementa prvog para i prvog elementa drugog para postoji $2nproc-1$ brojeva.

Iz (4) se vidi da se problem svodi na generisanje parova iz $U(0,1)$. Ideja algoritma je da procesor napravi 2 broja iz $U(0,1)$, pomoću njih napravi 2 broja iz $N(0,1)$ i onda napravi pomjeraj dužine $2nproc-1$.

Generisanje para uzastopnih U_1 i U_2 obezbjeđuje se generatorom (1) a pomjeraj se obezbjeđuje generatorom (2) za $k=2nproc$.

Dakle, na svakom procesoru moraju raditi generatori oba tipa tj. na i-tom, $i=\overline{0,nproc-1}$, procesoru se obavlja posao:

Algoritam *norm*

1 $A=1,\ C=0$

2 postaviti vrijednosti za a,c,m i z_0

3 FOR $l=0,2nproc-1$

4 $A=(aA) mod\ m$

5 $C=(aC+c) mod\ m$

6 IF $(l=2i)$ THEN

7 $Z=(AZ_0+c) mod\ m$

8 ENDIF

9 NEXT k

(* sada je određen generator (2) za $k=2nproc$ *)

10 $U_1=Z/m$ (* prvi u paru *)

11 $Z=(aZ+c) mod\ m$ (* generator (1) za sljedeći broj u paru *)

12 $U_1=Z/m$ (* drugi u paru *)

13 $X=\sqrt{-2lnU_2}\ cos2\pi U_1$

14 $Y=\sqrt{-2lnU_1}\ sin2\pi U_1$

(* par normalnih je generisan *)

15 $Z=(AZ+C)\ mod\ m$ (* generator (2), pravi pomjeraj $2nproc-1$ *)

Koraci u linijama od 10. do 15. se ponavljaju.

Sa (X_i, Y_i) označimo i-ti par generisan sekvencijalno tj. generatorom (4) a sa (X_k^l, Y_k^l) k-ti par generisan l-tim procesorim.

Generisanje po procesorima izgleda:

P_0 P_1 $P_{nproc-1}$

$(X_1^0, Y_1^0)=(X_1, Y_1)\rightarrow(X_1^1, Y_1^1)=(X_2, Y_2)\rightarrow(X_1^{nproc-1}, Y_1^{nproc-1})=(X_{nproc-1}, Y_{nproc-1})$

$\downarrow$

$(X_2^0, Y_2^0)=(X_{nproc}, Y_{nproc})\longrightarrow$

Slijedeći strelice vidimo da paralelni algoritam imitira sekvencijalni.

Razmotrimo još jedan popularan ali spor metod biranja brojeva sa $N(0,1)$ raspodjelom.

Ako su $X_1, X_2, \ldots$ nezavisne slučajne promjenljive i svaka sa istom raspodjelom, $E(X_k)=a$, $\sigma^2(X_k)=\sigma$ $k=1,2,\ldots$ onda raspodjela (prema centralnoj graničnoj teoremi) za standardizovani oblik njihovog zbira

$S_n=X_1+X_2+\ldots+X_n$ tj.

raspodjela slučajne promjenljive

$$S_n^* = \frac{S_n - E(S_n)}{\sqrt{\sigma^2(S_n)}} = \frac{S_n - an}{\sqrt{n\sigma^2}}$$

teži raspodjeli $N(0,1)$ kad $n \to \infty$.

Tačnost se povećava sa povećanjem n.

Ako se za $X_1, X_2, \ldots$ uzmu $U_1, U_2, \ldots$ slučajne promjenljive sa $U(0,1)$ raspodjelom, $E(U_k) = 1/2$, $\sigma^2(U_k) = 1/12$ onda je

$$S_n^* = \frac{\sum_{i=1}^{n} U_i - \frac{n}{2}}{\sqrt{n \cdot 1/12}} .$$

Za $n=12$ dobija se izraz $\sum_{i=1}^{12} U_i - 6$ koji za izračunavanje izgleda jednostavan a pokazalo se da je ova aproksimacija dovoljno dobra za praktičnu primjenu. Dakle, ako su U_i, $i=\overline{1,12}$, slučajni brojevi sa $U(0,1)$ raspodjelom onda je $X = \sum_{1}^{12} U_i - 6$ slučajan broj sa $N(0,1)$ raspodjelom. Paralelizujemo, za razliku od prethodnog slučaja, generisanje samo jednog ovakvog broja. Sekvencijalnom algoritmu treba 12 taktova da generiše sve U_i, $i=\overline{1,12}$. Posmatrajmo broj procesora koji je pogodan za primjenu ideje sa pomjerenim generatorom. Ako imamo na raspolaganju 12 procesora onda se generisanje svih U_i, $i=\overline{1,12}$ obavlja u jednom taktu tako da na svakom procesoru radi generator tipa (1).

br. procesora	br. takta	rade generatori
6	2	(1) i (2) za $k=2nproc$
4	3	(1) i (2) za $k=3nproc$
3	4	(1) i (2) za $k=4nproc$
2	6	(1) i (2) za $k=6nproc$

Primjer

U radu [12] je izložen probabilistički metod za nalaženje minimuma realne funkcije pomoću niza slučajnih brojeva sa normalnom distribucijom.

Ideja metoda

Neka je $I \subseteq R^k$ jedinični kub i $f: I \to R$. Generišu se dva niza X_n, Y_n slučajnih brojeva gdje je:

1. Y_n niz slučajnih brojeva uniformno distribuiranih
2. $X_{n+1} = \begin{cases} X_n \text{ ako je } f(X_n) \leq f(Y_n) \\ Y_n \text{ ako je } f(X_n) > f(Y_n) \end{cases}$.

Uz neke uslove za f, niz X_n konvergira, u vjerovatnoći, ka $\min_{x \in I} f(x)$, ako minimum postoji. Ovakav proces je spor pa ideja da se ubrza bržom koncentracijom brojeva Y_n oko očekivane vrijednosti. Tako, niz Y_n se generiše sa normalnom distribucijom čija je očekivana vrijednost X_n a standardna devijacija σ_n, gdje je $\lim_n \sigma_n = 0$. Prema tome, uslov 1 se zamjenjuje uslovom $Y_n | X_n : N(X_n, \sigma_n^2)$, $\lim_n \sigma_n = 0$. Dizajniran je sekvencijalni algoritam koji umjesto I posmatra $D = [a_1, b_1] x \ldots x [a_n, b_n]$, za početne vrijednosti se uzimaju $x_i = \frac{a_i + b_i}{2}$, $\sigma_i = \frac{b_i - a_i}{2}$, $i = \overline{1,n}$. Uslov $\lim_n \sigma_n = 0$ se može obezbijediti na više načina, između ostalog sa $\sigma_i = q\sigma_i$, $0 < q < 1$, $i = \overline{1,n}$. Izlazni kriterijum, takođe, može da se definiše na više načina, zadavanjem broja ciklusa izračunavanja, zadavanjem ε_i, $i = \overline{1,n}$ tako da $\sigma_i < \varepsilon_i$, $i = \overline{1,n}$ itd. Pokazalo se, da algoritam, za sve primjere na koje se primjenjivao nalazi rješenje u ne više od $100 \cdot 4^{n-1}$ računskih koraka. Prema tome, sekvencijalno, on je koristan za funkcije sa šest ili manje promjenljivih. Kada se paralelizuje on može biti koristan i za funkcije sa većim brojem promjenljivih.

Opisani metod sadrži dosta elemenata koji se mogu paralelizovati. Odmah se nameću dva načina paralelizacije. Jedan način bi bio zasnovan na paralelnom generisanju slučajnih n-torki iz normalne distribucije a drugi bi vršio paralelizaciju domena.

Opišimo ideje za dva paralelna algoritma.

Prvi algoritam dizajniramo za SIMD mašinu sa djeljivom memorijom gdje je $nproc=n$. Zasnovan je na paralelizaciji generisanja niza Y_n, paralelizaciji izmjene $X_n \leftrightarrow Y_n$ i paralelizaciji računanja σ_i, $i=\overline{1,n}$.

Algoritam izgleda:

1 FOR $i=1,n$ P_i DO IN PARALLEL IN SM

2 READ a_i, b_i, ε_i

(* smještanje početnih podataka u djeljivu memoriju *)

3 NEXT i

4 FOR $i=1,n$ P_i DO IN PARALLEL IN SM

5 $x_i = \frac{a_i+b_i}{2}$

6 $\sigma_i = \frac{b_i-a_i}{2}$

7 ENDFOR

(* početni uslovi za X_n i niz disperzija su smješteni u djeljivoj memoriji *)

8 $w=f(x_1,\ldots,x_n)$

(* zavisno od oblika f i njeno računanje se može paralelizovati *)

9 FOR $i=1,n$ P_i DO IN PARALLEL IN SM

10 generate $y_i | N(x_i, \sigma_i^2)$

11 NEXT i

12 $h=f(y_1,\ldots,y_n)$

```
13  IF (h<w) THEN
14      FOR i=1,n P_i DO IN PARALLEL IN SM
15          x_i=y_i
16      NEXT i
17      w=f(x_1,...,x_n)
18  ENDIF
19  FOR i=1,n P_i DO IN PARALLEL IN SM
20      σ_i=qσ_i
21  NEXT i
21  FOR i=1,n P_i DO IN PARALLEL IN SM
22      IF (σ_i<ε_i) THEN
23          l_i=1
24      ENDIF
25      l_i=0
26  NEXT i
    (* l_i=1 znači da je na i-tom procesoru zadovoljen
    kriterijum izlaska *)
27  u=l_1+...+l_n
    (* i ovo sumiranje se može raditi paralelno *)
28  IF (u=n) THEN
    (* da li je na svim procesorima zadovoljen kriterijum
    izlaska *)
29      WRITE x_1,...,x_n
30      STOP
31  ENDIF
32  GO TO LINE 9
```

Brojevei y_i se mogu određivalti kao $y_i=x_i+\sigma_i y_i'$ gdje je $y_i'=\sum_1^{12} U_j-6$. Da bi se postigla imitacija sekvencijalnog algoritma na svakom procesoru (u svakoj niti) radi pomjereni generator (2) za $k=12\cdot n$.

Paralelizacija algoritma po domenu funkcije f se može ostvariti tako što svaki od *nproc* procesora radi sekvencijalni algoritam na određenom podskupu domena. Ako je $D=[a_1,b_1]x\ldots x[a_n,b_n]$ onda procesor P_j, $j=\overline{1,nproc}$, radi na domenu $D_j=[a_1+(j-1)\frac{b_1-a_1}{nproc}, a_1+j\frac{b_1-a_1}{nproc}]x\ldots x[a_n+(j-1)\frac{b_n-a_n}{nproc}\ a_n+j\frac{b_n-a_n}{nproc}]$.

Primjer

Stohastičke metode u optimizaciji često koriste generisanje slučajnih tačaka na sferi.

Ako su $X_1,\ldots,X_n$ brojevi sa $N(0,1)$ raspodjelom, slučajna tačka na n-dimenzionalnoj sferi je $(X_1/r, X_2/r,\ldots,X_n/r)$, gdje je $r=\sqrt{X_1^2+X_2^2+\ldots+X_n^2}$. Paralelno generisanje ovakvih tačaka može se obezbijediti direktnom primjenom nekih od opisanih algoritama za generisanje brojeva sa $N(0,1)$ raspodjelom.

4.4 Generisanje pseudo-slučajnih nizova bitova

U kriptografiji se, između ostalog, izučavaju pseudo-slučajni nizovi bitova kao i specijalni hardver namijenjen za njihovo generisanje. Generatori pseudo-slučajnih nizova bitova, ‚GPSN, su bazirani na pomjeračkim registrima. Oni proizvode i-ti bit u nizu na osnovu prethodnih n bitova na sljedeći način:

$$b_i=f(b_{i-1},\ldots,b_{i-n}),\quad f:\{0,1\}^n\to\{0,1\}$$

Za funkciju f obično se uzima linearna rekurentna veza

$$b_i=(d_1b_{i-1}+\ldots+d_nb_{i-n})\,mod2 \qquad (5)$$

gdje su $d_1,\ldots,d_n$ binarne konstante.

Sabiranje po modulu 2 i $\oplus$ (ekskluzivno ili) imaju istu istinitosnu tablicu pa (5) možemo da zapišemo kao

$$b_i=b_{i-l_1}\oplus b_{i-l_2}\ldots b_{i-l_k},$$

gdje je $d_{l_1}=\ldots=d_{l_k}=1$ i ostali $d_i=0$.

Maksimalna perioda generatora tipa (5) je 2^{n-1}. Jedino su zanimljivi oni koji imaju maksimalnu periodu.

Računanje periode i razmatranje pitanja dostizanja maksimalne periode obavlja se pomoću metoda faktorizacije polinoma nad konačnim poljima. Rekurentnoj vezi (5) pridružuje se polinom

$$p(x)=x^n+d_1x^{n-1}+\ldots+d_n.$$

Uobičajeno je da se razmatraju trinomi

$1+x^r+x^s$ gdje je $1\le r\le s$.

Ovim trinomom odgovara rekurentna veza

$$b_i=b_{i-s}\oplus b_{i-(s-r)}. \qquad (6)$$

Obrtanjem niza dobijamo trinom

$$1+x^{s-r}+x^s$$

za koji je odgovarajuća veza

$$b_i=b_{i-s}\oplus b_{i-r}. \qquad (7)$$

(6) i (7) imaju istu periodu.

Cilj je, dakle, da se dizajnira paralelni algoritam koji za date s i r i početnih s bitova generiše niz slučajnih bitova koji se dobijaju rekurentnom vezom (7).

Paralelni algoritam

Izložićemo ideju za $s=9$, $r=5$, $nproc=4$.

Uz male transformacije ideja se prilagođava drugim vrijednostima za s, r i $nproc$.

Dakle, generator radi po principu

$b_i=b_{i-9}\oplus b_{i-5}$ uz početne bitove $b_1, b_2, \ldots, b_9$. (8)

Prije početka nastavljanja niza bitovi $b_1, \ldots, b_9$ se smjeste u zajedničku memoriju. Onda, svaki od četiri procesora generiše po jedan bit, tako da se u jednom taktu niz b nastavlja za 4 bita. Za vrijeme izvršavanja algoritma niz b je stalno u zajedničkoj memoriji.

Generisanje po procesorima izgleda:

	P_0	P_1	P_2	P_3
prvi takt	b_{10}	b_{11}	b_{12}	b_{13}
drugi takt	b_{14}	b_{14}	b_{16}	b_{17}
	.			
	.			
	.			
	b_{41-2}	b_{41-1}	b_{41}	b_{41+1}

Na k-tom procesoru se izvršavaju koraci

```
FOR i=10+k, m, 4
    b_i=b_{i-9}⊕b_{i-5}     (* m je dužina niza *)
NEXT i
```

Poslije nekoliko taktova, kada se "istroše" početne vrijednosti, za rad procesora P_0 potrebni su samo bitovi koje generiše procesor P_3. Na analogan način rad procesora P_i zavisi od procesora P_{i-1} $i=1,3$. Pokažimo zavisnost P_1 od P_0. Primijetimo da su indeksi elemenata niza b na procesoru P_0 oblika $4l-2$, na P_1 oblika

$4l-1$, na P_2 oblika $4l$ i na P_3 oblika $4l+1$.

Onda za neko l na procesoru P_1 se generiše

$b_{4l-1}=b_{4l-1-9}\oplus b_{4l-1-5}=b_{4l_1-2}\oplus b_{4l_2-2}$ a b_{4l_1-2} i b_{4l_2-2} su generisani procesorom P_0.

Ostale zavisnosti se pokazuju na analogan način.

Pretpostavlja se idealna situacija, da svaki procesor obavlja svaku operaciju u istom vremenskom intervalu. Ako situacija nije idealna onda se čekanje među procesorima reguliše pomoću neke od raspoloživih tehnika: semaforima, indikatorima itd.

Iz izloženog se vidi da je ocjena efikasnosti algoritma optimalna tj. jednaka $O(nproc)$.

Programska realizacija ovog algoritma data je fortranskom datotekom *shift1* i konfiguracionom *shift1.cfg*. Iskorišćena je mogućnost dijeljenja memorije na transpjuteru.

4.5 Generalizovani generator pseudo-slučajnih brojeva baziran na pomjeračkom registru - GFSR

Konstruktori GFSR-a, Lewis i Payne, ostaju u svijetu bitova i princip rada svoga generatora zasnivaju na pravljenju n-bitnih prirodnih brojeva. Preciznije, iz slučajnog niza bitova izdvaja se n nesusjednih bitova i dobija se n-bitni prirodni broj:

$$X_l=b_l b_{l-j_2}\ldots b_{l-j_n} \qquad (9)$$

Svaki bit od X_l pokorava se zakonu (6) pa se može formirati rekurentna veza $X_l=X_{l-s}\oplus X_{l-(s-r)}$.

Prvih s bitova ne moraju da zadovoljavaju (9). Perioda niza (X_l) zavisi od početnih uslova.

Trivijalan način da se od ovakvog niza dobiju slučajni brojevi sa $\mathcal{U}(0,1)$ raspodjelom je $U_i=2^{-n}X_i$.

Navedimo neke parove r i s za koje se postiže maksimalna perioda 2^s-1

r	s
2	1
3	1, 2
.	
17	3, 5, 6, 11, 12, 14
.	
33	13, 20
.	
36	11, 25

Paralelni algoritam

Algoritam je dizajniran za SIMD mašinu sa djeljivom memorijom.

Dakle, treba paralelizovati generisanje niza n-bitnih prirodnih brojeva po vezi:

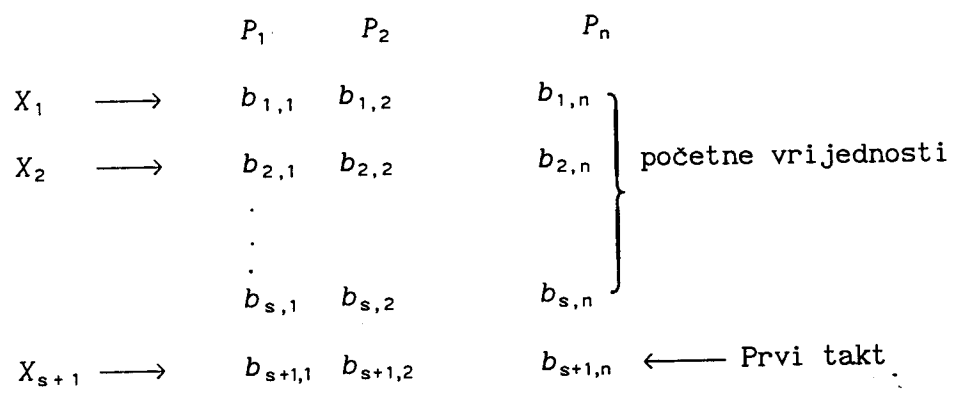
$X_i=X_{i-s}\oplus X_{i-(s-r)}$, gdje su dati je $X_1,\ldots,X_s$.

Pretpostavimo da je $nproc=n$.

Ideja algoritma je da se sabiranja po modulu 2 vrše vektorski. Matrica koja čuva n-bitne brojeve smiješta se u zajedničku memoriju. Elementi te matrice su bitovi, u jednoj vrsti je jedan n-bitni broj, dakle, matrica ima n kolona.

Prije početka generisanjanovih brojeva matrica sadrži s početnih vrijednosti (s vrsta).

Procesor P_i, $i=\overline{1,n}$ generiše i-ti bit, gledano slijeva na desno, novog broja. Tako, u jednom taktu generisan je jedan n-bitni broj.



Znači, procesor P_i stalno radi na i-toj koloni i njegov rad ne zavisi od ostalih procesora niti utiče na njih.

Na procesoru P_i, $i=\overline{1,n}$ se izvršavaju sljedeći koraci:

$b_{1,l}=b_{1-s,l}\oplus b_{1-(s-r),l}$, $l=s+1, s+2, \ldots$

Pošto su u jednom taktu dobijeni bitovi potrebni za jedan n-bitni broj onda svaki procesor može da izvrši transformaciju svoga bita tj. da izvrši operaciju $b_{1,l}\cdot 2^{n-i}$ a sabiranje ovakvih članova može da se reguliše dva po dva paralelno.

Iz izloženog se vidi da je ocjena efikasnosti algoritma optimalna tj. jednaka je $O(nproc)$.

U slučaju da je $nproc<n$ ideja vektorskog sabiranja po modulu 2 ostaje samo što sada procesori rade na blokovima kolona. Procesor P_i obrađuje kolone $(i-1)\lceil n/nproc\rceil+1, \ldots, i\lceil n/nproc\rceil$. Može se desiti, zavisno od n, da neki procesori obrađuje manje blokove.

Prilikom programske realizacije ovog algoritma, kao i kod prethodnog, iskoristila se mogućnost simulacije djeljivosti memorije na transpjuteru.

Programska realizacija ove ideje data je fortranskom datotekoa *shift2* i konfiguracionom *shift2.cfg*. Kao kod prethodnog algoritma i ovdje se prilikom programske realizacije iskoristila mogućnost simulacije dijeljenja memorije na procesoru.

Urađen je testni primjer za generator koji ima maksimalnu periodu: $s=7$, $r=1$ i početna matrica je

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}.$$

Generišu se redom brojevi 0,1,2,3,4,5,6,1,3,1,7,1,7,1,7,2,2,6,... Pošto se generišu 3-bitni prirodni brojevi uzeli smo *nproc=3*. Algoritam ostaje isti ako se s, r i početna matrica zadaju na ulazu.

4.6 Generisanje pseudo-slučajnih osnovnih kombinatornih objekata

4.6.1 Varijacije

Algoritam je dizajniran za SIMD mašinu.

Ideja je zasnovana na numeraciji varijacija. Numeracija je opisana u 3.1.

Koristeći numeraciju biranje slučajnih varijacija svodi se na biranje slučajnog prirodnog broja iz skupa $\{1,\ldots,V_k^n\}$. U ovom algoritmu, biranje slučajnog prirodnog broja zasnovano je na sljedećem: ako je U sa $\mathcal{U}(0,1)$ raspodjelom onda je $\lfloor m \cdot U \rfloor$ prirodan broj iz skupa $\{0,\ldots,m-1\}$. Dodavanjem jedinice dobija se broj iz skupa $\{1,\ldots,m\}$. Ovakvim načinom biranja slučajnog prirodnog broja problem se sveo na paralelno generisanje brojeva sa $\mathcal{U}(0,1)$ raspodjelom a to smo opisali u 4.2.

Dakle, na svakom od *nproc* procesora bira se slučajan prirodan broj iz $\{1,\ldots,V_k^n\}$ i pomoću var^{-1} dobija se odgovarajuća mu vari-

jacija.

Na i-tom, $i=\overline{0,nproc-1}$ procesoru obavlja se sljedeći posao:

Algoritam *sl_var*

1 Postaviti vrijednosti za Z_0, m, a, c

2 $A=1,\ C=0$

3 FOR $j=0,nproc-1$

4 $A=(aA) mod\ m$

5 $C=(aC+c) mod\ m$

6 IF $(j=i)$ THEN

7 $Z=(AZ_0+C) mod\ m$

8 ENDIF

9 NEXT k

(* definisan je generator tipa 2 za $k=nproc$ i njegovo sjeme *)

10 $r=Z/m$

11 $sluč\_int=\lfloor r \cdot V_k^n \rfloor+1$

(*dobija se slučajan prirodan broj iz skupa $\{1,\ldots,V_k^n\}$ *)

12 $kombinv(n,k,sluč\_int,v_1,\ldots,v_k)$

(* dobijena je varijacija jednaka $komb^{-1}(n,k,sluč\_int)$ *)

13 WRITE $v_1,\ldots,v_k$

14 $Z=(AZ+C) mod\ m$

Koraci u linijama od 10. do 14. ponavljaju se.

Programska realizacija ovog algoritma data je fortranskim datotekama *sl_var1*, *sl_var2*, *sl_var3*, *sl_var4* i konfiguracionom datotekom *sl_var.cfg*.

4.6.2 Permutacije

Ako se u prethodnom algoritmu izvrši zamjena $k=n$ dobija se algoritam za generisanje slučajnih permutacija.

Razmotrimo pitanje paralelizacije sekvencijalnog algoritma za generisanje jedne slučajne permutacije.

Sekvencijalni algoritam transformiše datu permutaciju $p=p_1 \ldots p_n$ u slučajnu na sljedeći način:

```
FOR i=n,2,-1
    p_i ↔ p_rand(1,i)  (* mijenjaju mjesta *)
NEXT i
```

$Rand(1,i)$ je slučajan broj iz skupa $\{1,\ldots,i\}$. Paralelni algoritam dizajniramo za SIMD mašinu sa djeljivom memorijom koja ima $n-1$ procesora.

Procesor P_i, $i=\overline{1,n-1}$, generiše slučajan prirodan broj iz skupa $\{1,\ldots,i+1\}$ i smiješta ga u djeljivu mamoriju recimo u y_i. Onda se vrše izmjene

```
FOR i=n,2,-1
    p_i ↔ p_{y_i}
NEXT i
```

Ovako je postignuto da se u jednom taktu proizvede $n-1$ slučajan broj, dok se kod sekvencijalnog slučaja to mora obaviti kroz $n-1$ takt.

4.6.3 Kombinacije

Ideja i algoritam su analogni algoritmu *sl_var* opisanom u

4.5.1. Jedina izmjena je da se poziv u liniji 11 algoritma *sl_var* zamijeni rečenicom *kombinv*$(n,k,c_1,\ldots,c_k)$.

4.6.4 Podskupovi

Algoritam je zasnovan na korespodenciji koja postoji između partitivnog skupa nekog skupa od n elemenata i skupa $\{0,1,\ldots,2^n-1\}$.

Neka je $X=\{x_1,\ldots,x_n\}$ skup čije podskupove tražimo i $b_1\ldots b_n$ binarni zapis nekog od brojeva iz skupa $\{0,1,\ldots,2^n-1\}$ onda $b_1\ldots b_n$ odgovara podskupu $\{x_i|b_i=1\}$.

Ideja za paralelni algoritam je ista kao kod algoritma *sl_var*. U pseudo kodu za algoritam *sl_var* zamjenjujemo rečenicu koja se nalazi u 11. liniji rečenicom *sluč_int*$=\lfloor 2^n\cdot r\rfloor$, poziv u liniji 12 mijenjamo pozivom *bin*$(sluč\_int,n,b_1,\ldots,b_n)$, i rečenicu u liniji 13 rečenicama: 13.1. FOR $i=1,n$

13.2. IF $(b_i\neq 0)$ THEN

13.3. WRITE x_i

13.4. ENDIF

13.5. NEXT i .

Procedura *bin*$(m,n,b_1,\ldots,b_n)$ daje binarni zapis $b_1\ldots b_n$ broja m.

5. ZAKLJUČAK

Predložene metode generisanje kombinatornih i slučajnih objekata posmatramo kao početnu fazu pravljenja elemenata potrebnih za bilo koju složenu aplikaciju stohastičke simulacije koja bi se eventualno realizovala na opremi koja nam je na raspologanju. Imajući to u vidu, posebno smo posmatrali algoritme koji se stvarno mogu realizovati na ovoj opremi.

Predlog za proširenje skupa ovih algoritama je da se naprave paralelni izvori slučajnosti sa diskretnim distribucijama. Osim predloženih načina paralelizacije ovdje bi se paralelizovalo i pretraživanje tabela vjerovatnoća.

Naš drugi cilj je bio tehničke prirode, da razradimo mogućnosti simulacije djeljive memorije na transpjuteru. Time smo obezbijedili da možemo dizajnirati algoritme koji se samo uz izmjenu nekih tehničkih detalja mogu implementirati na opremi koja ima stvarnu mogućnost dijeljenja memorije među procesorima.

Postoji mogućnost simuliranja složenih situacija koje se javljaju u momentu kada se procesori "bore" za djeljive resurse, pa što se tehničkog aspekta tiče predlog je da se te mogućnosti razrađuju.

Ideje opisana u ovom radu uzete su u obzir prilikom definisanja projekta koji će ispitivati komutacioni procesor specijalizovan za prenos podataka komutacijom paketa. Ovo je polje imitacionog modeliranja računarskih sistema pa je potreba za brzim generisanjem slučajnih objekata velika.

LITERATURA

1. Aho, Alfred V., Hopcroft, John E. and Ullman, Jeffrey D. The Design and Analysis of Computer Algorithus, London [etc.], Addison - Wesley Publishing Company, 1976.

2. Akl, S[elim] G. A comparison of combination generation methods. In: ACM Transactions on Mathematical Software, Vol.7, No.1, March 1981, 42-45.

3. Akl, S [elim] G. Adaptive and optimal parallel algorithms for enumerating permutation and combinations. In: The Computer Journal, Vol.30, NO.5, 1987, 433-436.

4. Akl, Selim G. The Design and Analysis of Parallel Algorithms, London, Prentice - Hall International, cop.1989.

5. Brower, Steven. Introduction to Parallel Programming, Boston [etc.], Academic Press,cop.1989.

6. Chen, G.H., and Chern, M - S. Parallel Generation of Permutations and Combinations. In: BIT, Vol.26, 1986, 277-283.

7. Cosnard, M. and Ferreria A.G. Generating Permutations on VLSI Suitable Linear Network. In: The Computer Journal, Vol. 32, No. 6, 1989, 571-572.

8. Duncan, Ralph. A Survey of Parallel Computer Architectures. In: Survey & Tutorial series, 1990 February, 5-15.

9. Fox, Geofrey C. ... et al. Solving Problems on Concurrent Processors, Vol. 1, London [etc.], Prentice - Hall International, cop 1989.

10. Gupta, Phalguni and G.P. Bhattacharjee. Parallel Generation of Permutations. In: The Computer Journal, vol.26. No.2, 1983, 97-105.

11. Knuth, Donald E. The Art of Computer Programming, London [etc.], Addison - Wesley Publishing Company, cop. 1969. (Addison - Wesley series in Computer science and information processing, vol.2)

12. Mijajlović, Ž [arko] i Peruničić P. A Random Search Algorithm for Finding Minium. U: Matematički vjesnik, 38 (1986), 181-195.

13. Nijenbuis, A., and Wilt, H.S. Combinatorial Algorithus, New York, Academic, 1978.

14. Quinn, Michall J. Designing Efficient Algorithms for Parallel Computers, New York [etc.], McGraw - Hill Book Company, cop. 1987. (Supercomputing and artifical intelligence).

15. Reingold, Edward M. Nievergelt Jurg and Deo Narsingh. Combinatorial Algorithms, New Jersey, Prentise - Hall, Englewood Cliffs, 1977.

16. Ripley, Brian D. Stochastic simultation, New York [etc.], John Wiley & Sons, cop. 1987 (Wiley series in probability and mathematical statistics. Applied probability and statistics).

PRILOG

Konfiguraciona datoteka

Konfiguraciona datoteka se piše na 3L konfiguracionom jeziku. Opisaćemo osnovne elemente ovog jezika. Osnovna pravila pisanja:

- ne pravi se razlika između velikih i malih slova,
- linija se nastavlja znakom -,
- prazni prostor unutar linije i među linijama se ignoriše,
- sve što se u liniji nalazi iza znaka ! tretira se kao komentar.

Konfiguraciona datoteka jedne aplikacije mora sadržati sljedeće informacije:

- koji procesori postoje u mreži,
- fizičke veze među procesorima,
- broj i/O portova svakog zadatka u aplikaciji,
- memorijske zahtjeve zadataka,
- raspored zadataka po procesorima,
- veze između ulaznih i izlaznih portova zadataka.

Osnovne naredbe

Za naznačavanje procesora koji je u mreži služi naredba PROCESSOR.

Sintaksa: PROCESSOR ime procesora

Primjer: PROCESSOR HOST !PC

PROCESSOR T1 !jedan od procesora ploče.

Naredba WIRE služi za naznačavanje fizičkih veza (žica) među procesorima. Svaki procesor u mreži ima 4 Inmos veze numerisane sa 0 do 3. Žice imaju imena ali ako ne želimo da ih naznačavamo možemo umjesto imena žice upotrijebiti znak ?.

Sintaksa: WIRE ime1 proc1[v_proc1] proc2[v_proc2]

Ova rečenica znači da su procesori proc1 i proc2 povezani žicom čije je ime ime1 tako što su spojene veze v_proc1 i v_proc2.

Primjer: WIRE ? T1[1] T2[0]

Na slici 1 je prikazana mreža modela na kome radimo i na njoj se vidi na koji je način izvršeno fizičko povezivanje procesora.

Ne moramo opisati kompletnu mrežu. Dovoljno je da naznačimo ono što je potrebno za datu aplikaciju.

Pomoću naredbe TASK se naznačava koliko zadaci imaju ulaznih i izlaznih portova i koliko se memorije rezerviše za zadatke. Ukoliko se na jednom procesoru nalazi samo jedan zadatak onda se ne mora rezervisati memorija, međutim ako na jednom procesoru postoji više zadataka onda rezervisanje memorije nije potrebno za samo jedan od njih.

Sintaksa: TASK ime_T ins=br1 outs=br2 data=br3

Ova rečenica znači da zadatak čije je ime ime_T ima br1 ulaznih i br2 izlaznih portova i za njega je rezavisan prostor od br3 bajta.

Primjer: TASK var2 ins=1 outs=1

Pomoću naredbe PLACE naznačava se koji zadatak je smješten na kojem procesoru.

Sintaksa: PLACE ime zadatka ime procesora

Primjer: PLACE var2 T2

Veze između zadataka (portova zadataka) se uspostavljaju naredbom CONNECT.

Sintaksa: CONNECT ime1 P_Z1 zad1[izl_port] zad2[ul_port]

Rečenica znači da su izlazni port izl_port i ulazni port zadatka 2 ul_port vezani vezom koja se zove ime1. Ukoliko nećemo da

naglašavamo ime onda možemo staviti znak ?.

Primjer: CONNECT ? var1[3] var3[0]

izlazni port 3 zadatka var1 povezan je sa ulaznim portom 0 zadatka var3.

Ovdje se mora voditi računa o redosljedu tj. prvi se piše zadatak čiji se izlazni port vezuje.

Primijetimo da konfiguracionim datotekama koje su date u prilogu stalno pojavljuju zadaci afserver i filter.

Zadatak atserver je MS DOS izvršna datoteka .Afserver se izvršava na PC-ju (host). Njegov zadatak je da transferuje izvršne datoteke do transpjutera. Osim toga, ovaj program igra ulogu servera obrađujući I/O zahtjeve transpjutera.

Između afserver-a i korisničkog programa nalazi se zadatak filter koji se izvršava paralelno sa njima i prenosi poruke u oba smjera.

Kompilacija, povezivanje i izvršavanje

Opisaćemo osnovne korake potrebne da se programi napisani na 3L Parallel Fortranu pripreme za izvršavanje na transpjuterskoj ploči sa procesorima T800.

Uzmimo primjer aplikacije za generisanje slučajnih varijacija. Ta aplikacija se sastoji od 4 zadatka, po jedan zadatak na jednom procesoru. Zadaci su u izvornom kodu dati datotekama sl_var1.f77, sl_var2.f77, sl_var3.f77 i sl_var4.f77 a konfiguraciona datoteka za ovu aplikaciju se zove sl_var.cfg. Kompilacija se vrši naredbom t8f.

Dakle, poslije koraka

```
>t8f sl_var1
>t8f sl_var2
>t8f sl_var3
>t8f sl_var4
```

imamo object datoteke sl_var1.bin, sl_var2.bin, sl_var3.bin i sl_var4.bin.

Zadatak koji se nalazi na procesoru koji je povezan sa PC-jem linkuje se naredbom t8ftask a ostali naredbom t8fstask.

Dakle, poslije koraka:

```
>t8ftask sl_var1
>t8fstask sl_var2
>t8fstask sl_var3
>t8fstask sl_var4
```

dobijaju se datoteke sl_var1.b4, sl_var2.b4, sl_var3.b4 i sl_var4.b4.

Program koji se zove configurer na osnovu konfiguracione datoteke aplikacije povezuje njene zadatke i uobličava je za izvršavanje.

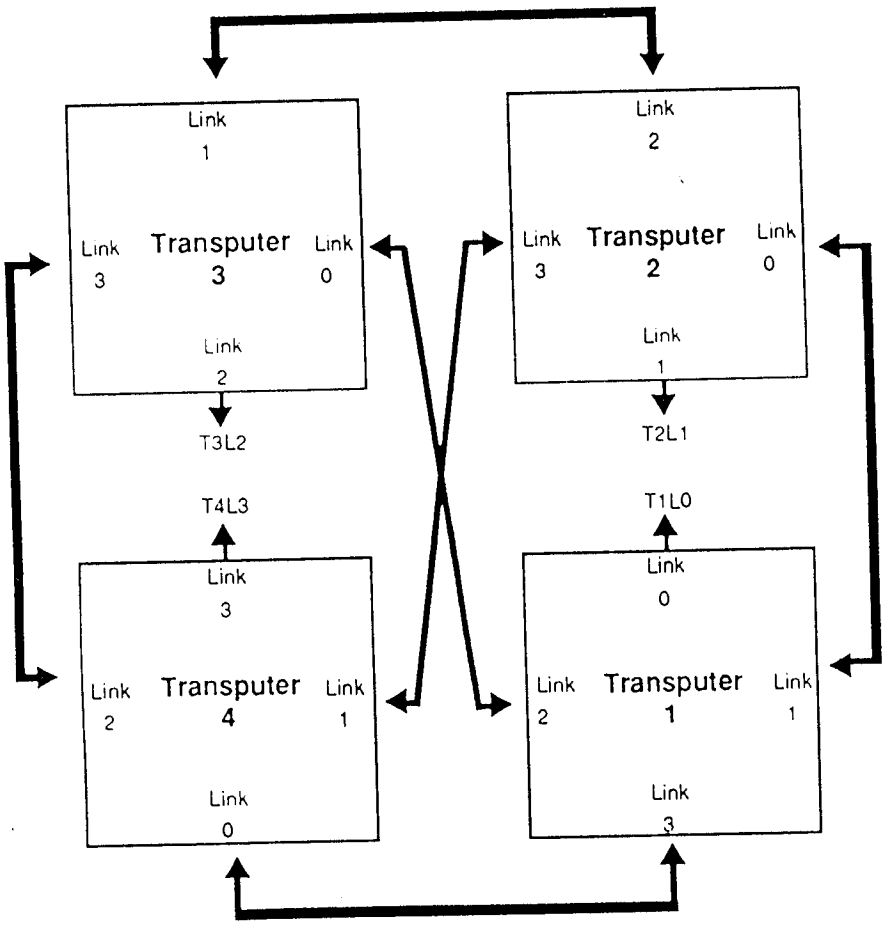
Pozivanje programa configurer za ovu aplikaciju izgleda

```
>config sl_var.cfg sl_var.app
```

Sada je sve spremno za izvršavanje koje se vrši pomoću afserver-a.

Pozivanje programa afserver za aplikaciju sl_var izgleda:

```
>afserver -: b sl_var.app
```

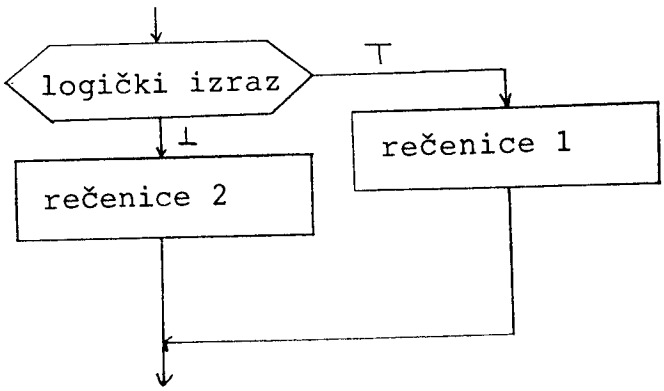


Jezik koji je korišćen za opis algoritama

1. Sintaksa:

```
IF logički izraz THEN
   rečenice 1
ELSE
   rečenice 2
ENDIF
```

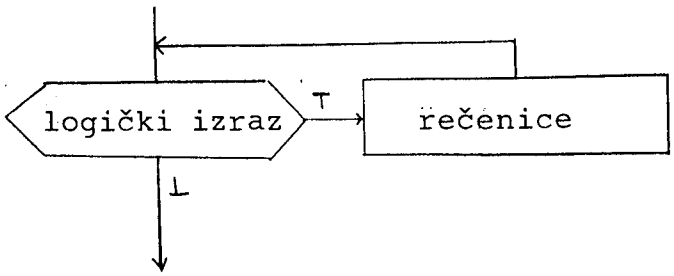
semantika:



2. sintaksa:

```
DO WHILE logički izraz
   rečenice
ENDDO
```

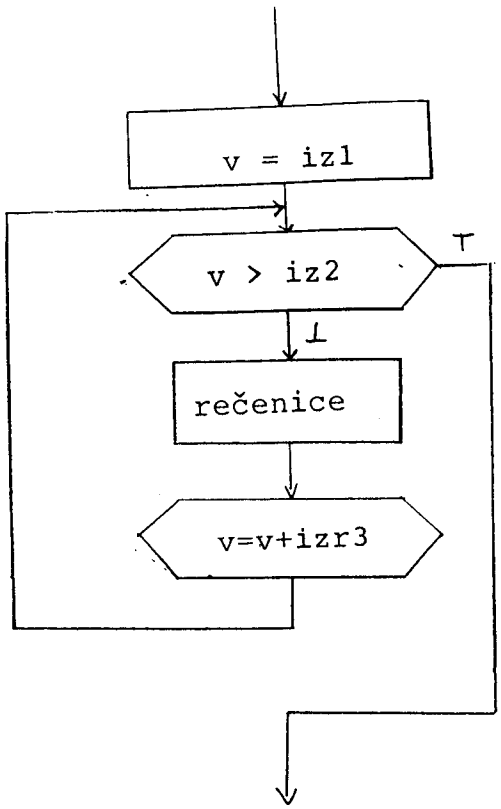
semantika:



3.Sintaksa

```
FOR v=iz1,iz2,iz3
    rečenice
NEXT v
```

semantika:



ili v<iz2 ako je iz3<0

4. Ako se na procesoru P_i izvršava naredba

RECEIVE $v_1, \ldots, v_n$ FROM P_k

znači da procesor P_i prima vrijednosti $v_1, \ldots, v_n$ od procesora P_k .

Ako se na procesoru P_i izvršava naredba:

SEND $v_1, \ldots, v_n$ TO P_k

znači da procesor P_i šalje vrijednosti $v_1, \ldots, v_n$ procesoru P_k .

5. Naredba

```
FOR i=1,n Pi DO IN PARALLEL IN SM
    rečenice
ENDFOR
```

znači da procesori paralelno izvršavaju rečenice u djeljivoj memoriji.

PROGRAMSKI PRILOG

SHIFT1

GPSN

SHIFT1.F77

SHIFT1.CFG

```
!konfiguraciona datoteka za zadatak shift1
! shift1.cfg
!
! Hardware
!
processor host
processor t1
processor t2
processor t3
wire ? host[0] t1[0]              !anonymous wire connecting PC to transputer
wire ? t1[1] t2[0]
wire ? t1[2] t3[0]

!
! Task declarations indicating channel I/O ports and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K

task shift1  ins=5 outs=5 data=500k
!
! Assign software tasks to physical processors
!
place afserver host

place shift1 t1
place filter t1


!
! Set up the connections between the tasks.
!
connect ? afserver[0] filter[0]
connect ? filter[0] afserver[0]

connect ? filter[1] shift1[1]
connect ? shift1[1] filter[1]
```

```
      PROGRAM SHIFT1
*************************************************************************
c  Vrsi se paralelizacija rada generatora slucajnih bitova.Simulira se
c  rad cetri procesa P1,P2,P3 i P4 koji dijele zajednicki prostor kroz
c  common zonu ZONA1.
c  Glavna nit P1 kreira niti P2,P3 i P4 koje imaju isti prioritet kao
c  ona.
c  niz bitova koji se generisu je niz DATA
*************************************************************************
c  Uvodjenje datoteke paketa THREAD
       INCLUDE 'thread.inc'
c  Radni prostori niz2,niz3 i niz4 za niti P2,P3,P4
       INTEGER niz2(2500),niz3(2500),niz4(2500)
       INTEGER data
       EXTERNAL P2,P3,P4
c Definisanje zajednicke zone
       COMMON/zona1/data(300),ip2,ip3,ip4
c      Prioritet tekuce niti koji kao argument treba predati ostalima
       iprio=f77_thread_priority()
       PRINT *,'duzina niza ?'
       READ *,iduz
c postavljanje pocetnih bitova
       DO 130 i=1,9
130       data(i)=1
100    CONTINUE

c Kreiranje niti P2,P3 i P4
       CALL F77_THREAD_START(P2,niz2,2500*4,iprio,1,iduz)
       CALL F77_THREAD_START(P3,niz3,2500*4,iprio,1,iduz)
       CALL F77_THREAD_START(P4,niz4,2500*4,iprio,1,iduz)
       PRINT *,'kreirao sam'
c Proces (nit) P1 radi svoj dio posla
       DO 122 i=10,iduz,4
       data(i)=mod(data(i-9)+data(i-5),2)
122    CONTINUE
c Da li su ostali zavrsili posao? ako jeste stampanje niza
900    IF((ip2+ip3+ip4).eq.3)THEN
                                PRINT *,(data(i),i=1,iduz)
                                STOP
                             ELSE
                                GO TO 900

       ENDIF
       END

c**************PROCES P2*****************************
       SUBROUTINE P2(KK)
       INTEGER data
       COMMON/zona1/data(300),ip2,ip3,ip4
       ip2=0
       DO 100 i=11,kk,4
           data(i)=mod(data(i-9)+data(i-5),2)
100    CONTINUE
       ip2=1
       END

c**************PROCES P3*****************************
       SUBROUTINE P3(kk)
       INTEGER data
```

```
      COMMON/zona1/data(300),ip2,ip3,ip4
      ip3=0
      DO 100 i=12,kk,4
          data(i)=mod(data(i-9)+data(i-5),2)
100   CONTINUE
      ip3=1
      END

c*****************PROCES P4*********************************
      SUBROUTINE P4(kk)
      INTEGER data
      COMMON/zona1/data(300),ip2,ip3,ip4
      ip4=0
      DO 100 I=13,KK,4
         data(i)=mod(data(i-9)+data(i-5),2)
100   CONTINUE
      ip4=1
      END
```

SHIFT2

GFSR

SHIFT2.F77

SHIFT2.CFG

```
!konfiguraciona datoteka za zadatak shift2
! shift2.cfg
!
! Hardware
!
processor host
processor t1
processor t2
processor t3
wire ? host[0] t1[0]              !anonymous wire connecting PC to transputer
wire ? t1[1] t2[0]
wire ? t1[2] t3[0]

!
! Task declarations indicating channel I/O ports and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K

task shift2  ins=5 outs=5 data=500k
!
! Assign software tasks to physical processors
!
place afserver host

place shift2 t1
place filter t1


!
! Set up the connections between the tasks.
!
connect ? afserver[0] filter[0]
connect ? filter[0] afserver[0]

connect ? filter[1] shift2[1]
connect ? shift2[1] filter[1]
```

```
      PROGRAM SHIFT2
*************************************************************************
C Vrsi paralelizaciju rada uopstenog generatora slucajnih brojeva,pomocu
c  3 - bitnih prirodnih brojeva.Simulira se rad tri procesa P1,P2 i P3.
c  Proces (nit) P1 kreira P2 i P3 koji su istog prioriteta kao i ona.
c  Matrica mat sadrzi 3-bitne prirodne brojeve
c  P1 formira prirodne brojeve od vrsta matrice mat koja je u djeljivoj
c  memoriji koju smo simulirali preko common zone zona1
c  Napomena : pomjeranja se mogu parametrizovati.Dakle ,vrijednosti
c  za s i r se mogu ucitavati i prenijeti nitima kao ulazni argumenti.
c  Ovdje to nije uradjeno nego program radi za s=7 i r=1
c
***************PROCES P1 ***************************************************
c Proces P1 kreira ostale,radi na prvoj koloni , formira prirodne brojeve
c i stampa.
C Uvodjenje datoteke paketa THREAD
      INCLUDE 'thread.inc'
c Radni prostori za P2 i P3
      INTEGER niz2(2500),niz3(2500)
      EXTERNAL P2,P3
      DIMENSION niiz(3000)
c Definisanje djeljive memorije
      COMMON/zona1/mat(300,3),ipp2,ipp3,ipp4
c Pocetne vrste matrice
      mat(1,1)=0
      mat(1,2)=0
      mat(1,3)=0
      mat(2,1)=0
      mat(2,2)=0
      mat(2,3)=1
      mat(3,1)=0
      mat(3,2)=1
      mat(3,3)=0
      mat(4,1)=0
      mat(4,2)=1
      mat(4,3)=1
      mat(5,1)=1
      mat(5,2)=0
      mat(5,3)=0
      mat(6,1)=1
      mat(6,2)=0
      mat(6,3)=1
      mat(7,1)=1
      mat(7,2)=1
      mat(7,3)=0

      PRINT *,'koliko vrsta ?'
      READ *,ivrs
c Definisanje prioriteta tekuce niti
      IPRIO=F77_THREAD_PRIORITY()
      PRINT *,((mat(I,J),J=1,3),I=1,7)
c Kreiranje niti P2 i P3
      CALL F77_THREAD_START(P2,NIZ2,2500*4,IPRIO,1,IVRS)
      CALL F77_THREAD_START(P3,NIZ3,2500*4,IPRIO,1,IVRS)

      PRINT *,'Kreirao sam'
c P1 obavlja svoj dio posla ,radi na prvoj koloni
      DO 111 i=8,ivrs
111   mat(i,1)=mod(mat(i-7,1)+mat(i-6,1),2)
```

```
c Da li su P2 i P3 zavrsili posao? Ako jesu onda formiram prirodne brojeve
c i stampam ih (zbog provjere)
9000    IF(ipp2+ipp3.eq.2)THEN
                          DO 112 i=1,ivrs
                             niiz(i)=mat(i,1)*4+mat(i,2)*2+mat(i,3)
                             PRINT *,niiz(i)
                          CONTINUE
112                      ELSE
                          GO TO 9000

        ENDIF
        END

**********PROCES P2 RADI DRUGU KOLONU**************************

        SUBROUTINE P2(KK)
        COMMON/zona1/mat(300,3),ipp2,ipp3
        ipp2=0
        DO 22 i=8,kk
              mat(i,2)=mod(mat(i-7,2)+mat(i-6,2),2)
22      CONTINUE
        ipp2=1
        END

**********PROCES P3 RADI TRECU KOLONU*******************************

        SUBROUTINE P3(KK)
        COMMON/zona1/mat(300,3),ipp2,ipp3
        ipp3=0
        DO 22 i=8,kk
              mat(i,3)=mod(mat(i-7,3)+mat(i-6,3),2)
22      CONTINUE
        ipp3=1
        END
```

SLUČAJNE VARIJACIJE

SL_VAR1.F77

SL_VAR2.F77

SL_VAR3.F77

SL_VAR4.F77

SL_VAR.CFG

```
!Konfiguraciona datoteka za slucajne varijacije
! sl_var.cfg
!
! Hardware
!
processor host
processor t1
processor t2
processor t3
processor t4
wire ? host[0] t1[0]              !anonymous wire connecting PC to transputer
wire ? t1[1] t2[0]
wire ? t1[2] t3[0]
wire ? t1[3] t4[0]
!
! Task declarations indicating channel I/O ports and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K

task sl_var1 ins=5 outs=5
task sl_var2 ins=1 outs=1
task sl_var3 ins=1 outs=1
task sl_var4 ins=1 outs=1

!
! Assign software tasks to physical processors
!
place afserver host

place sl_var1 t1
place filter t1
place sl_var2 t2
place sl_var3 t3
place sl_var4 t4
!
! Set up the connections between the tasks.
!
connect ? afserver[0] filter[0]
connect ? filter[0] afserver[0]

connect ? filter[1] sl_var1[1]
connect ? sl_var1[1] filter[1]

connect ? sl_var1[2] sl_var2[0]
connect ? sl_var1[3] sl_var3[0]
connect ? sl_var2[0] sl_var1[2]
connect ? sl_var3[0] sl_var1[3]
connect ? sl_var1[4] sl_var4[0]
connect ? sl_var4[0] sl_var1[4]
```

```
      program sl_var1
c     generise permutacije sve po kodu
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in(2:4),out(2:4),pro,p(100),q(100)
c     definisanje kanala
      do 1000 i=2,4
      in(i)=f77_chan_in_port(i)
      out(i)=f77_chan_out_port(i)
1000  continue
      print *,'unesi vrijednosti za n i m'
      read *,n,m
      print *,'koliko  varijacija'
      read *,kol
      call iclock(isek)
c     poslati vrijednosti za m i n i kol ostalima
      koll=kol/4
      do 900 i=2,4
      call f77_chan_out_word(n,out(i))
      call f77_chan_out_word(m,out(i))
900   call f77_chan_out_word(koll,out(i))
c     sracunati npm
      npm=n
      do 5000 i=1,m-1
      npm=npm*(n-i)
5000  continue
      seed =7.d0
      pro=0
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      modd=2.**16+1
      do 120 i=0,3
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
      do 8000 jk=1,kol/4
      r=r1/modd
      kod=npm*r
      kod=kod+1
      kod1=kod
      call varinv(kod1,n,m,p)
      r1=dmod(a1*r1+c1,modd)
8000  continue
      call iclock(isekk)
      print *,isekk-isek
      end
c*************************
      subroutine varinv(d,n,m,p)
      integer s(100),a,b,d,p(1)
      d=d-1
      do 2 i=1,n
2     s(i)=0
      a=1
      do 3 i=m-1,1,-1
```

```
            a=a*(n-m+i)
3           continue
            do 4 i=1,m
            b=d/a
            d=d-a*b
            if(n.gt.i)then
              a=a/(n-i)
            endif
            k=0
            j=0
9000         if (k.lt.(b+1))then
                j=j+1
                if(s(j).eq.0)then
                   k=k+1
                endif
              go to 9000
             endif
             p(i)=j
             s(j)=1
4           continue
            end
```

```
      program sl_var2
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in,out,pro,p(100),q(100)
c     definisanje kanala

      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)

c     primiti vrijednosti za n i m i kol/3
      call f77_chan_in_word(n,in)
      call f77_chan_in_word(m,in)
      call f77_chan_in_word(kol3,in)
c     sracunati npm
      npm=n
      do 5000 i=1,m-1
      npm=npm*(n-i)
5000  continue
      seed =7.d0
      pro=1
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      modd=2.**16+1
      do 120 i=0,3
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
      do 8000 ip=1,kol3
      r=r1/modd
      kod=npm*r
      kod=kod+1
      kod1=kod
      call varinv(kod1,n,m,p)
c     poslati varijaciju  prvom
      do 8001 i=1,m
      call f77_chan_out_word(p(i),out)
8001  continue
      r1=dmod(a1*r1+c1,modd)
8000  continue
      end
c*************************
      subroutine varinv(d,n,m,p)
      integer s(100),a,b,d,p(1)
      d=d-1
      do 2 i=1,n
2     s(i)=0
      a=1
      do 3 i=m-1,1,-1
      a=a*(n-m+i)
3     continue
      do 4 i=1,m
      b=d/a
      d=d-a*b
      if(n.gt.i)then
```

```
          a=a/(n-i)
        endif
        k=0
        j=0
9000     if (k.lt.(b+1))then
            j=j+1
            if(s(j).eq.0)then
               k=k+1
            endif
          go to 9000
         endif
         p(i)=j
         s(j)=1
4       continue
        end
```

```
      program sl_var3
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in,out,pro,p(100),q(100)
c     definisanje kanala

      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)

c     primiti vrijednosti za n i m i kol/3
      call f77_chan_in_word(n,in)
      call f77_chan_in_word(m,in)
      call f77_chan_in_word(kol3,in)
c     sracunati npm
      npm=n
      do 5000 i=1,m-1
      npm=npm*(n-i)
5000  continue
      seed =7.d0
      pro=2
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      modd=2.**16+1
      do 120 i=0,3
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
      do 8000 ip=1,kol3
      r=r1/modd
      kod=npm*r
      kod=kod+1
      kod1=kod
      call varinv(kod1,n,m,p)
c     poslati varijaciju prvom
      do 8001 i=1,m
      call f77_chan_out_word(p(i),out)
8001  continue
      r1=dmod(a1*r1+c1,modd)
8000  continue
      end
c*************************
      subroutine varinv(d,n,m,p)
      integer s(100),a,b,d,p(1)
      d=d-1
      do 2 i=1,n
2     s(i)=0
      a=1
      do 3 i=m-1,1,-1
      a=a*(n-m+i)
3     continue
      do 4 i=1,m
      b=d/a
      d=d-a*b
      if(n.gt.i)then
```

```
          a=a/(n-i)
        endif
        k=0
        j=0
9000     if (k.lt.(b+1))then
            j=j+1
            if(s(j).eq.0)then
               k=k+1
            endif
          go to 9000
         endif
         p(i)=j
         s(j)=1
4       continue

        end
```

```
      program sl_var4
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in,out,pro,p(100),q(100)
c     definisanje kanala

      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)

c     primiti vrijednosti za n i m i kol/3
      call f77_chan_in_word(n,in)
      call f77_chan_in_word(m,in)
      call f77_chan_in_word(kol3,in)
c     sracunati npm
      npm=n
      do 5000 i=1,m-1
      npm=npm*(n-i)
5000  continue
      seed =7.d0
      pro=3
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      modd=2.**16+1
      do 120 i=0,3
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
      do 8000 ip=1,kol3
      r=r1/modd
      kod=npm*r
      kod=kod+1
      kod1=kod
      call varinv(kod1,n,m,p)
c     poslati varijaciju prvom
      do 8001 i=1,m
      call f77_chan_out_word(p(i),out)
8001  continue
      r1=dmod(a1*r1+c1,modd)
8000  continue
      end
c*************************
      subroutine varinv(d,n,m,p)
      integer s(100),a,b,d,p(1)
      d=d-1
      do 2 i=1,n
2     s(i)=0
      a=1
      do 3 i=m-1,1,-1
      a=a*(n-m+i)
3     continue
      do 4 i=1,m
      b=d/a
      d=d-a*b
      if(n.gt.i)then
```

```
          a=a/(n-i)
        endif
        k=0
        j=0
9000     if (k.lt.(b+1))then
            j=j+1
            if(s(j).eq.0)then
               k=k+1
            endif
          go to 9000
         endif
         p(i)=j
         s(j)=1
4       continue

        end
```

N O R M A L N A R A S P O D J E L A

NORM1.F77

NORM2.F77

NORM3.F77

NORM4.F77

NORM.CFG

```
! konfiguraciona datoteka N(0,1)
! norm.cfg.CFG
!
! Hardware
!
processor host
processor t1
processor t2
processor t3
processor t4
wire ? host[0] t1[0]              !anonymous wire connecting PC to transputer
wire ? t1[1] t2[0]
wire ? t1[2] t3[0]
wire ? t1[3] t4[0]
!
! Task declarations indicating channel I/O ports and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K

task norm1 ins=5 outs=5
task norm2 ins=1 outs=1
task norm3 ins=1 outs=1
task norm4 ins=1 outs=1

!
! Assign software tasks to physical processors
!
place afserver host

place norm1 t1
place filter t1
place norm2 t2
place norm3 t3
place norm4 t4
!
! Set up the connections between the tasks.
!
connect ? afserver[0] filter[0]
connect ? filter[0] afserver[0]

connect ? filter[1] norm1[1]
connect ? norm1[1] filter[1]

connect ? norm1[2] norm2[0]
connect ? norm1[3] norm3[0]
connect ? norm2[0] norm1[2]
connect ? norm3[0] norm1[3]
connect ? norm1[4] norm4[0]
connect ? norm4[0] norm1[4]
```

c P_1

```
      program norm1
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in(2:4),out(2:4),pro
      print *,'koliko hoces normalnih'
      read *,kol
      call iclock(isek)
c     definisanje kanala
      do 1000 i=2,4
      in(i)=f77_chan_in_port(i)
      out(i)=f77_chan_out_port(i)
1000  continue
      seed =7.d0
      pro=0
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      do 120 i=0,11
      modd=2.**16+1
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.3*pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
c     pocinje da generise svoje
      do 100 l=1,kol
      s=0.
      do 780 l1=1,2
       s=s+r1/modd
       r1=dmod(a*r1+c,modd)
780   continue
      s=s+r1/modd
c     tu je zavrsio svoje sumiranje sada prihvati od ostalih
      do 781 i=2,4
      call f77_chan_in_word(s1,in(i))
      s=s+s1
781   continue
      s=s-6.
      r1=dmod(a1*r1+c1,modd)
100   continue
      call iclock(isekk)
      print *,isekk-isek
      end
```

c P_2

```
      program norm2
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in,out,pro
c     definisanje kanala
      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)
      seed =7.d0
      pro=1
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      do 120 i=0,11
      modd=2.**16+1
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.3*pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
c     treba da pocne da sumira
100   s=0.
      do 780 l=1,2
        s=s+r1/modd
        r1=dmod(a*r1+c,modd)
780   continue
      s=s+r1/modd
c     treba poslati svoju sumu
      call f77_chan_out_word(s,out)
      r1=dmod(a1*r1+c1,modd)
      go to 100
      end
```

c P_3

```
      program norm3
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in,out,pro
c     definisanje kanala
      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)
      seed =7.d0
      pro=2
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      do 120 i=0,11
      modd=2.**16+1
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.3*pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
c     treba da pocne da sumira
100   s=0.
      do 780 l=1,2
        s=s+r1/modd
        r1=dmod(a*r1+c,modd)
780   continue
      s=s+r1/modd
c     treba poslati svoju sumu
      call f77_chan_out_word(s,out)
      r1=dmod(a1*r1+c1,modd)
      go to 100
      end
```

c P_4

```
      program norm3
      include 'chan.inc'
      double precision a,a1,c,c1,r,r1,seed,modd,drugi,treci
      integer in,out,pro
c     definisanje kanala
      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)
      seed =7.d0
      pro=3
      a1=1.d0
      c1=0.d0
      a=75.d0
      c=0.d0
      do 120 i=0,11
      modd=2.**16+1
      a1=dmod(a*a1,modd)
      c1=dmod(a*c1+c,modd)
      if(i.eq.3*pro)then
      r1=dmod(a1*seed+c1,modd)
      endif
120   continue
c     treba da pocne da sumira
100   s=0.
      do 780 l=1,2
        s=s+r1/modd
        r1=dmod(a*r1+c,modd)
780   continue
      s=s+r1/modd
c     treba poslati svoju sumu
      call f77_chan_out_word(s,out)
      r1=dmod(a1*r1+c1,modd)
      go to 100
      end
```

INTEGRAL

INT1.F77

INT2.F77

INT3.F77

INT4.F77

INT.CFG

```
!Konfiguraciona datoteka za integral


! int.cfg
!
! Hardware
!
processor host
processor t1
processor t2
processor t3
processor t4
wire ? host[0] t1[0]              !anonymous wire connecting PC to transputer
wire ? t1[1] t2[0]
wire ? t1[2] t3[0]
wire ? t1[3] t4[0]
!
! Task declarations indicating channel I/O ports and memory requirements
!
task afserver ins=1 outs=1
task filter ins=2 outs=2 data=10K
task int1 ins=5 outs=5
task int2 ins=1 outs=1
task int3 ins=1 outs=1
task int4 ins=1 outs=1
!
! Assign software tasks to physical processors
!
place afserver host

place int1 t1
place filter t1
place int2 t2
place int3 t3
place int4 t4
!
! Set up the connections between the tasks.
!
connect ? afserver[0] filter[0]
connect ? filter[0] afserver[0]
connect ? filter[1] int1[1]
connect ? int1[1] filter[1]
connect ? int1[2] int2[0]
connect ? int1[3] int3[0]
connect ? int2[0] int1[2]
connect ? int3[0] int1[3]
connect ? int1[4] int4[0]
connect ? int4[0] int1[4]
```

```
c računanje integrala - P0

      include 'chan.inc'
      integer in(2:4),out(2:4),avel,amal,cvel,cmal,seed,proc,rvel
c     uspostaviti veze
      f(u)=u**3
      do 100 i=2,4
      in(i)=f77_chan_in_port(i)
      out(i)=f77_chan_out_port(i)
100   continue
      print *,'koliko tačaka'
      read *,koliko
      call iclock(ISEK1)
      modd=2**16+1
      isuma=0
      proc=0
      avel=1
      cvel=0
      amal=75
      cmal=0
      seed=7
c     poslati svima koliko
      do 200 i=2,4
200      call f77_chan_out_word(koliko,out(i))
c     postaviti seed i avel i cvel za ovaj
      do 300 i=0,3
         avel=mod(avel*amal,modd)
         cvel=mod(amal*cvel+cmal,modd)
         if (i.eq.proc)then
            rvel=mod(avel*seed+cvel,modd)
         endif
300   continue
      do 400 i=1,koliko/4
      rmal=(rvel+0.0)/modd
      rvel=mod(avel*rvel+cvel,modd)
      suma=suma+f(rmal)
400   continue
c sracunate su koordinate
c     ovaj je zavrsio treba od ostalih da primi
      do 500 i=2,4
      call f77_chan_in_word(sumaa,in(i))
      suma=suma+sumaa
500   continue
      kol=koliko/4
      kol1=4*kol
      vjerov=(suma+0.0)/kol1
      print *,vjerov
      call iclock(isek2)
      print *,isek2-isek1
      end
```

```
c računanje integrala P1
      include 'chan.inc'
      integer in,out,avel,amal,cvel,cmal,seed,proc,rvel
c     uspostaviti veze
      f(u)=u**3

      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)
      call f77_chan_in_word(koliko,in)
      kol=koliko/4

      modd=2**16+1
      isuma=0
      proc=1
      avel=1
      cvel=0
      amal=75
      cmal=0
      seed=7


c     postaviti seed i avel i cvel za ovaj
      do 300 i=0,3
         avel=mod(avel*amal,modd)
         cvel=mod(amal*cvel+cmal,modd)
         if (i.eq.proc)then
            rvel=mod(avel*seed+cvel,modd)
         endif
300   continue
      do 400 i=1,koliko/4
         rmal=(rvel+0.0)/modd
         rvel=mod(avel*rvel+cvel,modd)
         suma=suma+f(rmal)
400   continue
c     ovaj je zavrsio treba da posalje rezultat
      call f77_chan_out_word(suma,out)
      end
```

```
c računanje integrala P2
      include 'chan.inc'
      integer in,out,avel,amal,cvel,cmal,seed,proc,rvel
c     uspostaviti veze
      f(u)=u**3
      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)
      call f77_chan_in_word(koliko,in)
      kol=koliko/4
      modd=2**16+1
      isuma=0
      proc=2
      avel=1
      cvel=0
      amal=75
      cmal=0
      seed=7
c     postaviti seed i avel i cvel za ovaj
      do 300 i=0,3
         avel=mod(avel*amal,modd)
         cvel=mod(amal*cvel+cmal,modd)
         if (i.eq.proc)then
            rvel=mod(avel*seed+cvel,modd)
         endif
300   continue
      do 400 i=1,koliko/4
         rmal=(rvel+0.0)/modd
         rvel=mod(avel*rvel+cvel,modd)
         suma=suma+f(rmal)
400   continue
c     ovaj je zavrsio treba da posalje rezultat
      call f77_chan_out_word(suma,out)
      end
```

```
c računanje integrala - P3
      include 'chan.inc'
      integer in,out,avel,amal,cvel,cmal,seed,proc,rvel
c     uspostaviti veze
      f(u)=u**3
      in=f77_chan_in_port(0)
      out=f77_chan_out_port(0)
      call f77_chan_in_word(koliko,in)
      kol=koliko/4
      modd=2**16+1
      isuma=0
      proc=3
      avel=1
      cvel=0
      amal=75
      cmal=0
      seed=7
c     postaviti seed i avel i cvel za ovaj
      do 300 i=0,3
         avel=mod(avel*amal,modd)
         cvel=mod(amal*cvel+cmal,modd)
         if (i.eq.proc)then
            rvel=mod(avel*seed+cvel,modd)
         endif
300   continue
      do 400 i=1,koliko/4
         rmal=(rvel+0.0)/modd
         rvel=mod(avel*rvel+cvel,modd)
         suma=suma+f(rmal)
400   continue
c     ovaj je zavrsio treba da posalje rezultat
      call f77_chan_out_word(suma,out)
      end
```