MATEMATIČKI FAKULTET

UNIVERZITETA U BEOGRADU

PARALELNI ALGORITMI U ARITMETICI

-MAGISTARSKI RAD-

GORAN GOGIĆ

BEOGRAD, 1991.

S A D R Ž A J

0. Uvod

Ovaj rad je zamišljen kao skup paralelnih algoritama za rešavanje osnovnih problema aritmetike. Problemi nalaženja najvećeg zajedničkog delioca dva broja i računanje vrednosti funkcije $\pi(x)$ su ranije rešeni, pa su ovde ti algoritmi izloženi u kraćim crtama. Ostali algoritmi predstavljaju originalne rezultate samim tim što je paralelno programiranje izuzetno nova oblast pa se slični algoritmi još uvek nisu pojavili u literaturi.

Algoritam za nalaženje najvećeg zajedničkog delioca dva broja dali su Chor i Goldreich, ali je za sada taj rezultat nemoguće primeniti u praksi. Tako se već na početku susrećemo sa velikim poteškoćama koje se pojavljuju u paralelnom programiranju.

Problem računanja funkcije $\pi(x)$ ima veoma dugu istoriju. Do sada najbrži sekvencijalan algoritam su dali Lagarias, Miller i Odlyzko, a pošto je moguće izvršiti njegovu idealnu paralelizaciju, on predstavlja danas najbrži paralelni algoritam za računanje funkcije $\pi(x)$. Na žalost, ovaj algoritam nije moguće realizovati na arhitekturama računara koje su danas u opticaju. Stoga je na kraju poglavlja dat predlog za računanje funkcije $\pi(x)$ na postojećim arhitekturama. Iako to nije najbolje rešenje sa teoretske tačke gledišta, stepen paralelizacije koji se postiže njegovom primenom je veoma visok.

U 4.poglavlju je dat paralelni algoritam za nalaženje prostih brojeva iz određenog intervala. Ovaj algoritam je dobijen paralelizacijom poznatog sekvencijalnog algoritma "Eratostenovo sito". Na osnovu dobijenog rezultata pri proceni vremena vidimo da je dobijeni paralelni algoritam idealna paralelizacija odgovarajućeg sekvencijalnog algoritma. Detaljno je razmotren slučaj kada broj procesora neograničeno raste, i dat je algoritam za dobijanje što veće iskorišćenosti procesora.

U 5. poglavlju se ispituje množenje višecifrenih brojeva što je veoma bitna stvar u mnogim algoritmima. U ovom radu je dato rešenja problema u dva slučaja: kada imamo najmoćniju arhitekturu, i tada dobijamo najbolje moguće vreme, i u drugom slučaju kada imamo samo niz linearno povezanih procesora. Ovo drugo rešenje je od velike važnosti zato što se može primeniti i na ostale arhitekture. Navedeni algoritmi su brži od svih do sada poznatih sekvencijalnih algoritama, jer problem rešavaju u linearnom vremenu.

Algoritmi za ispitivanje primalnosti su opisani u 6. poglavlju. Do sada su pravljeni samo u sekvencijalnom obliku, tako da nije bilo moguće uporediti rezultat dobijen u ovom radu sa nekim drugim paralelnim algoritmima. Međutim, ono što ide u prilog našim rezultatima je činjenica da su ovi algoritmi nastali potpunom paralelizacijom brzih sekvencijalnih algoritama. U istom poglavlju dat je i jedan probabilistički algoritam, koji samo sa određenom tačnošću može reći da li je broj prost ili ne. Ovakvi algoritmi su veoma važni u oblasti kao što je kriptografija, gde se u cilju dobijanja na vremenu isplati rizikovati tačnost informacije.

U 7. poglavlju opisuju se algoritmi za faktotizaciju. Izvršena je paralelizacija poznatog "trial division" algoritma. Posebno je zanimljiva paralelizovana verzija Fermat-ovog metoda za faktorizaciju, jer je do nedavno ovaj metod bilo bezuspešno primeniti na običnim računarima - drugi algoritmi su tada bili mnogo bolji.

Za navedene algoritme data je i procena njihovog vremena izvršavanja. Neki od njih su realizovani na transpjuterskoj ploči sa četiri transpjutera, i pokazali veoma dobre rezultate.

Na kraju su izloženi rezultati dobijeni pri ispitivanju Kurepine hipoteze o levom faktorijelu kao i hipoteze o alternirajućem faktorijelu. Ove dve hipoteze su testirane na transpjuterskoj ploči i nije pronađen kontraprimer u prvih 1000000 brojeva. Ovim je povećana gornja granica jer je ranije pokazana tačnost Kurepine hipoteze o levom faktorijelu za prvih 300000 (Mijajlović) odnosno 45000 brojeva (Wagstaf) , i za prvih 45000 brojeva u slučaju hipoteze o alternirajućem levom faktorijelu.

Kao dodatak su priloženi listinzi programa koji su pisani na jeziku 3L Parallel FORTRAN za transpjutere.

Teorija brojeva

Sasvim je sigurno da ne postoji oblast koja je više uzburkavala duhove među matematičarima. Mnogi veliki matematičari su veliku slavu stekli baš zahvaljujući teoriji brojeva, ali ima i veliki broj onih koji su davali "dokaze" netačnih tvrđenja ili čvrsto verovali u tačnost nekih hipoteza koje su kasnije opovrgnute. Takođe treba naglasiti da se među velikim brojem problema koji još uvek nisu rešeni nalazi dosta problema iz teorije brojeva.

Fermat je izrekao hipotezu da je broj $F_n=2^{2^n}+1$ prost za sve prirodne brojeve n i u to vreme je došlo do razilaženja matematičara po pitanju tačnosti ove hipoteze. Problem je rešio Euler pokazavši da je

$$F_5=2^{2^5}+1=641\cdot 6700417.$$

Primećujemo da je kontraprimer nađen već za n=5, i danas bi se ova jednakost mogla proveriti na običnom džepnom računaru. Današnji najmoćniji kompjuteri su izračunali da je F_{14} složen broj (a sastoji se od 4933 cifre), dok je F_{20} najmanji Fermat-ov broj za koji se ne zna da li je prost ili složen.

Fermat-ov savremenik Mersenne je ustanovio da od 55 prostih brojeva $p\leq 257$, 2^p-1 je prost samo za 11 vrednosti $p=2,3,5,7,11,13,17,19,31,67,127,257$. Verovatno je Mersenne pomoću dobijenog rezultata pokušao postaviti hipotezu o tome kada je broj 2^p-1 prost. Njegov pokušaj je već na početku bio osuđen na neuspeh jer se kasnije utvrdilo da je u prethodnom iskazu napravio pet grešaka. Iz liste su ispušteni brojevi 61,89 i 107 dok su pogrešno uneti brojevi 67 i 257. Suvišno je i govoriti koliku bi korist Mersenne imao od današnjih računara pomoću kojih je danas pronađeno 29 prostih brojeva navedenih osobina, među kojima je najveći 132049. Uz pomoć računara Cray 2 su Bateman, Selfridge i Wagstaff postavili novu Mersenne-inu hipotezu:

Ukoliko su dva od sledeća tri iskaza o nekom neparnom prirodnom broju p tačni, tada je i treći iskaz takođe tačan:

a) $p=2^k+1$, $p=2^k-1$, $p=4^k+3$ ili $p=4^k-3$
b) 2^p-1 je prost
c) $(2^p+1)/3$ je prost

Kakvu će ulogu ova hipoteza odigrati u teoriji brojeva i da li će se pokazati kao tačna u mnogome zavisi i od razvoja računarstva u doglednoj budućnosti.

Kriptografija

Kriptografija je matematička disciplina koja je nastala početkom veka ali je potpuni procvat doživela tek pre petnaest godina. Postoji mišljenje da je pogrešno šifrovanje poruka u mnogome uticalo na tok drugog svetskog rata. Teorija brojeva priskače u pomoć kriptografiji 1976. godine kada Rivest, Shamir i Adleman pronalaze veoma siguran način za šifrovanje poruka. Ovaj algoritam se zasniva na činjenici da je problem faktorizacije broja za sada nemoguće rešiti u vremenu koje je linearno zavisnom od broja cifara datog broja. Zato je danas glavni pravac u kome se razvija kriptografija konstruisanje algoritama za brz rad sa brojevima koji imaju veliki broj cifara.

Navedeni primeri su dovoljni pokazatelji potrebe razvoja i primene računarstava u teoriji brojeva. Razvoj se ogleda u traženju što bržih algoritama. Tako se dolazi do susreta sa paralelnim računarima koji predstavljaju poboljšanje današnjih računara u cilju povećanja brzine. Danas je paralelno programiranje jedna od najpopularnijih oblasti računarstva.

Zahvaljujem se mr Milanu Dražiću na programskim paketima za višecifarsku aritmetiku pisanim u FORTRAN-u. Takođe bih želeo izraziti veliku zahvalnost doc. dr Dušanu Tošiću koji me je upoznao sa paralelnim programiranjem i koji mi je davao veliku podršku kroz kurseve i seminare iz ove oblasti.

Posebnu zahvalnost dugujem svom mentoru, prof. dr Žarku Mijajloviću, koji je tokom ovih godina uvek bio spreman da mi priskoči u pomoć, posavetuje, uputi na prave stvari, i koji me je uveo u tajne istraživačkog rada.

1. PARALELNO PROGRAMIRANJE

Oblast koja je nastala ne tako davno, danas je glavni objekat mnogih proučavanja. Veliki broj praktičnih problema, nekada napuštenih kao nerešivi, danas jedinu pomoć mogu očekivati tek daljim razvojem paralelnog računarstva. Vremenska prognoza, sigurnost nuklearnih reaktora, veštačka inteligencija, računarska grafika, numerička izračunavanja, kriptografija i mnoge druge oblasti, kao glavno oružje koriste Cray, Connection Machine, ili neku drugu računarsku arhitekturu sličnih osobina.

O poreklu paralelizma

Osnovna uloga računara od njegovog nastanka je obavljanje poslova veoma velikom brzinom (čuvanje informacija je danas ipak u drugom planu). Stoga se i razvoj računarstva u najvećoj meri zasniva na usavršavanju hardverskih i softverskih dostignuća u cilju povećavanja brzine rada. Poboljšanja brzine usavršavanjem hardverskih elemenata su izvedena u velikoj meri, pa neki veći napredak u tom pravcu nije moguće postići. Na usavršavanju algoritama za rešavanje raznih problema takođe je mnogo urađeno i još se uvek mnogo radi. Međutim, za veliki broj problema nađeni su algoritmi koje nije moguće dalje poboljšavati, pa se ni u tom pravcu ne može očekivati veći napredak. Tako se dolazi do potrebe uvođenja nekih drugih metoda koje će povećati brzinu rešenja problema. Podeliti dati problem na više manjih problema, angažovati zatim određen broj procesora tako da svaki od njih rešava po jedan od tih problema, i na kraju objediniti sva tako dobijena rešenja u cilju dobijanja konačnog rešenja predstavlja

osnovnu ideju paralelnog programiranja..

Klasifikacija paralelnih računara

Postoji više klasifikacija današnjih paralelnih računara, u zavisnosti od osobina koje se posmatraju. U cilju lakšeg razmatranja, navodimo klasifikacije opisane u [24].

a) Tip i broj procesora

Pri ovoj podeli razmatramo "masivne paralelne sisteme" - računare u čiju arhitekturu mogu ući i više hiljada procesora. Nasuprot njima su "krupnozrni" računari koji sadrže relativno mali broj procesora.

b) Flyn - ova podela u zavisnosti od postojanja globalnog mehanizma

U zavisnosti od toka operacija koje izvršavaju procesori, kao i u zavisnosti od toka podataka koji se obrađuju, razlikujemo:

SISD (Single Instruction Single Data) model. Današnji jednoprocesorski računari spadaju u ovu grupu. U svakom trenutku se nad jednim određenim podatkom izvršava jedna određena operacija.

SIMD (Single Instruction Multiple Data) model. Svaki procesor u datom trenutku izvršava jednu istu naredbu, ali nad posebno određenim podatkom. U ovu kategoriju spadaju procesorki nizovi. Rad ovih računara se jasno vidi na primeru sabiranja vektora: svi procesori obavljaju istu operaciju (sabiranje dva broja), ali nad različitim podacima (odgovarajuće koordinate vektora).

MISD (Multiple Instrunction Single Data) model. Ovaj model

Flyn navodi samo u cilju postizanja kompletne teoretske podele, u praksi nešto slično se ne pojavljuje. S obzirom da Flyn-ova definicija podele nije sasvim precizna, u literaturi se mogu naći podele koje neke računare svrstavaju u ovu grupu.

· MIMD (Multiple Instruction Multiple Data) model. Svaki procesor u odgovarajućem trenutku izvršava različitu naredbu. U ovakvim situacijama procesori razmenjuju relativno mali broj podataka koje su na početku dobili. Očigledno da ovaj model ima bolje karakteristike od SIMD modela, jer MIMD računar može obavljati iste poslove kao SIMD računar.

c) Sinhroni i asinhroni modeli

Ukoliko se rad procesora odvija u diskretnim vremenskim trenucima $t_0, t_1, \ldots$ (da budemo precizniji, to su vremenski intervali) tako da svaki procesor u odgovarajućem vremenskom trenutku izvrši određen broj osnovnih operacija, kažemo da je rad procesora sinhronizovan. Na ovaj način se lakše obavlja komunikacija između procesora, jer je gubljenje vremena usled međusobne komunikacije svedeno na minimum. Na drugoj strani su MIMD modeli koji svoj rad uglavnom obavljaju asinhrono.

d) Klasifikacija u zavisnosti od načina korišćenja memorije od strane procesora

Način na koji procesori razmenjuju određene informacije dovodi do jedne od glavnih podela. Postoji mogućnost je da svaki procesor ima samo svoju, lokalnu memoriju, koju koristi za rad. U tom slučaju se procesori moraju povezati kanalima (direktno ili indirektno) kako bi mogli razmenjivati informacije. Međutim,

ukoliko procesori pored lokalne (opciono) imaju i jednu zajedničku, globalnu memoriju, tako da joj može pristupiti svaki procesor, i preko koje se vrši komunikacija, dobijamo model sa zajedničkom memorijom (shared memory model). Ova podela je od velike važnosti, pa ćemo je kasnije detaljnije razmotriti.

Pored ovih, pojavljuju se i mnoge druge podele. Međutim, treba imati u vidu da je današnje računare veoma teško pri nekoj podeli svrstati u određenu grupu, jer predstavljaju kombinaciju dve ili više grupa.

Računari sa razdeljenom memorijom

U [3] je detaljno razrađen pojam RAM (random access machine) moela, a mi ćemo navesti samo osnovne podatke. RAM model se sastoji iz memorije, ulazne trake, izlazne trake i programa koji se ne nalazi u memoriji i ne može se modifikovati. Glava za čitanje se pomera po poljima ulazne trake (u kojima se nalaze celi brojevi), rezultat rada programa se nalazi u poljima izlazne trake (koja takođe sadrži cele brojeve), a za vreme rada koristimo memoriju koja se sastoji iz registara u koje možemo upisivati i iz kojih možemo čitati celobrojne vrednosti. Instrukcije za pisanje programa su osnovne instrukcije koje imaju današnji računari: učitavanje, upisivanje, sabiranje, oduzimanje, ispitivanje znaka, bezuslovni skok...

Uvođenjem u igru dodatnih instrukcija dobijamo PRAM (Parallel Random Access Machine) model sa razdeljenom memorijom. To je model koji se sastoji od više RAM procesora čiji je skup instrukcija pojačan nekim dodatnim instrukcijama koje omogućuju

realizaciju paralelizma. Svaki procesor ima svoju, lokalnu memoriju, ali je takođe svakom od njih omogućen i pristup zajedničkoj, globalnih memoriji. Preko globalne memorije procesori mogu vršiti komunikaciju. Pošto procesori rade simultano, pojavljuje se mogućnost njihovog međusobnog sukobljavanja pri pokušaju upisivanja ili čitanja iz iste memorijske lokacije. U zavisnosti od toga na koji je način rešen ovaj problem, javljaju se različite vrste PRAM računara sa zajedničkom memorijo:

1) EREW (Exclusive-Read, Exclusive-Write) model. Nikoja dva procesora ne mogu istovremeno da čitaju sadržaj iste memorijske lokacije, niti da istovremeno vrše upisivanje u istu memorijsku lokaciju. Ovo je model sa najvećim ograničenjima, tako da svaki algoritam koji je projektovan za EREW-model može biti izveden i na ostalim modelima.

2) CREW (Concurrent-Read, Exclusive-Write) model. Procesori ne mogu istovremeno vršiti upisivanje u istu memorijsku lokaciju, ali im je omogućeno istovremeno čitanje iz iste memorijske lokacije. Predstavlja prelaz od najslabijeg ka najjačem modelu.

3) CRCW (Concurrent-Read, Concurrent-Write) model. Procesorima je omogućeno da istovremeno čitaju, ali i da istovremeno upisuju neki sadržaj u istu memorijsku lokaciju. Pri tome mora postojati pravilo po kome se vrši upis ukoliko dva procesora treba da upišu različite vrednosti u isti registar: prioritet ima procesor sa manjim rednim brojem, procesor koji želi da upiše veću vrednost ili se u registar upisuje suma vrednosti koje trebaju biti upisane.

Računari sa međusobno povezanim procesorima

Lakše ostvarljiv, ali manje moćan način za komunikaciju procesora jeste mreža međusobno povezanih procesora. Svaki procesor ima svoju lokalnu memoriju, a komunikaciju sa ostalim procesorima obavlja preko jednosmernih ili dvosmernih kanala. U zavisnosti od topološke konfiguracije nastale povezivanjem nekih procesora, dobijamo različite arhitekture.

1) Potpuno povezan skup procesora. Ovo je naravno najmoćnija arhitektura, jer je svaki procesor povezan sa svim ostalim procesorima, pa je i vreme komunikacije između bilo koja dva procesora minimalno. Problem je što je kod velikog broja procesora nemoguće praktično ostvariti toliki broj veza.

2) Linearni procesorski niz. Procesori (kojih ima k) se označe celim brojevima od 0 do k-1, a zatim se svaki poveže sa svojim prethodnikom (osim prvog) i sledbenikom (osim poslednjeg). Ova arhitektura se često razmatra u praksi jer je najjednostavnija i algoritam napravljen za procesorski niz može se ostvariti i na svim drugim konfiguracijama.

3) Dvodimenzionalni procesorski niz. Procesori se nalaze u čvorovima kvadratne matrice i svaki (osim ivičnih) je povezan sa svoja četiri suseda. Dimenzije matrice su mxm (pri čemu imamo $n=m^2$ procesora), i svakom procesoru je pridružen uređen par (i,j) gde je $i,j \in \{0,\ldots,m-1\}$. Ova arhitektura je veoma popularna, i mnogi današnji računari imaju ovu konfiguraciju.

4) Hiper-kocka. Ubedljivo najpopularnija arhitektura. Datih $n=2^k$ procesora obeležavamo brojevima od 0 do n-1 ali u binarnom zapisu, a zatim povezujemo one procesore čije je Hemingovo rastojanje jednako 1. Na ovaj način svaki procesor je povezan sa k drugih procesora, i veoma lako i brzo može stupiti u komunikaciju sa bilo kojim drugim procesorom.

Transpjuteri

Jedan od najpoznatijih modela paralelnih računara je transpjuterska ploča sa međusobno povezanim transpjuterima. Svaki transpjuter ima 4 ulazno-izlazna kanala koji se mogu povezati sa ulazno izlaznim kanalima drugih transpjutera. Uglavnom se ova povezivanja izvršavaju tako da se dobije dvodimenzionalni procesorski niz. Veliku popularnost transpjuteri su postigli jer se ugrađuju u PC-računare koji su danas veoma rasprostranjeni. Naš PC-procesor se povezuje sa jednim od transpjutera i reguliše operacije ulaza i izlaza.

Bitna osobina procesora je mogućnost simuliranja rada drugih modela paralelnih računara. Naime, mi možemo kreirati procese koji se dodeljuju transpjuterima. Pri tome se više procesa koji su dodeljeni istom transpjuteru izvršavaju paralelno (naravno, sa našeg nivoa apstrakcije). Između procesa koji su na istom transpjuteru može se definisati jedan ili više kanala (ovo su logičke a ne fizičke veze) preko kojih se obavlja komunikacija. Takođe, ukoliko dva procesa pripadaju procesorima koji su međusobno povezani fizičkim kanalom, između njih možemo uspostaviti kanal za komunikaciju.

Implementacija algoritama na transpjuterskoj ploči se vrši

a uz to znajući da važi

3) NZD(a,0)=a

dobijamo ideju za algoritam. Naime, pošto je

$$\frac{a-b}{4} \leq \frac{a+b}{4} \leq a$$

to je onda očigledno da konačnom primenom pravila 1) ili 2) dolazimo do situacije kada se primenom pravila 3) dolazi do kraja rada algoritma.

Algoritam NZD1:

```
{ Ulaz: brojevi a,b za koje važi: a je neparan, a≤2^n, 0<b≤2^n }
{ Izlaz: a=NZD(a,b) }

01  δ=0
02  REPEAT
03        WHILE bmod2=0 DO
04              b=b/2
05              δ=δ+1
06        IF δ>0 THEN
07                c=a
08                a=b
09                b=c
10                δ=-δ
11        IF (a+b)mod4=0 THEN
12                b=(a+b)/4
13        ELSE  b=(a-b)/4
14  UNTIL b=0
```

Za razliku od Euklidovog algoritma koji u svakoj iteraciji mora da primeni funkciju min(a,b) (pa je potrebno u svakom trenutku znati vrednosti a i b), algoritam NZD1 ispituje samo dva poslednja bita od a i b (provera deljivosti sa 4), a operacije koje se primenjuju su oduzimanje, množenje i deljenje sa 2 (aritmetičko šiftovanje za 1 bit u levo). Nije teško pokazati da se tokom rada algoritma izvrši $2n=O(n)$ ciklusa REPEAT naredbe. Znajući za tu činjenicu, posmatranjem IF-naredbi datog algoritma, zaključujemo da je svaki ciklus REPEAT-naredbe jednoznačno određen vrednostima dva poslednja bita promenljivih a i b, kao i celobrojne promenljive δ. Uopšte, k+1 uzastopnih ciklusa REPEAT-naredbe je determinisano vrednošću promenljive δ, i k+1-nim poslednjim bitom promenljivih a i b. Preciznije, nama je potreban samo znak od δ, i njena vrednost u slučaju $|\delta|\leq k$. Sada se izvršavanjem k ciklusa REPEAT-naredbe (u daljem tekstu k-transformacija) izvršava

$a=2^{-k}(ac+be)$

$b=2^{-k}(ad+bf)$

$\delta=\sigma\cdot\delta+g$,

gde su c,d,e i f celi brojevi iz intervala $[-2^k,2^k]$, $-k\leq g\leq k$ i $\sigma\in\{-1,1\}$. Stoga ćemo prvo izračunati tablicu $2^{k+1}\times 2^{k+1}\times(4k+4)$ pomoću koje ćemo na osnovu trenutnih vrednosti a, b i δ nalaziti konstante potrebne za primenu k-transformacije. Ako sa a' i b' obeležimo k+1 poslednjih bitova od a i b, a sa δ' promenljivu čiji prvi bit kazuje da li je $\delta>0$, drugi da li $\delta\in[-k,k]$ a ostali vrednost za δ u tom slučaju (imamo ukupno 4k+4 vrednosti za promenljivu δ'), tada je $(c,d,e,f,\sigma,g)=T(a',b',\delta')$. Sada se paralelni algoritam izvršava sukcesivnom primenom k-transformacija.

Algoritam NZD2:

{Ulaz: prirodni brojevi a i b }

{Izlaz: prirodni broj a=NZD(a,b) }

{Arhitektura: procesori sa zajedničkom CRCW memorijom }

01 REPEAT

02 $a'=a \bmod 2^{k+1}$

03 $b'=b \bmod 2^{k+1}$

04 δ'=bitovi za znak, interval i logk zadnjih bitova od δ

05 $(c,d,e,f,\alpha,g)=T[a',b',\delta']$

06 $a=(ac+be)/2^k$

07 $b=(ad+ef)/2^k$

08 $\delta=\alpha\delta+g$

09 UNTIL b=0

Kako ovaj algoritam izvršava k uzastopnih ciklusa algoritma NZD1, a prema ranije rečenom algoritam NZD1 ima najviše 2n ciklusa, to se u algoritmu NZD2 izvršava najviše 2n/k ciklusa. Ostaje još da pokažemo kako se svaki ciklus može izvršiti u konstantnom vremenu, čime će vreme rada algoritma iznositi $O(n/k)$:

1) Naredbe (02-04) su k-bitne naredbe dodeljivanja pa je vreme njihovog izvršavanja konstantno.

2) Tablica T ima $(4k+4)\cdot 2^{2k+2}$ elemenata, od kojih svaki ima ne više od 5k bitova. Koristeći činjenicu da L-bitnom elementu iz tablice od S elemenata možemo pristupiti u konstantnom vremenu pomoću $S\cdot\max(L,\log S)$ procesora sa zajedničkom CRCW memorijom, dobijamo da je u ovom slučaju potrebno $5k(4k+4)2^{2k+2}<k^2\cdot 2^{2k+7}$

procesora. Algoritam se izvršava tako što logS procesora određuje adresu traženog elementa, a L procesora je potrebno da donese L bitova. Stoga se i naredba (05) izvršava u konstantnom vremenu.

3) Ostaje još da se ispitaju naredbe (06-08), gde se vrši množenje, sabiranje i deljenje višebitnih brojeva. Deljenje sa 2^k se izvršava u konstantnom vremenu jer je to u stvari aritmetičko šiftovanje u levo za k bitova. Sabiranje n+k bitnih brojeva se može izvršiti u konstantnim vremenu pomoću $n\cdot 2^k$ procesora (Chandra, 1983). Množenje n-bitnog i k-bitnog broja se vrši u konstantnom vremenu tako što se broj dužine n podeli na n/k blokova dužine k, pa uz pomoć tablice za množenje k-bitnih brojeva svaka grupa procesora obavi po jedno množenje. Za ovaj posao potrebno je ukupno $2n\cdot 2^{2k}$ procesora.

Sada zaključujemo da se algoritam NZD2 može izvršiti u vremenu $O(n/k)$ uz pomoć $\max(2n^{2k}, k^2 2^{2k+7})$ procesora. Da bi ovaj broj procesora bio polinomijalan k mora biti $O(\log n)$. U tom slučaju je $T_2(n)=O(n/\log n)$. Pripremni algoritam - konstrukcija tabele, može se izvršiti pomoću $O(k^3 2^{3k+\sigma})$ procesora u vremenu $O(\log n)$ pa ovo vreme ne utiče značajno na izvršavanje glavnog algoritma.

Napomenimo još da su autori napravili i drugu verziju ovog algoritma u cilju prilagođavanju procesorima sa CREW memorijom. U tom slučaju vreme izvršavanja je $O((n/k)\log k+\log^2 k)$ uz pomoć $O(k^3 2^{3k}+n2^{2k})$ jednobitnih procesora. Naravno, pošto današnji procesori imaju mogućnost rada sa registrima od po 64 bita, to je dovoljno imati 2^σ puta manje procesora, ali time ne dobijamo nikakvo asimptotski zadovoljavajuće poboljšanje.

Neki drugi pravci u razvoju NZD algoritma

Pored izloženog algoritma postoje i drugi algoritmi za NZD ali sa malo većim vremenom izvršavanja. Pokušaj paralelizacije Euklidovog algoritma za NZD na isti način kao što je urađeno u prethodnom algoritmu - "pakovanjem" uzastopnih iteracija u jednu transformaciju- uradili su Kannan, Miller i Rudolph. Dobijen je paralelni algoritam koji radi u vremenu $O(n \log\log n / \log n)$ uz pomoć $O(n^2 \log^2 n)$ procesora sa zajedničkom CRCW memorijom. To je bio prvi sublinearan algoritam za NZD.

Ukupno gledano, možemo reći da je na rešenju ovog problema tek napravljen početni korak. Dati algoritmi se za sada ne mogu primeniti u praksi (navedene arhitekture se pojavljuju samo pri teoretskim razmatranjima). Čak i ako jednom ove arhitekture budu ostvarene u praksi, paralelni algoritmi na njima neće biti mnogo ekonomični u odnosu na sekvencijalne algoritme, jer je za njihov rad potreban veliki broj procesora.

3. RAČUNANJE FUNKCIJE PI(X)

Funkcija $\pi(x)$ kao vrednost daje broj prostih brojeva ne većih od x. Već na početku možemo uočiti vezu između problema nalaženja vrednosti $\pi(x)$ i nekih drugih problema koje ćemo kasnije razmatrati. Naime, ako znamo sve proste brojeve manje od x, onda znamo i vrednost $\pi(x)$, pa možemo reći da je problem koji rešavamo ne veće složenosti od problema nalaženja svih prostih brojeva manjih od izvesne granice. Sa druge strane, ako sa P obeležimo skup prostih brojeva, onda se karakteristična funkcija χ_P datog skupa može definisati kao:

$\chi_P : \mathbb{N} \to \{0,1\}$,

$\chi_P(1)=0$,

$\chi_P(n)=\pi(n)-\pi(n-1)$ za $n>1$.

To u stvari znači da je problem ispitivanja primalnosti broja ne manje složenosti od problema nalaženja vrednosti $\pi(x)$. Nama je naravno cilj da nađemo algoritam koji je efikasniji od algoritma za rešavanje problema na koji se dati problem svodi, a navedene činjenice nam služe kao orjentacija.

Neke važne formule za računanje $\pi(x)$

Prvi značajniji rad na ovom polju dao je Legendre. Neka je $p_1<p_2<\ldots$niz svih prostih brojeva. Tada je

$$\pi(x)=\pi(x^{1/2})-1+[x]-\sum [x/p_i] + \sum [x/p_i p_j]-\sum [x/p_i p_j p_k]+\ldots$$

Ovu formulu nije teško izvesti ako pođemo od činjenice da se skup prirodnih brojeva ne većih od x sastoji od svih prostih brojeva ne većih od x, svih složenih brojeva ne većih od x i jedinice. Datu formulu Legendre je primenio 1830. godine za računanje vrednosti $\pi(10^6)$ metodom "papira i olovke". S obzirom da se sve to dešava više od 100 godina pre pojavljivanja računara, greška napravljena u toku rada (dobijen je rezultat za 28 veći od ispravnog) može se smatrati zanemarljivom.

Nemački astronom E.D.F. Meissel je četrdeset godina kasnije generalizovao Legendre-ovu formulu, čime je dobio veoma značajne rezultate koji su osnova danas najbržih algoritama za nalaženje vrednosti $\pi(x)$. Neka je p_i i-ti prost broj, i neka je a proizvoljan prirodan broj. Neka je dalje $P_k(x,a)$ skup svih prirodnih brojeva ne većih od x koji se mogu predstaviti kao proizvod k prostih brojeva čiji su indeksi veći od a. Tada se skup prirodnih brojeva manjih od x sastoji od :

1) broja 1

2) prvih a prostih brojeva $p_1, p_2, \ldots, p_a$

3) brojeva kojima je bar jedan prost faktor manji od p_a i kojih ima :

$$\sum [x/p_i] - \sum_{i<j} [x/p_i p_j] + \sum_{i<j<k} [x/p_i p_j p_k] \ldots$$

4) svih prostih brojeva p takvih da je $p_a < p \leq x$, ima ih:

$P_1(x,a) = \pi(x) - a$

5) $P_2(x,a)$ prirodnih brojeva $n = p_i p_j \leq x$ takvih da je $a < i \leq j$

6) $P_3(x,a)$ prirodnih brojeva $n = p_i p_j p_k \leq x$ takvih da je $a < i \leq j \leq k$

7) $P_s(x,a)$ prirodnih brojeva n konstruisanih analogno sa 5) i 6)

Pošto je suma svih pomenutih brojeva jednaka [x] a iz 5) vidimo da među sabircima učestvuje $\pi(x)$, to je očigledno $\pi(x)$ moguće izračunati ako se znaju vrednosti sabiraka pod 5), 6) i 7). Uglavnom se a bira tako da bude $p_{a+1} > x^{1/r}$ pa je onda $P_s(x,a)=0$ za $s>r$. Kada je $r=3$ dobijamo Meissel-ovu formulu, za $r=4$ dobijamo Lehmer-ovu formulu, dok je za $r>4$ isuviše komplikovano naći $P_s(x,a)$ pa se ovaj slu;aj ne primenjuje u praksi. Formule za P_2 i P_3 su:

$$P_2(x,a)=-(b-a)(b-a+1)/2+\sum_{i=a+1}^{b}\pi(x/p_i)\ ,\ \text{gde je } b=\pi(x^{1/2})$$

$$P_3(x,a)=\sum_{i=a+1}^{c}\ \sum_{i}^{bi}\{\pi(x/p_ip_j)-(j-1)\},\ \text{gde je } c=\pi(x^{1/3}),\ b=\pi((x/p_i)^{1/2})$$

Označimo još broj svih prirodnih brojeva ne većih od x koji među činiocima nemaju prostih brojeva ne većih od p_a kao $\phi(x,a)$ (suma iz 3, nazivamo je još i Legendre-ova suma). Dakle

$\phi(x,a)=[x]-\sum[x/p_i]+\sum[x/p_ip_j]-\sum[x/p_ip_jp_k]+\ldots$

Kada je $p_a \le x^{1/2} < p_{a+1}$ dobijamo na početku navedenu Lagrange-ovu formulu. Iz definicije se lako zaključuje da važi rekurentna veza $\phi(x,a)=\phi(x,a-1)-\phi(x/p_a,a-1)$ što je najvažnija relacija u svim algoritmima namenjenim za izračunavanje $\pi(x)$.

Sekvencijalni algoritmi za računanje $\pi(x)$

Zavisno od vrednosti koje uzima a (preciznije p_a) u odnosu na x, dobijamo i različite varijante algoritama.

$p_a \le x^{1/2} < p_{a+1}$:

U ovom slučaju je $P_2(x,a)=P_3(x,a)=\ldots=0$ i ceo posao se sastoji u računanju vrednosti $\phi(x,a)$. Ovaj slučaj razmatrao je

David C. Mapes 1963. godine razvijanjem ranije navedene rekurentne formule, a kada a postane dovoljno malo primenjuje se Lehmer-ova formula. Vreme rada algoritma je $O(x^{0,7})$, a njegovom primenom je izračunata vrednost $\pi(10^9)$.

$p_a \leq x^{1/3} < p_{a+1}$:

Sada je $P_3(x,a)=P_4(x,a)=...=0$, pa treba izračunati $P_2(x,a)$, ali je zato posao pri izračunavanju $\phi(x,a)$ manji, jer je a manje nego u prethodnom primeru. Algoritam je razmatrao Meissel i našao $\pi(10^9)$, doduše uz malu grešku jer je računanje izvršeno 1885. godine, naravno bez primene računara.

$p_a \leq x^{1/4} < p_{a+1}$:

U ovom slučaju je $P_4(x,a)=P_5(x,a)=...=0$, vreme potrebno za računanje $\phi(x,a)$ je smanjeno ubacivanjem u igru vrednosti P_2 i P_3. Primenom ovog algoritma Maissel je našao tačnu vrednost za $\pi(10^9)$ ali tek 1958. godine.

Zajedničko za sve ove algoritme je zahtev za velikim memorijskim prostorom. Naime, svaki od algoritama zahteva poznavanje vrednosti funkcije π za veliki broj argumenata kako bi se računanje izvelo što efikasnije. Samim tim je potrebno izvršiti i velike pripreme pre početka izvršavanja glavnog dela algoritma. Što se tiče funkcije ϕ, u sva tri algoritma se mora računati primenom rekurentne formule, čime se dobija binarno drvo izračunavanja. Da bi se zaustavilo grananje, u jednom trenutku se primenjuje pravilo za prestanak dalje primene rekurentne formule. Pri tome se u posebno napravljenoj tablici mora čuvati vrednost funkcije u tačkama u kojima je primenjeno pravilo za prestanak rada. Svi navedeni algoritmi stoga se dele na više različitih

algoritama u zavisnosti od toga koje se pravilo prestanka grananja primenjuje.

Paralelni algoritam za računanje vrednosti $\pi(x)$

Opisaćemo 'Prošireni Meissel-Lehmer-ov algoritam' (Extended Meissel-Lehmer Algorithm) koji su 1985. godine objavili Lagarias, Miller i Odlyzko. Ovaj algoritam danas predstavlja najbrži sekvencijalni algoritam za računanje $\pi(x)$, a pošto je moguće izvršiti i idealnu paralelizaciju navedenog algoritma, onda je to sigurno i najbolji paralelni algoritam koji rešava dati problem.

Neka je dat realan pozitivan broj x, i neka je M broj procesora koji su nam na raspolaganju, pri čemu stavljamo ograničenje $1 \le M \le x^{1/3}$. Izaberimo $a \in \mathbb{N}$ tako da je $p_a \le x^{1/3} < p_{a+1}$. Vrednost $\pi(x)$ računamo primenom formule $\pi(x)=\phi(x,a)-P_2(x,a)+a-1$. Algoritam se izvršava u dve faze: prvo se računa vrednost $P_2(x,a)$, a zatim se nalazi $\phi(x,a)$.

Računanje vrednosti $P_2(x,a)$: ako pogledamo formulu

$$P_2(x,a)=-(b-a)(b-a+1)/2 + \sum_{i=a+1}^{b} \pi(x/p_i) \quad , \text{ gde je } b=\pi(x^{1/2})$$

videćemo da treba zaposliti procesore na izračunavanju sume

$$\sum_{i=a+1}^{b} \pi(x/p_i)=\sum \pi(x/p) \quad , \text{ za } \quad x^{1/3} < p \le x^{1/2}$$

Pošto je $x/p < x^{2/3}$ kada je $p > x^{1/3}$, sumu možemo izračunati tako što ćemo naći sve proste brojeve iz intervala $[1, x^{2/3}]$ - svaki od M procesora dobija podinterval iste dužine koji treba da obradi

omogućeno da lako izračunaju vrednosti funkcije π na tom intervalu. Zatim se svaki procesor angažuje na računanju sume onih sabiraka $\pi(x/p)$ za koje x/p pripada njegovom intervalu. Na kraju se sabiraju sume dobijene u svim procesorima i dobijeni rezultat prosleđuje glavnom procesoru koji konačno računa $P_2(x,a)$. Ceo posao se obavlja u vremenu $O(M^{-1}x^{2/3+\varepsilon})$ pri čemu se rezerviše memorijksi prostor $O(x^{1/3+\varepsilon})$ za svaki procesor.

Računanje $\phi(x,a)$: primenom rekurzivne formule koju smo ranije naveli, dobijamo binarno drvo u čijim su čvorovima vrednosti $\phi(x/n,b)$ gde je n proizvod ne više od a-b prostih brojeva. Za završne čvorove drveta uzimamo one kod kojih je

1) $b=0$ i $n\leq x^{1/3}$ ili

2) $n>x^{1/3}$

pri čemu čvorove tipa 1) zovemo ordinalni a čvorove tipa 2) specijalni. Kao i u prethodnoj fazi, interval $[1,x^{2/3}]$ se podeli na podintervale koje obrađuju procesori. U toku obrade procesori nalaze sumu S_1 ordinalnih čvorova, kao i vrednosti $\phi(z_o,i)$ gde je $1\leq i\leq\pi(x^{1/3})$ a $z_o=[x^{1/3}]$. Zatim se računa suma S_2 specijalnih čvorova, i na kraju $\phi(x,a)=S_1+S_2$. Svaki procesor će utrošiti vreme $O(M^{-1}x^{2/3+\varepsilon})$ uz zauzeće $O(x^{1/3+\varepsilon})$ memorijskih lokacija.

Vidimo da se dati algoritam izvršava u vremenu $O(M^{-1}x^{1/3})$. Problem za njegovu praktičnu primenu je mogućnost realizacije samo na arhitekturama sa zajedničkom CREW memorijom. Sekvencijalna verzija ovog algoritma je upotrebljena za računanje vrednosti $\pi(x)$ gde je $x=4\cdot 10^{16}$. Program je na računaru IBM 3081 Model K radio 1730 minuta da bi sračunao do danas najveću poznatu vrednost funkcije $\pi(x)$.

Kako praktično izračunati $\pi(x)$

Videli smo da navedeni algoritam nije moguće primeniti na arhitekturama koje se danas koriste u praksi. Stoga se nameće potreba primene nekog drugog algoritma za računanje vrednosti $\pi(x)$. Kao što smo već rekli, sekvencijalna verzija navedenog algoritma je danas najbrži način za računanje vrednosti $\pi(x)$. Znajući za formulu $\phi(x,a)=\phi(x,a-1)-\phi(x/p_a,a-1)$, možemo u svakom trenutku $\phi(x,a)$ izraziti preko proizvoljno mnogo vrednosti funkcije ϕ u nekim manjim tačkama. Na primer, za n=5 je:

$$\phi(x,a)=\phi(x,a-3)-\phi(x/p_{a-3},a-3)-\phi(x/p_{a-2},a-2)-$$
$$-\phi(x/p_a,a-2)+\phi(x/p_ap_{a-1},a-2)$$

Konstrukcija formule je očigledna: pošto se svakom primenom rekurentne veze dobija jedan sabirak više na desnoj strani, to znači da formulu treba primeniti onoliko puta koliko sabiraka želimo imati na kraju. Svaki sabirak je oblika $\phi(x/n,a-k)$ gde je n proizvod s različitih prostih brojeva. Nama je cilj da se sume s+k kod svakog sabirka što manje razlikuju, pa stoga primenjujemo rekurentnu formulu na onom sabirku $\phi(x/n,a-k)$ kod koga je s+k najmanje.

Neka je sada dato x za koje se traži vrednost $\pi(x)$. Uzmimo a tako da je $p_a\leq x<p_{a+1}$ pa je sada $[x]=\phi(x,a)+P_0(x,a)+P_1(x,a)$, pri čemu je $P_0(x,a)=1$, $P_1(x,a)=\pi(x)-a$ odakle dobijamo konačno traženu formulu: $\pi(x)=[x]-\phi(x,a)+a-1$. Neka je broj procesora k. Tada vrednost $\phi(x,a)$ možemo izraziti preko k vrednosti funkcije ϕ u manjim tačkama. Dodelićemo zatim svakom procesoru po jedan par $(x/n,a-k)$ za koji treba da izračuna vrednost funkcije π. Zbog načina na koji smo birali date sabirke, procesori će utrošiti približno isto vreme na računanje vrednosti koje su im zadate.

Naravno, računanje se obavlja primenom drugog dela ranije navedenog algoritma. Na kraju, svi procesori šalju dobijene vrednosti glavnom procesoru koji zatim nalazi vrednost $\phi(x,a)$.

Dobra osobina ovog algoritma je što se može primeniti na proizvoljnoj arhitekturi procesora koji rade paralelno. Vreme utrošeno za nalaženje $\pi(x)$ je ipak veće od vremena koji troši prethodni algoritam zato što svi procesori posebo rade neke iste stvari (na primer tablicu za ϕ). Naravno, to je posledica osobine arhitekture na kojoj obavljamo rad, jer smo zabranili procesorima da imaju pristup istim delovima memorije.

4. ERATOSTENOVO SITO

Problem nalaženja svih prostih brojeva manjih od izvesne granice n rešen je odavno, i danas je uz Euklidov algoritam za nalaženje NZD verovatno najpoznatiji algoritam koji se primenjuje u teoriji brojeva. Izuzimajući varijante napravljene zbog prilagođavanja algoritma osobinama današnjih računara, ovaj algoritam od svoje prve verzije nije pretrpeo značajne promene.

Algoritam A1:

```
{ Ulaz : prirodan broj n }
{ Izlaz: skup P prostih brojeva ne većih od n }

01   S={x | 2≤x≤n}
02   P={}
03   p=min S
04   WHILE p^2≤n DO
05        P=P∪{p}
06        m=p
07        WHILE m≤n DO
08             S=S\{m}
09             m=m+p
10        p=min S
11   P=P∪S
```

Procenimo vreme $T_1(n)$ datog algoritma A1 u zavisnosti od ulazne veličine n. Prva WHILE petlja se izvršava za sve proste brojeve p takve da je $p^2 \leq n$, a za svako p druga WHILE petlja se

izvršava n/p puta tako da je $T_1(n)\sim\sum(n/p)=n\sum(1/p)$, gde je p prost i $p^2\leq n$. Da bismo izračunali gornju sumu iskoristićemo formulu (*Merten*) $\prod(1-1/p)=(e^{\gamma}\ln x)^{-1}$, gde je γ-Ojlerova konstanta, $p\leq x$. Logaritmovanjem obe strane date jednakosti dobijamo $\sum\ln(1-1/p)=-\gamma-\ln\ln x$ pa koristeći Tejlorov razvoj imamo $-\gamma-\ln\ln x=\sum\ln(1-1/p)=\sum\sum(-1)^{n-1}(-1/p)^n/n=\sum\sum-1/(np^n)=-\sum(n^{-1}\sum p^{-n})$ $=-\sum p^{-1}-\sum(n^{-1}\sum p^{-n})=-\sum p^{-1}-\sum\sum n^{-1}p^{-n}=-\sum p^{-1}-\sum\sum n^{-1}p^{-n}<-\sum p^{-1}-\sum\sum p^{-n}$ $=-\sum p^{-1}-\sum p^{-2}(1-1/p)^{-1}=-\sum p^{-1}-\sum(p^2-p)^{-1}$ odnosno:

$\sum p^{-1}=\ln\ln x-\sum(p^2-p)^{-1}-\gamma$ pa pošto $\sum(p^2-p)^{-1}$ konvergira po poredbenom kriterijumu $0\leq\sum(p^2-p)^{-1}\leq\sum n^{-2}$ to onda stavljajući da $x\to\infty$ u poslednjoj jednakosti dobijamo $\sum p^{-1}\sim\ln\ln x$ odnosno $T_1(n)\sim n\ln\ln n^{1/2}\sim n\ln\ln n$. Dalje je $\ln\ln n=\ln(\log n\cdot\ln 2)$ $=\log(\log n\cdot\ln 2)\ln 2=(\log\log n+\log\ln 2)\ln 2$ pa je kona;no $T_1(n)=O(n\log\log n)$. Ovde je učinjen prelazak sa prirodnog logaritma na binarni zbog ustaljenog obi;aja u teoriji složenosti algoritama. Pošto je pojam binarnog logaritma intuitivno jasan (mnogi jednostavni algoritmi rade u vremenu $O(\log n)$ - npr. binarno pretraživanje, stepenovanje, zapisivanje broja u sistemu sa drugom osnovom) to se i složeniji algoritmi tada lakše shvataju.

Paralelizacija algoritma 'Eratostenovo sito'

Sada ćemo razraditi paralelnu verziju datog algoritma. Problem rešavamo na PRAM sa razdeljenom memorijom. Kasnije ćemo razmatrati mogućnost primene algoritma na drugim konfiguracijama.

Posmatrajući algoritam A1 već na početku se nameće jedno rešenje - promenljiva p će predstavljati vektor, i svaki procesor će biti zadužen za jednu njegovu koordinatu. To znači da bi i-ti procesor uzimao i-ti minimalni elemenat skupa S i njime prosejao

skup S. Medjutim, tada bi dolazilo do slučajeva kada dva procesora moraju istovremeno koristiti iste lokacije memorije za čitanje i upisivanje, pa bi algoritam bio ostvarljiv ali samo na PRAM sa CRCW zajedničkom memorijom. Odbacujemo ovo rešenje jer je problem rešiv u asimptotski istom vremenu na mašinama sa jačim ograničenjima.

Problem ćemo rešiti deljenjem glavnog objekta - intervala brojeva od 1 do n na jednake disjunktne podintervale tako da svaki procesor obrađuje po jedan interval. Zato, neka raspolažemo sa k procesora , i ne remeteći opštost zadatka možemo pretpostaviti da je n=k. Uvedimo oznaku $I_{a,b}$ za zatvoren interval prirodnih brojeva, odnosno $I_{a,b}=\{n \mid n\in\mathbb{N}, a\leq n\leq b\}$ gde $a,b\in\mathbb{N}$. Tada je

$$I_{1,N}=\bigcup I_{im+1,(i+1)m},\ 0\leq i\leq k-1$$

Procesoru Pi dodeljujemo interval $I_{im+1,(i+1)m}$. Prosejavanje se ne može izvršiti dok nisu poznati prosti brojevi manji od $r=((i+1)m)^{1/2}$, pa će stoga svaki procesor prvo morati izvršiti algoritam A1 na intervalu $I_{1,r}$. Zatim se izvršava prosejavanje odgovarajućeg intervala prostim brojevima čiji kvadrat ne prelazi gornju grenicu tog intervala. Algoritam je najefikasniji ako se primeni na arhitekturi kod koje procesori imaju zajedničku memoriju. Ostvarljiva je i primena na drugim arhitekturama, ali se gubi određeno vreme pri slanju dobijenih rezultata glavnom procesoru.

Algoritam A2:

{Ulaz: prirodan broj n}

{Izlaz: skup P prostih brojeva ne većih od n}

{Arhitektura: procesori $P_0,\ldots,P_{k-1}$ sa zajedničkom EREW memorijom}

```
01    DO in parallel 0≤i≤k-1
02        Pi: r=((i+1)m)^1/2
03            Primeniti algoritam A1 za n=r, P=Pi
04            S={n|n∈N, im+1≤n≤(i+1)m}
05            WHILE S≠0 DO
06                p=minS
07                P=P∪{p}
08                s=⌈(im+1)/p⌉·p
09                WHILE s≤(i+1)m DO
10                    S=S\{s}
11                s=s+p
```

Procenimo sada vreme $T_2(n,k)$ potrebno za izvršenje algoritma A2 pomoću k procesora. Ako sa $T_{2,i}(n)$ obeležimo vreme rada i-tog procesora, onda je $T_2(n,k)=\max\{T_{2,i}(n)\}$, $0\le i<k$ i očigledno da P_{k-1} troši najviše vremena, a radi od početka do kraja algoritma A2 pa je $T_2(n,k)=T_{2,k-1}(n)$. Rad procesora P_{k-1} se sastoji u primeni algoritma A1 za ulaz n a zatim u izvršenju prve petlje (koja se izvršava $\pi(n^{1/2})$ puta) unutar koje se nalazi druga petlja (koja se izvršava m/p puta za svaki prost $p\le n^{1/2}$). Stoga, primenjujući usput formulu dobijenu pri računanju vremena

pa je $T_2(n)=O((n^{1/2}+m)\log\log n)=O((n^{1/2}+n/k)\log\log n)$ i sad možemo razmatrati dati izraz znajući da je $n^{1/2}+n/k$ najmanje kada je k maksimalno.

$k<n^{1/2}$:

$T_2(n)=O((n/k)\log\log n)$ pa je dati algoritam asimptotski idealna paralelizacija algoritma A1

$k=n^{1/2}$;

$T_2(n)=O(n^{1/2}\log\log n)$ i dobijamo najkraće vreme za koje se može izvršiti algoritam A2, a pri tome je i iskorišćenost procesora maksimalna

$k>n^{1/2}$:

$T_2(n)=O(n^{1/2}\log\log n)$, gubi se idealna iskorišćenost procesora. Problem se pojavljuje jer je interval namemjen za prosejavanje manji od intervala koji se obrađuje u pripremi - generisanju prostih brojeva koji služe za prosejavanje.

Očigledno je da navedena klasifikacija samo teoretski doprinosi razmatranju paralelizacije Eratostenovog sita. Za $n=10^{18}$ bilo bi potrebno čak milijardu procesora što je u današnjim uslovima neostvarljivo.

Možemo li brže izvršiti algoritam 'Eratostenovo sito' ?

Videli smo da se za $k\leq n^{1/2}$ postiže asimptotski gledano idealna paralelizacija, poboljšanja su moguća samo u pogledu smanjenja skrivene konstante. To se uglavnom postiže korišćenjem hardverskih osobina računara, dakle mašinski zavisnim poboljšanjima. Postavlja se pitanje koliko brzo možemo izvoditi navedeni algoritam u najboljim mogućim uslovima?

Neka imamo k procesora sa zajedničkom memorijom iz koje je dozvoljeno istovremeno čitati brojeve za više procesora. Vreme potrebno za izvršenje algoritma E2 je $T_2(n)=O((n^{1/2}+n/k)\log\log n)$, i vidimo da je vreme ograničeno zbog primene sekvencijalnog algoritma Eratostenovog sita u intervalu $[1,n^{1/2}]$ od strane poslednjeg po redu procesora (pripremna faza), dok za u istom trenutku ostali procesori traže iste podatke samo u manjem obimu. Stoga se nameće ideja da angažujemo procesore paralelno na istom poslu samo za slučaj intervala $[1..n^{1/2}]$. Kao što smo ranije dobili, $n^{1/4}$ procesora će primenom algoritma E2 traženi posao uraditi u vremenu $O(n^{1/4})$ pa ako uzmemo $k=n^{3/4}$ procesora dobijamo vreme novog algoritma E3: $T_3(n)=O(n^{1/4}\log\log n)$. Međutim, analogno prethodnom razmišljanju, možemo algoritam E3 primeniti na nalaženje prostih brojeva iz intervala $[1..n^{1/2}]$ pa ćemo dati posao uraditi za vreme $O(n^{1/8})$ i uzimanjem $k=n^{7/8}$ procesora za drugi deo posla dobijamo konačno vreme $T_4(n)=O(n^{1/8}\log\log n)$. Nastavljajući navedeni postupak dobijamo generalizovano tvrđenje: za dato n, uz pomoć $k=n^{1-2^{-r}}$ procesora, algoritam A2 se izvršava u vremenu $T(n,k)=O(n^{2^{-r}})=O(nk^{-1}\log\log n)$.

Efikasnost algoritma u praksi

Primena Eratostenovog sita predstavlja dovoljno dobro rešenje za problem generisanja prostih brojeva sa teoretske tačke gledišta. U prilog tome ide i činjenica da je danas u teoretskom računarstvu NP-kompletnost jedan od glavnih problema, pa se rešenje koje radi u linearnom ili skoro linearnom vremenu retko pokušava poboljšati. Međutim, u praksi je situacija potpuno drugačija. Najveće Eratostenovo sito je napravljeno za

$N=7{,}263\cdot 10^{13}$ (Jeff Young, Aaron Potler). Program je pisan kombinovano na jezicima FORTRAN i CAL (Cray Assembly Language), i izvršen na računaru CRAY-2. Ovo N je ipak isuviše malo da bi se uspešno rešili mnogi problemi teorije brojeva, pa je potrebno uvesti neka poboljšanja. Većina tih novina se zasniva na hardverskim osobinama računara na kojima izvodimo program, ali takva poboljšanja ne obećavaju značajnije pomeranje gornje granice. Poboljšanja zasnovana na konstrukciji bržih sekvencijalnih algoritama su takođe bez perspektive. Zbog toga je primena paralelnih računara u ovom slučaju veoma značajna jer predstavlja jedini pravac koji obećava značajan napredak u povećanju brzine.

Realizacija algoritma Eratostenovo sito na transpjuterima

Navedeni algoritam je realizovan na transpjuterskoj ploči sa četiri transpjutera T800. Program je pisan na jeziku 3L Parallel FORTRAN za transpjutere. U veoma kratkom vremenskom intervalu program je generisao bazu prostih brojeva manjih od 10^9. Na početku rada svaki transpjuter je generisao skup svih prostih brojeva p sa osobinom $p^z \leq 10^9$. Ceo interval na kome je trebalo vršiti prosejavanje podeljen je na blokove dužine 9600960 brojeva - ukupno 104 bloka, tako da je svaki procesor bio zadužen za prosejavanje 26 blokova. Rad se obavljao i ciklusima, i na kraju svakog od 26 ciklusa glavni procesor je sakupljao obrađene intervale od ostalih procesora. Najveći deo vremena potrošen je na pristup spoljašnjoj memoriji, jer dobijena baza zauzima izuzetno veliki memorijski prostor.

Opišimo još realizaciju glavnog dela algoritma -

prosejavanje intervala. Kao osnovnu meru uzimamo interval dužine $L=2\cdot3\cdot5\cdot7\cdot11\cdot13=30030$, pa se ceo skup koji obrađujemo sastoji iz intervala $[k\cdot L+1,(k+1)\cdot L]$ gde je $0\leq k<32\cdot4\cdot26$. Svaki od 4 procesora ukupno 26 puta izvršava prosejavanja intervala dužine $32\cdot L=9600960$ koji se zatim "pakuje" i šalje glavnom procesoru. Za obradu intervala koristimo niz čiji su elementi 32-bitni registri (celobrojni tip u FORTRANU). Niz ima 15015 članova i pri tome I-ti bit J-tog člana nosi informaciju o tome da li je broj $dg+(I-1)\cdot30030+2\cdot J+1$ prost ili složen, gde je $dg+1$ donja granica intervala koji obrađujemo, $dg\equiv0(mod30030)$. Već na početku smo uštedeli prostor i vreme isključivši sve parne brojeve. Dobitak u vremenu ostvarićemo takođe i izbegavanjem deljenja sa brojevima 3,5,7,11 i 13, a prosejavanje vršimo samo prostim brojevima većim od 13.

Po završetku prosejavanja ceo interval pakujemo u drugi, manji niz. Za ovaj posao koristimo ranije izračunat niz UZP čiji su elementi svi brojevi uzajamno prosti sa $L=30030$ a istovremeno manji od L. Koristeći činjenicu da ako je $dg+r$ prost broj onda mora biti $NZD(r,30030)=1$ (jer je $dg\equiv0(mod\ 30030)$), i znajući da niz UZP ima $\varphi(30030)=5760$ članova (φ-Ojlerova funkcija), informacije o prostim brojevima iz intervala dužine 30030 možemo čuvati na 5760 bitova odnosno u 720 bajta. Sada konačno dobijamo ceo algoritam: prvo sve članove niza dužine 15015 inicijalizujemo vrednošću 0 a zatim u svaki bit koji predstavlja broj deljiv sa nekim prostim brojem većim od 13 postavimo vrednost 1. Na kraju vršimo pakovanje dobijenog rezultata tako što pregledamo sve bitove koji predstavljaju brojeve uzajamno proste sa 30030. Tako smo umesto jednog bita za svaki broj trošili po jedan bit na svakih $30030:5760\approx5$ brojeva čime smo ostvarili uštedu od 80%.

Baza prostih brojeva koju smo dobili na ovaj način može nam biti od višestruke koristi. Ispitivanje da li je neki broj prost lako izvodimo pretraživanjem date baze (ukoliko je dati broj u granicama obrađenog intervala), rastavljanje broja na proste činioce (odnosno nalaženje činilaca koji pripadaju obrađenom intervalu) može se jednostavno izvršiti sekvencijalnim prolaskom kroz bazu, a provera ispravnosti nekih hipoteza o prostim brojevima takođe se zasniva na sekvencijalnoj obradi ove baze.

5. MNOŽENJE BROJEVA SA VIŠESTRUKOM PRECIZNOŠĆU

Rešenje problema pomoću linearnog procesorskog niza

Najjednostavnija procesorska arhitektura je niz procesora $P_0,P_1,\ldots,P_{k-1}$ tako da svaka dva procesora P_i i P_{i+1} $(0\leq i\leq k-1)$ mogu komunicirati u oba smera. Svi podaci su na početku smešteni u procesoru P_0 koji je zadužen za ulaz/izlaz, pa se mora napraviti algoritam koji će omogućiti ostalim procesorima da dovoljno brzo dobiju podatke koje treba obrađivati. Ukoliko se arhitektura procesora posmatra kao graf čiji su čvorovi procesori, a grane grafa su veze između procesora, možemo govoriti i o topološkim osobinama određene arhitekture. Procesorski niz se može utopiti u većinu poznatih arhitektura, što znači da se i algoritam koji ćemo izložiti može primeniti na te arhitekture.

Neka su prirodni brojevi X i Y dati nizovima svojih cifara $X=(x_{n-1}x_{n-2}\ldots x_0)$ i $Y=(y_{n-1}y_{n-2}\ldots y_0)$ koji se nalaze u memoriji procesora P_0 procesorskog niza $P_0,P_1,\ldots,P_{n-1}$. Algoritam možemo opisati u tri koraka: slanje potrebnih podataka ostalim procesorima, samostalni rad procesora na računanju potrebnih vrednosti i vraćanje dobijenih vrednosti glavnom procesoru P_0.

Algoritam M1:

(Ulaz: nizovi cifara prirodnih brojeva $X=(x_{n-1}x_{n-2}\ldots x_0)$

```
01  DO in parallel 0≤i≤n-1
02     P_i, i=0 :
03          FOR j=0,n-1
04               SEND(x_j,RIGHT)
05               PAUSE(1)
06          FOR j=0,n-2
07               SEND(y_j,RIGHT)
08               PAUSE(1)
09          SEND(y_n-1,RIGHT)
10          Z_o=X_o Y_o
11          z_n=0
12          FOR j=0,n-1
13               z_n=z_n+x_j y_n-j
14          DIV(z_o,B,p,z_o)
15          FOR i=1,n-1
16               RECEIVE(z_i,RIGHT)
17               DIV(z_i+p,B,p,z_i)
18          DIV(z_n+p,B,p,z_n)
19          FOR i=n+1,2n-2
20               RECEIVE(z_i,RIGHT)
21               DIV(z_i+p,B,p,z_i)
22     IF p>0 THEN
23                    k=2n
24                    z_k-1=p
25                ELSE
26                    k=2n-1
27     P_i, 0<i<n-1 :
28          PAUSE(i-1)
29          FOR j=0,n-1
```

```
30          RECEIVE(x_j,LEFT)
31          SEND(x_j,RIGHT)
32      FOR j=0,n-1
33          RECEIVE(y_j,LEFT)
34          SEND(y_j,RIGHT)
35      z_i=0
36      FOR j=0,i
37          z_i=z_i+x_j y_{i-j}
38      z_{n+i}=0
39      FOR j=0,i
40          z_{n+i}=z_{n+i}+x_j y_{n+i-j}
41      P_i, imod2=1:
42          FOR j=0,n-2-i
43              SEND(z_{i+j},LEFT)
44              RECEIVE(z_{i+j+1},RIGHT)
45          SEND(z_{n-1},LEFT)
46          FOR j=0,n-3-i
47              SEND(z_{n+i+j},LEFT)
48          RECEIVE(z_{n+i+j+1},RIGHT)
49          SEND(z_{2n-2},LEFT)
50      P_i, imod2=0 :
51          FOR j=0,n-2-i
52              RECEIVE(z_{i+j+1},RIGHT)
53              SEND(z_{i+j},LEFT)
54          SEND(z_{n-1},
55          FOR j=0,n-3-i
56              RECEIVE(z_{n+i+j+1},RIGHT)
57              SEND(z_{n+i+j},LEFT)
58          SEND(z_{2n-2},LEFT)
```

```
59      P_i, i=n-1 :
60          PAUSE(n-1)
61          FOR j=0,n-1
62              RECEIVE(x_j,LEFT)
63              PAUSE(1)
64          FOR j=0,n-1
65              RECEIVE(y_j,LEFT)
66              PAUSE(1)
67          z_n-1=0
68          FOR j=0,n-1
69
70          IF nmod2=1 THEN
71                          PAUSE(1)
72                          SEND(z_n-1,LEFT)
73                      ELSE
74                          SEND(z_n-1,LEFT)
```

Svi procesori obavljaju istu vrstu posla ali je zbog izvesnih razlika potrebno pisati programe za pocetni, krajnji, i procesore sa parnim odnosno neparnim indeksom. U navedenom algoritmu naredbe 02-27 odgovaraju prvom, 59-74 poslednjem, a naredbe 28-58 ostalim procesorima, pri cemu u jednom trenutku treba izvršiti podelu u zavisnosti od parnosti indeksa procesora. Rad procesora je sinhronizovan - izvodi se u taktovima - tako da svaka numerisana naredba predstavlja rad koji procesor izvrsava u jednom taktu. Ovakva organizacija rada nam je potrebna zbog prvog

ostalih procesora, a samim tim i poteškoća pri vremenskoj proceni. Prvi deo algoritma predstavlja slanje potrebnih podataka ostalim procesorima, jer na početku samo procesor P_0 poseduje ulazne podatke. Pri tome moramo voditi računa o činjenici da u jednom taktu procesor može samo primiti ili samo poslati podatak (isključeno je istovremeno slanje i primanje podataka). Drugi deo algoritma prestavlja izračunavanje određenih izraza i realizovan je tako da svi procesori troše podjednako vreme na njegovo izvršavanje. Treći deo algoritma je urađen analogno prvom delu i predstavlja slanje izračunatih vrednosti glavnom procesoru.

Nađimo još vreme $T_1(n)$ potrebno za izvršavanje algoritma. Pošto procesor P_0 započinje a takođe i završava rad algoritma, to je ukupno vreme jednako vremenu potrebnom procesoru P_0 da izvrši svoj deo posla. Dato vreme je sada lako izračunati jer svaka petlja u sebi sadrži samo proste naredbe. Posle sabiranja dobijamo $T_1(n)=9n-1=O(n)$ taktova potrebnih za obavljanje posla. S obzirom da je vreme potrebno za rešavanje datog problema analognim sekvencijalnim algoritmom $O(n^2)$ to je ubrzanje koje smo dobili asimptotski idealno. Međutim, postoje sekvencijalni algoritmi koji navedeni problem rešavaju u vremenu bliskom linearnom, te ovaj algoritam ne predstavlja najefikasniju primenu paralelizacije sa teoretske tačke gledista. Sa druge strane, pošto je skrivena konstanta u pomenutim sekvencijalnim algoritmima izuzetno velika, primena algoritma M1 u praksi se pokazuje vrlo efikasnom.

Množenje brojeva pomoću procesora sa zajedničkom memorijom

Neka su dati brojevi $X=(x_{n-1}x_{n-2}\ldots x_0)$ i $Y=(y_{n-1}y_{n-2}\ldots y_0)$ predstavljeni kao nizovi svojih cifara i neka imamo procesore

$P_0, P_1, \ldots, P_{n-1}$ sa zajedničkom EREW memorijom. Neka je dalje $Z=X\cdot Y$ i $Z=(z_{k-1}z_{k-2}\ldots z_0)$. Ako stavimo da je $\underline{z}_i=\sum x_j y_{i-j}$, $p_i=(\underline{z}_i+p_{i-1})\text{div}B$, $p_{-1}=0$, tada je $z_i=(\underline{z}_{i-1}+p_{i-1})\text{mod}B$ i algoritam će se sastojati iz dva dela, paralelnog, za računanje vrednosti $\underline{z}_i$, i sekvencijalnog za računanje vrednosti z_i. Glavni problem pri pravljenju prvog dela je kako rasporediti procesore da obavljaju približno istu količinu posla, da njihova iskorišćenost bude što veća, a što je i najvažnije, da se ne sukobljavaju pri čitanju memorije. Ideja je da svaki procesor osim jednog bude zadužen za računanje vrednosti $\underline{z}_i$ i $\underline{z}_j$ takve da je $i-j=n$. Pošto za računanje vrednosti z_i odnosno z_{2n-i-2} treba izvršiti $i+1$ množenja i $i+1$ sabiranja, to algoritam predviđa da procesor P_i bude angažovan na nalaženju vrednosti z_i i z_{n+i} čime će izvršiti ukupno n množenja i n sabiranja.

Algoritam M1:

(Ulazne vrednosti: nizovi cifara brojeva X i Y)
(Izlazne vrednosti: niz cifara broja $Z=X\cdot Y$)
(Arhitektura: procesori $P_0, P_1, \ldots, P_{k-1}$
sa zajedničkom EREW memorijom)

```
01      DO in parallel 0≤i≤n-1
02           Pi: zi=0
03               zn+i=0
04               ri=(i+1)/2
05               si=i/2
06      DO k=0,n-1
07           DO in parallel 0≤i≤n-1
```

```
08              P_i, imod2=0 :
09                      z_{r_i+s_i} = z_{r_i+s_i} + x_{r_i} · y_{s_i}
10                      r_i = (r_i+1) mod n
11                      s_i = (s_i+n-1) mod n
12              P_i, imod2=1 :
13                      z_{r_i+s_i} = z_{r_i+s_i} + y_{s_i} · x_{r_i}
14                      s_i = (s_i+1) mod n
15                      r_i = (r_i+n-1) mod n
16      p=0
17      DO i=0,2n-2
18          z_i = (z_i+p) mod B
19          p_i = (z_i+p) / B
20      IF p>0 THEN k=2n-1
21                  z_k = p
22              ELSE k=2n-2
```

Teorema 1. Procesor P_i $(0 \le i < n-1)$ po završetku prvog dela (01–15) algoritma M1 sadrži u promenljivima z_i i z_{n+i} vrednosti izraza $\Sigma x_j y_{i-j}$ odnosno $\Sigma x_j y_{i-j}$, a za vreme izračunavanja datih vrednosti ne dolazi u sukob sa drugim procesorima pri čitanju podataka iz memorije.

Dokaz: Dokažimo da r_i+s_i uzima vrednosti iz skupa $\{i, n+i\}$ gde je $0 \le i \le n-1$, tokom celokupnog izvršavanja drugog DO ciklusa (06–15) algoritma.

1) $i=2j$

Na početku je $r_i=j$ i $s_i=j$ pa je $r_i+s_i=2j=i$. U svakom narednom ciklusu izvršava se transformacija (14–15) i tada se r_i uvećava za 1 a s_i umanjuje za 1 pa zbir r_i+s_i ostaje nepromenjen. Izuzetak

je samo u slučaju kad je $r_i=n-1$ odnosno $s_i=0$. Međutim, pošto je $i<n$ to je $n-i/2>i/2$ pa je $n-1-r_i \geq s_i$ što znači da se slučaj $s_i=0$ nikad ne pojavljuje posle slučaja $r_i=n-1$. Dakle, suma r_i+s_i se tada uveća za n pa je $r_i+s_i=n+i$. Tek posle ovoga može nastati slučaj $r_i=n-1$ posle koga se ceo zbir smanjuje za n pa je tada $r_i+s_i=i$. To znači da procesor P_i menja samo vrednosti promenljivih z_i i z_{n+i}. Pored toga, r_i i s_i uzimaju tačno jednom vrednosti iz skupa $\{0,1,\ldots,n-1\}$ pa posmatrajući naredbe (09-11) vidimo da se na kraju rada u promenljivim z_i i z_{n+i} nalaze vrednosti navedene u iskazu teoreme.

Ovaj slučaj se dokazuje analogno slučaju 1) pa nema potrebe detaljno ga opisivati.

Ostaje još da se pokaže kako procesori ne dolaze u sukob pri uzimanju odgovarajućih vrednosti iz memorije. Očigledno je da dva procesora čiji su indeksi različite parnosti ne dolaze u sukob jer dok jedan čita vrednost člana niza X drugi čita vrednost nekog člana niza Y. S druge strane, ako posmatramo dva procesora sa indeksima iste parnosti, na primer P_i i P_j, pri čemu je $i=2u$ a $j=2v$, tada je $u=r_i \neq r_j=v$, $u=s_i \neq s_j=v$. Međutim, posmatrajući operacije (09-11), vidimo da će na početku svakog ciklusa važiti: $|r_i-r_j| \equiv |u-v| (\mathrm{mod} n)$ i $|s_i-s_j| \equiv |u-v| (\mathrm{mod} n)$ pa nikad neće biti $r_i=r_j$ odnosno $s_i=s_j$. To u stvari znači da procesori P_i i P_j neće dolaziti u sukob jer nikad u istom trenutku neće upućivati memoriji zahtev za čitanje vrednosti iste promenljive.

Vreme algoritma je sad jednostavno proceniti, iz zapisa

paralelizaciju odgovarajućeg sekvencijalnog algoritma. 'Usko grlo' algoritma je poslednji deo (16-22) kada radi samo jedan procesor, ali je gubitak vremena koji se tu pojavljuje zanemarljiv kada je n veliko.

6. Da li je N prost ili složen

Ispitivanje da li je neki broj prost je jedan od važnijih problema koji često predstavlja samo deo u rešavanju nekog složenijeg problema. Faktorizacija brojeva je uspešno završena tek onda kad se pokaže da su činioci dobijeni na kraju prosti. Stoga se svi oni moraju podvrgnuti nekom testu koji će ispitivati njihovu složenost. Očigledno je da se ovakav algoritam mora maksimalno ubrzati kako bi glavni algoritam bio rešen u nekom kraćem vremenu.

Važnije teoreme teorije brojeva upotrebljive pri prepoznavanju prostog broja

Teorema 1. (*Pierre Fermat*) Ako je p prost broj, a x ceo broj takav da $p \nmid x$, tada je $x^{p-1} \equiv 1 \pmod{p}$.

Definicija 1. Ako je p prost i (a,p)=1, tada kažemo da je a kvadratni rezidum za p ukoliko kongruencija $x^2 \equiv a \pmod{p}$ ima rešenja u skupu prirodnih brojeva i pišemo (a/p)=1, inače je a kvadratni nerezidum za p i pišemo (a/p)=-1.

Definicija 2. Za date brojeve a i n, pri čemu je n neparan i (a,n)=1 definišemo Jakobijev simbol (a/p) kao

$$(a/p)=\prod(a/p_i)^{\alpha i}, \text{ ako je } n=\prod p_i^{\alpha i}$$

Teorema 2.(*Leonhard Euller*) Ako je p prost neparan broj i a prirodan broj takav da je (a,p)=1 tada je $a^{(p-1)/2} \equiv (a/p) \pmod{p}$.

Očigledno da Teorema 2 predstavlja uopštenje Teoreme 1. Nama bi ipak od veće koristi bili obratni stavovi ovih teorema. Još 500. g. pre n.e. Kinezima je bila poznata činjenica da $p|2^p-2$ ako je p prost, a Fermat je dao opšte tvrđenje 1640.god. Međutim, u kineskim rukopisima se navodi obratno tvrđenje ($2^{n-1}-1$ nije deljivo sa n ako je n složen), a Leibniz čak navodi i dokaz obrnutog Fermatovog stava. Tek je 1830. naden kontraprimer, jer je $341=11\cdot 31$ a $341|2^{340-1}$. Čime nastaje novi pravac u Teoriji brojeva. Ostaje nam samo da primenom kontrapozicije dobijemo manje korisne teoreme, a za dalji rad uvodimo definicije:

Definicija 3. Ako je dat složen neparan broj n i ako postoji prirodan broj a takav da je $(a,n)=1$ i $a^{n-1}\not\equiv 1 \pmod n$ tada kažemo da je n Fermatov pseudoprim za osnovu (bazu) a.

Definicija 4. Neparan složen broj n se zove Carmichael-ov ako je Fermatov pseudoprim za svaku bazu.

Definicija 5. Ako je dat neparan broj n i ako postoji prirodan broj a takav da je $(a,n)=1$ i $a^{n-1}\not\equiv 1 \pmod n$ tada kažemo da je n Euler-ov pseudoprim za osnovu (bazu) a.

Definicija 6. Ako je dat neparan složen broj $n=2^s d+1$ pri čemu je d neparno i ako postoji a tako da je ili $a^d\equiv 1 \pmod n$ ili $a^{d\cdot 2^r}\equiv -1 \pmod n$ za neko $r=0,1,2,\ldots,s-1$ kažemo da je a jak pseudoprim za osnovu (bazu) a..

Lako je videti da je skup Fermatovih pseudoprimova podskup skupa Ojlerovih pseudoprimova, a može se dokazati i da je svaki

jak pseudoprim istovremeno i Ojlerov pseudoprim. U mnogim radovima se testovi bazirani na gornjim definicijama i teoremama nazivaju testovi primalnosti što je pogrešno, jer su to u stvari testovi složenosti pošto mogu samo dokazati da je broj složen. Ako je broj prost ovim testovima se to ne može dokazati u opštem slučaju. Brz test primalnosti se može dobiti primenom tablice pseudoprimova koju treba iskoristiti ako se složenost broja ne pokaže nekim od navedenih testova. Iskoristićemo rezultate objavljene u [21], gde su dati rezultati dobijeni ispitivanjem brojeva manjih od $25 \cdot 10^9$. Ispitivanje je pokazalo da ispod date granice postoji 1770 brojeva

koji su pseudoprimovi istovremeno za baze 2,3,5 i 7 pa uz pomoć 4 procesora koji će ispitivati pseudoprimalnost određenog broja za date 4 osnove dok peti procesor ispituje da li se dati broj nalazi u tablici, dobijamo veoma brz test. Šta više, jakih pseudoprimova istovremeno za osnove 2,3 i 5 ima 13, ako ubacimo u ispitivanje i 7 kao osnovu dobijamo samo broj 3215031751 kao izuzetak, a i on otpada proverom uslova za bazu 11. Sada već možemo da biramo koji ćemo algoritam upotrebiti u zavisnosti od raspoloživog vremena, broja procesora i dostupnog memorijskog prostora (zbog smeštanja tablice). Vratićemo se konstrukciji datog algoritma kasnije - u temi o probabilističkim algoritmima, zasad recimo samo da je ovaj slučaj zanimljiv pri pravljenju algoritma koji treba usput, brzo da reši i problem primalnosti broja. Za ispitivanje složenosti nekog do sada ne ispitivanog broja ne možemo primenjivati tablice,

Kako dokazati da je broj prost ?

Videli smo da prethodni testovi ne mogu dati dokaz da je neki broj prost, već samo pri ostvarenju određenih uslova dokazuju da je broj složen. U problemu dokazivanja primalnosti nekog broja glavnu ulogu igra sledeća teorema

Teorema 3. (*Eduard Lucas*) Neka je $n-1=\prod p_i^{\alpha_i}$ gde su p_i $(1\le i\le m)$ različiti prosti brojevi. Ako postoji a tako da je

(1) $$a^{(n-1)/p_i}\not\equiv 1 \pmod n \text{ za } j=1,2,\dots,m$$

i tako da je

(2) $$a^{n-1}\equiv 1 \pmod n$$

tada je n prost.

Već na početku vidimo da se pojavljuje problem rastavljanja broja na proste činioce, dakle moramo da pozovemo u pomoć i neki od metoda faktorizacije o kojoj će kasnije biti reči. No imajući u vidu da je n neparan broj koji ima kao delioce velike brojeve (ako ih uopšte ima), to je n-1 paran, a velika je verovatnoća da ima mnogo manjih činilaca. Ideja je da svakom nađenom p_i dodelimo odgovarajući procesor koji će za unapred određeno a proveravati kongruenciju (1), ali se pojavljuje problem međusobne zavisnosti procesora. Svaki od njih završi svoj posao a zatim čeka rezultate ostalih i tek tada zna za koje a treba da nastavi rad i da li uopšte treba da ga nastavi. Na taj način se ne ostvaruje velika iskorišćenost procesora pa ćemo oslabiti uslove prethodne teoreme:

Teorema 4. (*John Selfridge*) Neka je $n-1=\prod p_i^{\alpha_i}$ gde su p_i različiti prosti brojevi. Ako za svako p_i postoji a_i takvo da je

$a_i^{(n-1)/p_i} \not\equiv 1 \pmod{n}$ i $a_i^{n-1} \equiv 1 \pmod{n}$

tada je n prost.

Sada možemo opisati algoritam za izvođenje dokaza da je neki broj prost. Svaki procesor dobija broj n, izvršava faktorizaciju broja n-1, a zatim uzima odgovarajući prost činilac p_i i pokušava da nađe a tako da je

$a^{(n-1)/p_i} \not\equiv 1 \pmod{n}$ i $a^{n-1} \equiv 1 \pmod{n}$,

i zatim šalje odgovarajući izveštaj glavnom procesoru.

Algoritam P1:

{Ulaz: prirodan broj n}

{Izlaz: niz $p_1,\ldots,p_n$ prostih činilaca od n-1, i niz $a_1,\ldots,a_n$ prirodnih brojeva iz teoreme 4 ako je n prost, i a takvo da je $a^{n-1} \not\equiv 1 \pmod{n}$ ako je n složen.}

{Arhitektura: k nezavisnih procesora koji mogu stupati u komunikaciju (direktnu ili indirektnu) sa glavnim procesorom P}

```
01   DO in parallel i=1,k
02        Pi: izvršiti faktorizaciju broja n-1=∏pi^αi, 1≤i≤n
03            IF i>n THEN SEND(P,0)
```

09 IF $b^{P_i} \equiv 1 \pmod{p}$ THEN SEND(P,a)

10 ELSE SEND(P,-1)

11 SEND(a_i,P)

Vremensku procenu navedenog algoritma nije moguće izvršiti jer ne zavisi od veličine ulaza, već i od broja različitih faktora od n-1, i veličine najvećeg od tih faktora. Razmatrajući algoritam vidimo da se glavni deo vremena troši na stepenovanje po modulu n. Međutim, pošto se taj algoritam izvršava relativno brzo (nalaženje m-tog stepena se izvršava pomoću logm množenja), možemo reći da je njegovo vreme rada sasvim zadovoljavajuće. Problem se pojavljuje zbog slabe iskorišćenosti procesora u slučaju kada n-1 ima malo različitih prostih faktora. Sa druge strane, kada je n isuviše veliko za smeštanje u registar, višak procesora nam može dobro doći za množenje pri stepenovanju. Ovo je pravi primer kako je uopšteni problem veoma teško rešiti tako da se procesori što bolje iskoriste. Mnogo je bolje praviti poseban algoritam za svaki konkretan problem.

Probabilistički paralelni algoritmi za ispitivanje primalnosti

Algoritam naveden u prethodnom paragrafu nalazi dokaz da je broj prost ili da je složen. Međutim, vreme njegovog izvršavanja može biti veoma veliko u nekim situacijama. Kao što je ranije rečeno, ispitivanje primalnosti broja se često izvrvšava kao deo nekog algoritma, i pogotovo kada to treba uraditi više puta algoritam P1 nije baš najpogodniji. Zato u slučaju da imamo neko unapred zadato vreme za koje je potrebno rešiti problem, od velike pomoći mogu biti probabilistički algoritmi. Naravno, ti algoritmi

se ne mogu koristiti za izvodjenje dokaza u "čistoj" matematici, ali mogu biti od velike pomoći u kriptologiji ili pri pravljenju hipoteza.

Najednostavniji primer za probabilistički algoritam se dobija pomoću Fermat-ove teoreme. Ukoliko broj zadovoljava tražene uslove, možemo reći da je on prost ili je Fermatov pseudoprim za ispitivanu osnovu. Ukoliko želimo povećati moć testa (ili preciznije: smanjiti verovatnoću pravljenja greške) ponavljamo test za neku drugu bazu. Složeni brojevi koji za svaku bazu ispunjavaju uslove Fermat-ove teoreme nazivaju se Carmichael-ovi brojevi. Danas se dosta toga zna o Carmichael-ovim brojevima, ima ih mnogo manje nego prostih, ali ipak isuviše mnogo da bi Fermat-ova teorema za kratko vreme davala rezultate sa dovoljno malom verovatnoćom greške. Rad sa više procesora može značajno skratiti vreme (ili smanjiti verovatnoću greške, zavisno od toga šta nam je važnije) tako što se svakom procesoru dodeli jedna ili više baza za koju se ispituju uslovi Fermat-ove teoreme. Analogno sa do sada rečenim možemo iskoristiti i Euler-ovu teoremu.

Mnogo jači test možemo dobiti korišćenjem uslova definicije 5, što je pokazao Rabin [25] 1977. godine.

Definicija 6: Neka je n prirodan broj. Reći ćemo da je prirodan broj b svedok složenosti za n i označavati sa $W_n(b)$ ako su ispunjena sledeća dva uslova:

Upoređujući uslove definicije 6. sa uslovima ranije navedenih definicija, i znajući činjenice do sada navedene, lako zaključujemo da iz $W_n(b)$ sledi da je n složen.

Teorema 5. Ako je $n>4$ slo/en broj, tada je

$$3(n-1)/4 \leq c(\{b|W_n(b)\}),$$

gde c(S) označava broj elemenata skupa S.

Navedena teorema u stvari kazuje da ukoliko je neki broj složen, tada postoji bar 3(n-1)/4 brojeva koji svedoče da je on složen. Dakle, ako ispitujemo primalnost nekog broja n, i ako se ispostavi da za neku bazu b ne važe uslovi iz definicije 6. tada možemo reći da je n prost i pri tome je verovatnoća da smo napravili grešku jednaka 1/4. Ako uzmemo niz dužine k

$1<b_1<b_2<\ldots<b_k<n$

i ustanovimo da ne važi $W_n(b_i)$ za $1\leq i\leq k$, možemo reći da je n prost i pri tome je verovatnoća da smo napravili grešku $(1/4)^k$. Dakle u 4^k ispitivanja primalnosti pomoću nizova dužine k pravimo jednu grešku, što je veoma dobar rezultat, znajući da ga možemo dobiti koristeći paralelni algoritam. Svakom procesoru dodeljujemo po jednu bazu za koju on ispituje $W_n(b)$ (pretpostavka je da imamo k procesora).

Algoritam P2:

{Ulaz : prirodan broj n}

{Izlaz: izvestaj o složenosti broja n; u slučaju da odgovor bude "n je prost" verovatnoća da je napravljena greška iznosi 4^{-k}}

{Arhitektura: k procesora povezanih sa glavnim procesorom}

```
01   DO in parallel 0≤i≤k-1
02        P_i:  d=n-1
03              WHILE dmod2=0 DO
04                    d=d/2
05              b=f(i,n)
06              IF n mod b=0 THEN SEND(P,b)
07              b=b^d(mod n)
08              IF THEN SEND(P,b)
09              d=2d
10              WHILE d<n DO
11                    b=b^2(mod n)
12                    IF NZD(b,n) > 1 THEN
13                          SEND(P,b)
14                          STOP
15                    d=2d
16              SEND(P,1)
```

Recimo na početku da je funkcija f(i,n) u naredbi (05) korišćena u cilju generisanja broja b, tako da je $1<b<n$, pri čemu se tako dobijeno b razlikuje od odgovarajućih vrednosti za b koje koriste drugi procesori. Očigledno da je vreme datog algoritma jednako $O(\log n)$ jer se ceo rad sastoji iz $\log n$ množenja i modularnog deljenja. Naravno, ukoliko se te dve operacije ne mogu izvršiti u konstantom vremenu, navedenu procenu treba pomnožiti sa vremenom potrebnim za izvršavanje ove dve operacije, tako da i uz primenu najjednostavnijih algoritama za ove dve operacije, dobijamo vreme $Tz(n)=o(\log^3 n)$.

7. FAKTORIZACIJA

Rastavljanje brojeva na proste činioce predstavlja glavni, ali i najteži problem u aritmetici sa višestrukom tačnošću. Algoritmi za faktorizaciju zahtevaju primenu mnogih do sada pomenutih algoritama, i to uglavnom za obavljanje nekih usputnih poslova, pa nije ni malo čudno što do danas još uvek nije napravljen polinomijalan algoritam za faktorizaciju.

Najjednostavnije rešenje problema je ispitivanje deljivosti datog broja n sa prostim brojevima manjim od $n^{1/2}$. Ovaj algoritam je praktičan u slučajevima kada se n može predstaviti kao ceo broj u memoriji računara, dakle za današnje pojmove, u slučaju $n \sim 10^{18}$ (double precision). Svaki od procesora u jednom vremenskom intervalu uzima po jedan prost broj $p \leq n^{1/2}$ iz ranije napravljene baze i proverava da li je nmodp=0. Problem se pojavljuje zbog potrebe velikog prostora za čuvanje baze, a ukoliko baza nije ranije napravljena troši se vreme na primenu algoritma Eratostenovog sita. Oba problema rešavamo tako što ćemo delimično primenititi algoritam Eratostenovo sito a zatim nastaviti proveru deljivosti broja n sa brojevima koji nisu deljivi sa nekoliko prvih prostih brojeva. Za izvršavanje navedenog posla nam nije potreban memorijski prostor koji bi smo koristili u slučaju korišćenja baze prostih brojeva, a izbegavanjem pravljenja baze štedimo i vreme. Sa druge strane, povećava se vreme rada zbog nepotrebnih deljenja složenim brojevima, ali je to povećanje relativno malo.

Pretpostavimo da imamo 8 procesora koji mogu direktno ili indirektno komunicirati sa glavnim procesorom (tu ulogu može

koji nisu deljivi sa 2,3 i 5, dakle brojevima oblika 30k+t gde je t iz skupa {1,7,11,13,17,19,23,29}. Na kraju rada svaki od procesora čuva kao konačni rezultat sve faktore od n koji imaju odgovarajući ostatak pri deljenju sa 30.

Algoritam F1.

{Ulaz: 1) prirodan broj n čija faktorizacija je nepoznata
2) prirodan broj G - granica do koje vršimo proveru }
{Izlaz: prosti faktori od n manji od G }
{Konfiguracija: 8 povezanih procesora $P_0, P_1, \ldots, P_7$

```
01  CONST (r0,r1,r2,r3,r4,r5,r6,r7)=(1,7,11,13,17,19,23,29)
02  DO in parallel 0≤i≤7
03      Pi:  m=n
04           g=⌊min{g,n^(i/2)}⌋
05           S={x|2≤x≤g, xmod30=ri}
06           D=∅
07           WHILE d∈S AND m≠1 DO
08                IF d|m THEN
09                   D=D∪{d}
10                   WHILE d|N DO
11                        N=N/d
12  FOR j=2 DOWNTO 0
13      DO in parallel 0≤i≤2^(j+1)-1
14          Pi, i≥2^j:
15              SEND(P(i-2^j),D)
16          Pi, i<2^j:
17              RECEIVE(P(2^j+i),E)
```

```
18              D=DUE
19              m=n
20              WHILE (d∈D) AND (m≠1) DO
21                      IF d|m THEN
22                              WHILE d|m DO
23                                      m=m/d
24                      ELSE D=D\{d}
```

Drugi deo algoritma (12-24) predstavlja slanje dobijenih rezultata glavnom procesoru pri čemu se vrši izbacivanje onih činilaca od n koji su složeni. U svakom ciklusu FOR petlje polovina procesora šalje skup dobijenih faktora od n drugoj polovini procesora, koji vrše izbacivanje jednog dela složenih faktora. Konačan rezultat je skup svih prostih deliioca od n manjih od date granice.

Izračunajmo vreme rada datog algoritma. Prvi deo (2-11) je idealna paralelizacija odgovarajućeg sekvencijalnog algoritma. Na svakih 30 uzastopnih brojeva vrši se provera deljivosti n sa osam brojeva a svaki procesor vrši jednak broj deljenja, Ukupno se proverava deljivost broja n sa $n^{1/2}\cdot 8/30$ brojeva, što znači da svaki procesor izvrši $n^{1/2}/30$ deljenja. Konkretno za $n\sim 10^{9}$ izvršićemo $n^{1/2}\sim 10^{3}$ deljenja što je za pojam današnjih računara izuzetno brz posao. Drugi deo algoritma možemo zanemariti jer se izvršava relativno brzo (vreme je logaritamski zavisno od broja procesora)

Opisaćemo u kratkim crtama i algoritam koji se da primeniti u opštem slučaju. Ako je dato k procesora, onda prvo nađemo najveće $m_l=p_1p_2\ldots p_l$ tako da je $\varphi(m_l)<k$ (ovde je φ Euler-ova funkcija). Zatim interval $[1..[n^{1/2}]]$ podelimo na intervale dužine

$k\cdot\varphi(m_l)$ i u svakom intervalu podelimo procesorima po $\varphi(m_l)$ brojeva koji su potencijalni delioci od n. Pre toga ćemo iz intervala$[1..n^{1/2}]$ izbaciti brojeve deljive sa $p_1,p_2,\ldots,p_l$. Pri tome je dovoljno prosejati interval $[1..m_l]$, jer važi

$(r,m_l)=1 \Leftrightarrow (s\cdot m_l+r,m_l)=1.$

U ovom trenutku navedeni algoritam je samo od teoretske koristi jer i manji broj procesora izvršava posao dovoljno brzo. Valja očekivati da će daljim poboljšanjima hardverskih osobina računara (omogućavanje rada sa 64-bitnim ili čak većim registrima) ovaj algoritam naći potpunu primenu, ali će faktorizacija brojeva od po 100 cifara morati da se izvršava drugim algoritmima.

Fermatov metod za faktorizaciju

Metod koji ćemo izložiti uveo je Fermat u XVII veku i do pojave računara to je bio najefikasniji metod za rastavljanje prostih brojeva na činioce.

Cilj algoritma je predstaviti dati broj n kao razliku kvadrata dva prirodna broja x i y. Tada dobijamo $x^2-y^2=n$, odnosno $(x-y)(x+y)=n$ pa ako je $x-y\neq 1$ dobijamo da su x-y i x+y delioc od n. Problem rešavamo tako što definišemo niz

$x_0=[n^{1/2}]$,

$x_i=x_{i-1}+1$ za $i>0$,

pa nam sad ostaje da proveramo da li je x_i^2-n potpun kvadrat kad je i manje od izvesne granice. Dati niz ćemo podeliti odgovarajućem broju procesora koji će proveravati traženi uslov. U cilju uštede vremena za kvadriranje celih brojeva koristimo činjenicu da je $(m+1)^2=m^2+2m+1$. Takođe, znajući da između m^2 i $(m+1)^2$ nema celih brojeva, uštedećemo veliko vreme izbegavanjem

korenovanja. Pre početka rada proverićemo nekom od prethodnih metoda da li n ima kao činioce neke manje brojeve. Pošto smo se uverili da n nema činilaca ispod $\psi(n)$ (gde je ψ neka ranije određena funkcija) zaključujemo

$\psi(n)<x=y<n^{1/2} \Rightarrow x=n^{1/2}<y$

$n^{1/2}<x+y<n/\psi(n) \Rightarrow y<n/\psi(n)-x$

i sada dobijamo

$x-n^{1/2}<n/\psi(n)-x \Rightarrow x<(n/\psi(n)+n^{1/2})/2=n^{1/2}+(n/\psi(n)-n^{1/2})/2$.

Stoga interval celih brojeva $(n^{1/2}, n^{1/2}+(n/\psi(n)-n^{1/2})/2)$ delimo na jednak broj intervala - svakom intervalu odgovara po jedan procesor koji ga obrađuje.

Algoritam F2.

{Ulaz: prirodan broj n}
{Izlaz: prosti činioci od n}
{Konfiguracija: k procesora koji mogu ostvariti direktan ili indirektan kontakt sa glavnim procesorom}

```
01   DO in parallel 0≤i≤k-1
02        P_i: step=[([n/ψ(n)]-[n^(1/2)])/2k]
03             x=[n^(1/2)]+i*g
04             ggran=x+step
05             raz=x^2
06             y=[raz^(1/2)]
07             sqry=y^2
08             WHILE (x<ggran) and (raz≠sqry) DO
09                  WHILE raz>sqry DO
10                       sqry=sqry+2y+1
```

```
11                    y=y+1
12                IF raz<sqry THEN
13                    raz=raz+2x+1
14                    x=x+1
15                IF raz=sqry THEN
16                    p=x-y
17                    q=x+y
18                    SEND(P_o,p)
19                    SEND(P_o,q)
```

Ukoliko je procesor pronašao dva činioca od n, šalje ga glavnom procesoru koji na osnovu do tada dobijenih rezultata od drugih procesora nalazi činioce od n. Ako se ispostavi da p odnosno q nisu prosti, algoritam se ponavlja za te brojeve. Vreme potrebno za izvršavanje algoritma F2 se ne može precizno odrediti jer zavisi od činilaca ulaza n. Najnepovoljniji slučaj nastaje kada se činioci nalaze na krajevima intervala koji su dodeljeni procesorima. Stoga je najlošije moguće vreme utrošeno za faktorizaciju broja n : $T_1(n)=O((n/\psi(n)-n^{1/2})/2k)$ a s obzirom da je $\psi(n)\ll n^{1/2}$ to je $n/\psi(n)\gg n^{1/2}$ pa je $T_1(n)=O(n/2k\psi(n))$. Sekvencijalan algoritam troši k puta više vremena pa algoritam F2 predstavlja maksimalno ubrzanje odgovarajućeg sekvencijalnog algoritma.

8. KUREPINA HIPOTEZA O LEVOM FAKTORIJELU

Na kongresu jugoslovenskih matematičara u Ohridu 1970.godine Đuro Kurepa je postavio sledeći problem: definišimo levi faktorijel kao sumu faktorijela svih nenegativnih celih brojeva manjih od datog broja; postavlja se pitanje šta može biti zajednički delilac levog i desnog faktorijela nekog broja. Kurepa postavlja hipotezu da je dvojka najveći zajednički delilac oba ova broja. Mada je problem veoma jednostavan, i na prvi pogled se čini da ne bi trebalo biti nekih poteškoća pri rešavanju, on do danas,

20 godina od nastanka, nije rešen.

Neki iskazi ekvivalentni Kurepinoj hipotezi

U cilju pojednostavljenja problema, pojavili su se mnogi iskazi ekvivalentni iskazu Kurepine hipoteze. Iskaz (E1) koji Kurepa navodi u [5] biće nam od velike pomoći pri proveri (KH) na računaru.

Definicija 1. Levi faktorijel prirodnog broja n (u oznaci !n) definišemo kao

$$!n=\sum i!=0!+1!+\ldots+(n-1)!.$$

Definicija 2. Kurepina hipoteza je iskaz

(KH) $\quad(\forall n>1)\ NZD(!n,n!)=2$

Definicija 3. Neka je P skup prostih brojeva. Iskaz (E1) je

(E1) $\quad(p\in P)(p>2 \Rightarrow !p\not\equiv 0 (\mathrm{mod}\, p))$

definisan na sledeći način:

$$S_0 = V(n,k),$$

$$S_i = S_{i-1} \cdot V(n-ik,k) + W(n-ik,k),\ 1 \le i < l,$$

$$S_l = S_{l-1} \cdot r! + !r,$$

tada je $S_l = !n$.

Dokaz: Neka je $k \in \mathbb{N}$ takvo da je $k < n$. Tada je

$$!n - !(n-k) = \sum (n-i)! - \sum (n-k-i)! = \sum (n-i)! - \sum (n-i)! = \sum (n-i)!$$ a pošto

je $(n-i)! = V(n-i,k-i) \cdot (n-k)!$ to je onda

$$!n - !(n-k) = \sum (n-i)! = \sum V(n-i,k-i) \cdot (n-k)!$$

$$= (n-k)! \sum V(n-i,k-i) = (n-k)! W(n,k)$$

Sada izvodimo formulu

$$!n = \sum (!(n-ik) - !(n-(i+1)k) + !(n-lk)$$

$$= \sum (!(n-ik) - !(n-ik-k)) + !r$$

$$= \sum (n-(i+1)k)! W(n-(i+1)k, n-ik) + !r$$

Uočimo još da za prirodne brojeve $a,b,A,B \in \mathbb{N}$ takve da takve da je $a > b$ važi

$$a! \cdot A + b! \cdot B = b!(a(a-1)\ldots(b+1) \cdot A + B) = b!(V(a,a-b) + B)$$ pa je sada

$$!n = (\ldots(W(n,k) \cdot V(n-k,k) + W(n-k,k)) \cdot V(n-2k,k) + W(n-2k,k)) \cdot$$

$$\cdot V(n-3k,k) + \ldots + W(n-(l-1)k,k)) \cdot r! + !r$$

pa sada navedeni niz ima tu osobinu da je $S_l = !n$.

Algoritam za proveru Kurepine hipoteze

Proveru Kurepine hipoteze izvodimo koristeći ranije izvedene relacije. Domen na kome vršimo ispitivanje je skup svih prostih brojeva, jer smo dokazali ekvivalentnost (KH) sa iskazom (E1). Tako, ako (KH) važi za neke proste brojeve $p_1, p_2, \ldots, p_k$ onda (KH) važi za sve brojeve oblika $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$. U računanju levog

formula:

$$T_1(n)=(1/k)\sum p_i=(1/k)\int \ln x\, d\pi(x)=(1/k)x\pi(x)-(1/k)\int \pi(x)dx \sim$$

$$\sim(1/k)x\pi(x)=x^2/(k\ln x)$$

Dakle $T_1(n)=O(x^2/k\ln x)$ gde je k broj procesora.

Realizacija algoritma na računaru

Navedeni algoritam je realizovan na transpjuterskoj ploči sa četiri transpjutera T800. Program je pisan u jeziku 3L Parallel FORTRAN za transpjutere. Posao je obavljen tako što je prvo napravljena baza prostih brojeva manjih od 1000000, a zatim je svaki procesor ispitivao tačnost iskaza (E1) za odgovarajuću grupu prostih brojeva koja mu je dodeljena. U određenim vremenski intervalima glavni procesor je skupljao informacije o dobijenim rezultatima svakog procesora. Konačan rezultat nije doneo nikakvu promenu u odnosu na dosadašnje rezultate. Za sve proste brojeve manje od milion iskaz (E1) odnosno (KH) je tačan. Preciznije rečeno, svi prirodni brojevi koji nemaju prost delilac veći od milion zadovoljavaju iskaz Kurepine hipoteze. Raniji provere Kurepine hipoteze su vršili Slavić za $n\leq 1000$, Wagstaff za $n\leq 50000$ i Mijajlović za $n\leq 310009$.

9. HIPOTEZA O ALTERNIRAJUĆEM FAKTORIJELU

Veoma slična Kurepinoj hipotezi o levom faktorijelu je hipoteza o alternirajućem levom faktorijelu. Odmah posle navođenja Kurepine hipoteze u [13] se postavlja pitanje da li postoji prirodan broj n za koji je broj

$$(n-1)!-(n-2)!+(n-3)!-\ldots+(-1)^n\cdot 1!$$

deljiv sa nekim činiocem od n. Postavljena je hipoteza da takvo n ne postoji. Pomoću računara je data hipoteza proverena za prvih 45000 prirodnih brojeva, i nije pronađen kontraprimer.

Ispitivanje tačnosti hipoteze pomoću računara

Iskaz date hipoteze je u mnogome sličan iskazu Kurepine hipoteze pa način na koji rešavamo dati problem ne odstupa mnogo od načina na koji smo rešili problem provere Kurepine hipoteze. Bez dokaza navodimo tvrđenje ekvivalentno navedenoj hipotezi:

(AFH) Ako je p prost broj, onda broj $(p-1)!-(p-2)!+\ldots+(-1)^p$ nije deljiv sa p.

Dokaz ekvialentnosti navedenih iskaza teče analogno dokazu ekvivalentnosti iskaza o Kurepinoj hipotezi. Problem smo sada sveli na ispitivanje tačnosti date relacije na skupu ~~prirodnih~~ prvih brojeva. Program za proveru hipoteze napisan je na jeziku 3L Parallel FORTRAN za transpjutere i nalazi se u dodatku na kraju rada. Kao što se može videti, dati program se od programa za proveru Kurepine hipoteze razlikuje samo u unutrašnjem ciklusu [illegible] izvršavanja oba

L I T E R A T U R A

[1] Adleman,L.M., Pomerance C., Rumely,R.S. On distinguishing prime numbers from composite numbers, Annals of Mathematics, 117(1983),173-206

[2] Akl,S.G. The design and analysis of parallel algorithms, Prentice Hall International, 1989

[3] Aho,A.V., Hopcroft,J.E., Ullman,J.D. The design and analysis of computer algorithms, Addison-Wesley Publishing Company, 1974

[4] Bengelloun,S.A. An incremental prime sieve, Acta Informatica 23(1986), 119-125

[5] Blair,D.W., Lacampgne,C.B., Selfridge,J.L. Factoring large numbers on a pocket calculator, American Math. Monthly

[6] Brent,R. The first occurence of large gaps between succesive primes, Math. Comp., vol.27 (1973),959-963

[7] Brent,R., The first occurence of certain large prime gaps, Math.Comp., vol.35 (1980), 1435-1436

[8] Chor,B.,Goldreich,O. An improved parallel algorithm for integer GCD, Algorithmica,5 (1990), 1-10

[9] Cohen,H.,Lenstra,H.W. Primality testing and Jacoby sums, Math. Comp., vol.42 (1984), 297-330

[10] Cormack,G.V.,Williams,H.C. Some very large primes of the form $k \cdot 2^m+1$, Math. Comp., vol.35(1980), 1419-1421

[11] Dickson,L.E. History of the theory of numbers, vol.I, New York, N.Y., 1952

[12] Dixon,J.D. Factorization and primality tests, American Math. Monthly

1981

[14] Hardy,G.H., Wright.E.H. An introduction to the theory of numbers, Oxford University Press, 1954

[15] Kurepa,D., On the left factorial function, Mathematica Balkanica 1 (1971), 147-153

[16] Kurepa,D., On some new factorial propositions, Mathematica Balkanica 4 (1974), 383-386

[17] Lagarias,J.C.,Miller,V.S.,Odlyzko,A.M. Computing $\pi(x)$: The Meissel-Lehmer method, Math.Comp., v.44, 637-560

[18] Leeuwen,J.V.(editor), Algorithms and complexity, MIT Press 1990

[19] Mapes,D.C. Fast method for computing the number of primes less than a given limit, American Math. Monthly

[20] Mijajlović,Ž. On some formulas involving !n and the verification of the !n - hypotesis by use of computers, Publ. Inst. Math., 47 (1990), 24-32

[21] Pomerance,C.,Selfridge,J.L.,Wagstaff,S.S., The pseudoprimes to $25 \cdot 10^9$, Math. Comp., 35 (1980), 1003-1026

[22] Potler,A., Young,J. First occurrence prime gaps, Math. Comp., vol.52 (1989), 221-224

[23] Pritchard,P., Explaining the wheel sieve, Acta Informatica 17(1982) , 477-485

[24] Quinn,M.J. Designing efficient algorithms for parallel computers, McGraw-Hill, 1987

[25] Rabin,M.O. Probabilistic algorithm for testing primality, Journal of Number Theory 12 (1980), 128-138

[26] Riesel,H. Prime numbers and computer methods for factorization, Birkhauser,1985

11. DODATAK A

PROGRAM ZA GENERISANJE ERATOSTENOVOG SITA

- KONFIGURACIONI FAJL
- PROGRAM "GLAVNI", POSAO PRVOG PROCESORA
- PROGRAM "DRUGI" , POSAO DRUGOG PROCESORA
- PROGRAM "TREĆI" , POSAO TREĆEG PROCESORA

- PROGRAM "ČETVRTI", POSAO ČETVRTOG PROCESORA

```
! konfiguracioni fajl programa Eratostenovo sito
! definisanje procesora
processor host
processor first
processor second
processor third
processor fourth

! definisanje kanala koji povezuju procesore
wire ? host[0]   first[0]
wire ? first[1]  second[0]
wire ? first[2]  third[0]
wire ? first[3]  fourth[0]


! definisanje procesa
! informacije o broju ulazno-izlaznih kanala
! informacije o memoriji
task afserver ins=1  outs=1
task filter   ins=2  outs=2  data=10k
task glavni   ins=5  outs=2  data=2500k
task drugi    ins=0  outs=1  data=2500k
task treci    ins=0  outs=1  data=2500k
task cetvrti  ins=0  outs=1  data=2500k

! dodeljivanje procesa procesorima
place afserver  host
place filter    first
place glavni    first
place drugi     second
place treci     third
place cetvrti   fourth

! definisanje kanala koji povezuju procese
connect ? filter[0]    afserver[0]
connect ? afserver[0]  filter[0]
connect ? filter[1]    glavni[1]
connect ? glavni[1]    filter[1]
connect ? drugi[0]     glavni[2]
connect ? treci[0]     glavni[3]
connect ? cetvrti[0]   glavni[4]
```

```
C Program za generisajne prostih brojeva manjih od 10**9
C Primenjuje se metod Eratostenovog sita
C Posao izvode dva procesora simultanim radom

C Program 'GLAVNI' - pored glavnog posla ima zadatak da
C podatke smesta u datoteku

      PROGRAM  GLAVNI

C Ukljucivanje fajla koji omogucuje komunikaciju procesora
      INCLUDE  'E:\TR\F\CHAN.INC'

C Podaci o varijablama
C   UZP-brojevi uzajamno prosti sa 2*3*5*7*11*13
C   PRIM-prosti brojevi potrebni za generisanje sita
C   BLOK-sito
C   BAZA-zavrsna verzija brojeva koji idu u datoteku

      INTEGER  FI,PE6,PE6POLA,DB,CETFI
      INTEGER  LEN,CETLEN,KOLPRIM
      PARAMETER ( FI = 5760,
     *          PE6 = 30030,
     *          CETFI = 23040,
     *          PE6POLA = 15015,
     *          LEN = 960960,
     *          CETLEN = 3843840,
     *          KOLPRIM = 3402,
     *          DB = 5760  )

      INTEGER  UZP(FI),PRIM(KOLPRIM)
      INTEGER  BLOK(PE6POLA),BAZA(CETFI)
      INTEGER  POC,S,T,A,GRAN
      INTEGER  BROJAC,GGRAN

C   INCHANADDR-adresa kanala za komunikaciju
      INTEGER  IN2,IN3,IN4

C   IMEDAT-nazivi datoteka sa zavrsnim podacima
      CHARACTER*8  IMEDAT
      COMMON       /ZAJEDNO/UZP,PRIM
C Inicijalizacija podataka
C     DATA     (PRIM(I),I=1,11)/2,3,5,7,11,13,17,19,23,29,31/
C     DATA     (UZP(I),I=1,6)/1,17,19,23,29,31/


C Pocetak merenja vremena
      CALL ICLOCK(IPOC)

C Uspostavljanje veze sa drugim procesorom

      IN2 = F77_CHAN_IN_PORT(2)
      IN3 = F77_CHAN_IN_PORT(3)
      IN4 = F77_CHAN_IN_PORT(4)
      CALL F77_CHAN_IN_WORD(POC,IN2)
      CALL F77_CHAN_IN_WORD(POC,IN3)
      CALL F77_CHAN_IN_WORD(POC,IN4)
      WRITE(*,'("KOMUNICIRAMO")')

C Generisanje niza UZP
```

```
      POC=LEN

C Radimo isto za svaku od 26 datoteka
      DO 100 BROJAC=1,2
      WRITE (*,'("BRDAT=",I2.2)') BROJAC
C Otvaramo datoteku
      WRITE(IMEDAT,'("PROSTI",I2.2)') BROJAC
      OPEN(23,FILE=IMEDAT,FORM='UNFORMATTED')

C Datoteku punimo  sa deset "baza"
      DO 90 IRED=1,10

      C=POC+LEN
      GGRAN=SQRT(C)
      DO 10 I=1,PE6POLA
   10 BLOK(I)=0

      J=7
   20 I=0
      K=PRIM(J)
      IF(K.LE.GGRAN) THEN
         N=K-MOD(POC,K)
         IF(.NOT.BTEST(N,0)) N=N+K
         N=ISHFT(N+1,-1)
   30       DO 40 L=N,PE6POLA,K
   40       BLOK(L)=IBSET(BLOK(L),I)
         IF(I.LT.31) THEN
            N=L-PE6POLA
            I=I+1
            GOTO 30
         ELSE
            J=J+1
            GOTO 20
         ENDIF
      ENDIF


   50 L=DB+1
      DO 60 I=L,CETFI
   60 BAZA(I)=0


      DO 80 I=0,31
      DO 70 J=1,FI
      IF (.NOT.BTEST(BLOK((UZP(J)+1)/2),I) ) THEN
         K=L+J/32
         BAZA(K)=IBSET(BAZA(K),MOD(J,32))
      ENDIF
   70 CONTINUE
      L=L+180
   80 CONTINUE

      DO 82 I=1,DB
   82 CALL F77_CHAN_IN_WORD(BAZA(I),IN2)
      DO 83 I=2*DB+1,3*DB
   83 CALL F77_CHAN_IN_WORD(BAZA(I),IN3)
      DO 84 I=3*DB+1,4*DB
```

```
      CALL ICLOCK(IKRAJ)
      WRITE(6,110) IPOC,IKRAJ,IKRAJ-IPOC
  110 FORMAT(1X,'TIME: ',/,'S: ',I15/,'E: ',I15/,'PAST: ',I15)
      STOP
      END
```

```
      PROGRAM DRUGI
      INCLUDE 'E:\TR\F\CHAN.INC'

      INTEGER FI,PE6,PE6POLA,DB,CETFI
      INTEGER LEN,CETLEN,KOLPRIM
      PARAMETER ( FI = 5760,
     *          PE6 = 30030,
     *          CETFI = 23040,
     *          PE6POLA = 15015,
     *          LEN = 960960,
     *          CETLEN = 3843840,
     *          KOLPRIM = 3402,
     *          DB = 5760 )

      INTEGER UZP(FI),PRIM(KOLPRIM)
      INTEGER BLOK(PE6POLA),BAZA(CETFI)
      INTEGER POC,S,T,A,GRAN
      INTEGER BROJAC,GGRAN
      INTEGER OUT1

      COMMON  /ZAJEDNO/ UZP,PRIM

C Inicijalizacija podataka

C     DATA    (PRIM(I),I=0,11)/1,2,3,5,7,11,13,17,19,23,29,31/
C     DATA    (UZP(I),I=1,6)/1,17,19,23,29,31/

      OUT1=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUT1)

C Generisanje niza UZP

      CALL GENUZP


C Generisanje niza PRIM
C Primenjuje se metoda 'trial division'

      CALL GENPRIM

      DO 210 I=1,P6POLA
  210 BLOK(I)=0

      DO 250 J=7,300
      I=0
      K=PRIM(J)
      N=ISHFT(K+1,-1)+K
  230 DO 240 L=N,P6POLA,K
      BLOK(L)=IBSET(BLOK(L),I)
  240 CONTINUE
      IF(I.EQ.31) GOTO 250
      N=L-P6POLA
      I=I+1
      GOTO 230
  250 CONTINUE
      BLOK(1)=IBSET(BLOK(1),0)

      DO 310 I=1,DB
  310 BAZA(I)=0
```

```
330 CONTINUE
   L=L+180
335 CONTINUE

   DO 350 I=1,DB
350 CALL F77_CHAN_OUT_WORD(BAZA(I),OUT1)

   POC=CETLEN

   DO 100 BROJAC=1,259
   C=POC+LEN
   GGRAN=SQRT(C)

   DO 10 I=1,PE6POLA
10 BAZA(I)=0


   DO 40 J=7,BRP
   I=0
   K=PRIM(J)
   IF(K.GT.GGRAN) GOTO 50
   N=K-MOD(POC,K)
   IF(.NOT.BTEST(N,0)) N=N+K
   N=ISHFT(N+1,-1)
20 DO 30 L=N,PE6POLA,K
   BLOK(L)=IBSET(BLOK(L),I)
30 CONTINUE
   IF(I.EQ.31) GOTO 40
   N=L-PE6POLA
   I=I+1
   GOTO 20
40 CONTINUE


50 L=1


   DO 60 I=1,DB
60 BAZA(I)=0


   DO 80 I=0,31
   DO 70 J=1,FI
   IF (.NOT.BTEST(BLOK((UZP(J)+1)/2),I) ) THEN
      K=L+J/32
      BAZA(K)=IBSET(BAZA(K),MOD(J,32))
   ENDIF
70 CONTINUE
   L=L+180
80 CONTINUE

   DO 90 I=1,DB
90 CALL F77_CHAN_OUT_WORD(BAZA(I),OUT1)
   POC=POC+CETLEN
100 CONTINUE
   STOP
   END
```

```
      PROGRAM  TRECI
      INCLUDE  'E:\TR\F\CHAN.INC'

      INTEGER  FI,PE6,PE6POLA,DB,CETFI
      INTEGER  LEN,CETLEN,KOLPRIM
      PARAMETER ( FI = 5760,
     *          PE6 = 30030,
     *          CETFI = 23040,
     *          PE6POLA = 15015,
     *          LEN = 960960,
     *          CETLEN = 3843840,
     *          KOLPRIM = 3402,
     *          DB = 5760  )

      INTEGER  UZP(FI),PRIM(KOLPRIM)
      INTEGER  BLOK(PE6POLA),BAZA(CETFI)
      INTEGER  POC,S,T,A,GRAN
      INTEGER  BROJAC,GGRAN,OUT1

      COMMON   /ZAJEDNO/ UZP,PRIM

C Inicijalizacija podataka

C     DATA     (PRIM(I),I=0,11)/1,2,3,5,7,11,13,17,19,23,29,31/
C     DATA     (UZP(I),I=1,6)/1,17,19,23,29,31/

      OUT1=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUT1)



C Generisanje niza UZP

      CALL GENUZP


C Generisanje niza PRIM
C Primenjuje se metoda 'trial division'

      CALL GENPRIM


      POC=2*LEN

      DO 100 BROJAC=1,260
      C=POC+LEN
      GGRAN=SQRT(C)

      DO 10 I=1,PE6POLA
   10 BLOK(I)=0


      DO 40 J=7,BRP
      I=0
      K=PRIM(J)
      IF(K.GT.GGRAN) GOTO 50
      N=K-MOD(POC,K)
      IF(.NOT.BTEST(N,0)) N=N+K
      N=ISHFT(N+1,-1)
   20 DO 30 L=N,PE6POLA,K
      BLOK(L)=IBSET(BLOK(L),I)
```

```
40 CONTINUE


50 L=1


   DO 60 I=1,DB
60 BAZA(I)=0


   DO 80 I=0,31
   DO 70 J=1,FI
   IF (.NOT.BTEST(BLOK((UZP(J)+1)/2),I) ) THEN
      K=L+J/32
      BAZA(K)=IBSET(BAZA(K),MOD(J,32))
   ENDIF
70 CONTINUE
   L=L+180
80 CONTINUE
```

```
90 CALL F77_CHAN_OUT_WORD(BAZA(I),OUT1)

   POC=POC+CETLEN

100 CONTINUE
   STOP
   END
```

```
      PROGRAM CETVRTI
      INCLUDE 'E:\TR\F\CHAN.INC'

      INTEGER FI,PE6,PE6POLA,DB,CETFI
      INTEGER LEN,CETLEN,KOLPRIM
      PARAMETER ( FI = 5760,
     *          PE6 = 30030,
     *          CETFI = 23040,
     *          PE6POLA = 15015,
     *          LEN = 960960,
     *          CETLEN = 3843840,
     *          KOLPRIM = 3402,
     *          DB = 5760 )

      INTEGER UZP(FI),PRIM(KOLPRIM)
      INTEGER BLOK(PE6POLA),BAZA(CETFI)
      INTEGER POC,S,T,A,GRAN
      INTEGER BROJAC,GGRAN,OUT1

      COMMON  /ZAJEDNO/ UZP,PRIM

C Inicijalizacija podataka

C     DATA    (PRIM(I),I=0,11)/1,2,3,5,7,11,13,17,19,23,29,31/
C     DATA    (UZP(I),I=1,6)/1,17,19,23,29,31/

      OUT1=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUT1)



C Generisanje niza UZP

      CALL GENUZP


C Generisanje niza PRIM
C Primenjuje se metoda 'trial division'

      CALL GENPRIM


      POC=3*LEN

      DO 100 BROJAC=1,260
      C=POC+LEN
      GGRAN=SQRT(C)

      DO 10 I=1,PE6POLA
   10 BLOK(I)=0


      DO 40 J=7,BRP
      I=0
      K=PRIM(J)
      IF(K.GT.GGRAN) GOTO 50
      N=K-MOD(POC,K)
```

```
  GOTO 20
40 CONTINUE


50 L=1


  DO 60 I=1,DB
60 BAZA(I)=0


  DO 80 I=0,31
  DO 70 J=1,FI
  IF (.NOT.BTEST(BLOK((UZP(J)+1)/2),I) ) THEN
    K=L+J/32
    BAZA(K)=IBSET(BAZA(K),MOD(J,32))
  ENDIF
70 CONTINUE
  L=L+180
80 CONTINUE

  DO 90 I=1,DB
90 CALL F77_CHAN_OUT_WORD(BAZA(I),OUT1)

  POC=POC+CETLEN

100 CONTINUE
  STOP
  END
```

12. DODATAK B

PROGRAM ZA PROVERU KUREPINE HIPOTEZE O LEVOM FAKTORIJELU

- KONFIGURACIONI FAJL
- PROGRAM "PRVI", POSAO PRVOG PROCESORA
- PROGRAM "DRUGI" , POSAO DRUGOG PROCESORA
- PROGRAM "TREĆI" , POSAO TREĆEG PROCESORA
- PROGRAM "ČETVRTI", POSAO ČETVRTOG PROCESORA

```
! konfiguracioni fajl programa za proveru Kurepine hipoteze
! definisanje procesora
processor  host
processor  first
processor  second
processor  third
processor  fourth

! definisanje kanala koji povezuju procesore
wire  ?  host[0]   first[0]
wire  ?  first[1]  second[0]
wire  ?  first[2]  third[0]
wire  ?  first[3]  fourth[0]

! definisanje procesa
! informacije o broju ulazno-izlaznih kanala
! informacije o memoriji
task  afserver  ins=1   outs=1
task  filter    ins=2   outs=2   data=10k
task  prvi      ins=5   outs=5   data=2500k
task  drugi     ins=1   outs=1   data=2500k
task  treci     ins=1   outs=1   data=2500k
task  cetvrti   ins=1   outs=1   data=2500k

! dodeljivanje procesa procesorima
place  afserver   host
place  filter     first
place  prvi       first
place  drugi      second
place  treci      third
place  cetvrti    fourth

! definisanje kanala koji povezuju procese
connect  ?  filter[0]     afserver[0]
connect  ?  afserver[0]   filter[0]
connect  ?  filter[1]     prvi[1]
connect  ?  prvi[1]       filter[1]
connect  ?  drugi[0]      prvi[2]
connect  ?  prvi[2]       drugi[0]
connect  ?  treci[0]      prvi[3]
connect  ?  prvi[3]       treci[0]
connect  ?  cetvrti[0]    prvi[4]
connect  ?  prvi[4]       cetvrti[0]
```

```
      PROGRAM PRVI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000),Q,BR
      DOUBLE PRECISION KNOD,QD
      INTEGER TIMEB,TIMEE,TIME,D(48),KAN,I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER INSEC,OUTSEC,INTTHE,OUTTHE,INFOU,OUTFOU


      OPEN (23,FILE='OBAVESTENJA.K3' )

      INSEC=F77_CHAN_IN_PORT(2)
      OUTSEC=F77_CHAN_OUT_PORT(2)
      CALL F77_CHAN_IN_WORD(KAN,INSEC)
      INTHE=F77_CHAN_IN_PORT(3)
      OUTTHE=F77_CHAN_OUT_PORT(3)
      CALL F77_CHAN_IN_WORD(KAN,INTHE)
      INFOU=F77_CHAN_IN_PORT(4)
      OUTFOU=F77_CHAN_OUT_PORT(4)
      CALL F77_CHAN_IN_WORD(KAN,INFOU)



      WRITE(6,10)
      WRITE(23,10)
   10 FORMAT(1X,'KOMUNICIRAMO')

      WRITE(6,20)
      WRITE(23,20)
   20 FORMAT(1X,'UNESI INDEKS PRVOG I POSLEDNJEG PRIM-BROJA')

      READ(5,30) LP
      READ(5,30 ) LD
   30 FORMAT(I8)

      CALL F77_CHAN_OUT_WORD(LP,OUTSEC)

      CALL F77_CHAN_OUT_WORD(LD,OUTSEC)

      CALL F77_CHAN_OUT_WORD(LP,OUTTHE)
```

```
   IS= SQRT(A)+1
   DO 50 K=5,N
   IF(MOD(S,P(K)).EQ.0) GOTO 70
   IF(P(K).GT.IS)  GOTO 60
50 CONTINUE
60 N=N+1
   P(N)=S
   IF(N.GE.LD) GOTO 80
70 S=S+D(J)
   CONTINUE


80 WRITE(6,90)
   WRITE(23,90)
90 FORMAT(1X,'NAPRAVIO PROSTE')

   KAN=0
   BR=0
   DO 1001 J=LP,LD,4
   KAN=KAN+1
   Q=P(J)
```

```
   KNOD=Q
   QD=DBLE(Q)
   DO 100 IL=Q-2,1,-1
100 KNOD=DMOD(DBLE(1+IL*KNOD),QD)
   KNO=KNOD
   IF (KNO.EQ.0) THEN
     WRITE(6,110) P(J),1
     WRITE(23,110) P(J),1
110    FORMAT(1X,'KONTRAPRIMER:',I15,I15)
     GOTO 509
   ENDIF
   IF (KAN.EQ.100) THEN
     CALL F77_CHAN_IN_WORD(KAN,INSEC)
     IF (KAN.NE.0) THEN
        WRITE(6,70) KAN,2
        WRITE(23,70) KAN,2
        GOTO 509
     ENDIF

     CALL F77_CHAN_IN_WORD(KAN,INTHE)
     IF (KAN.NE.0) THEN
        WRITE(6,110) KAN,3
        WRITE(23,110) KAN,3
        GOTO 509
     ENDIF

     CALL F77_CHAN_IN_WORD(KAN,INFOU)
     IF (KAN.NE.0) THEN
```

```
      ENDIF


      CALL F77_CHAN_IN_WORD(KAN,INTHE)
      IF (KAN.NE.0) THEN
         WRITE(6,110) KAN,3
         WRITE(23,110) KAN,3
      ENDIF

      CALL F77_CHAN_IN_WORD(KAN,INFOU)
      IF (KAN.NE.0) THEN
         WRITE(6,110) KAN,4
         WRITE(23,110) KAN,4
      ENDIF
  140 CALL ICLOCK(TIMEE)
      TIME= TIMEE-TIMEB
      WRITE(6,150) P(LD),TIME
      WRITE(23,150) P(LD),TIME
  150 FORMAT(1X,'POSAO ZAVRSEN',/,I15,/,'  Vreme: ',I15)

      CLOSE(23)
      STOP
      END
```

```
      ENDIF


      CALL F77_CHAN_IN_WORD(KAN,INTHE)
      IF (KAN.NE.0) THEN
         WRITE(6,110) KAN,3
         WRITE(23,110) KAN,3
      ENDIF

      CALL F77_CHAN_IN_WORD(KAN,INFOU)
      IF (KAN.NE.0) THEN
         WRITE(6,110) KAN,4
         WRITE(23,110) KAN,4
      ENDIF
  140 CALL ICLOCK(TIMEE)
      TIME= TIMEE-TIMEB
      WRITE(6,150) P(LD),TIME
      WRITE(23,150) P(LD),TIME
  150 FORMAT(1X,'POSAO ZAVRSEN',/,I15,/,'  Vreme: ',I15)

      CLOSE(23)
      STOP
      END
```

```
      PROGRAM DRUGI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000)
      INTEGER KAN,KNO,Q
      DOUBLE PRECISION QD,KNOD
      INTEGER D(48),I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER OUTFIR,INFIR

      OUTFIR=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUTFIR)

      INFIR=F77_CHAN_IN_PORT(0)
      CALL F77_CHAN_IN_WORD(LP,INFIR)
      CALL F77_CHAN_IN_WORD(LD,INFIR)

      DO 30 I=1,100000
      DO 30 J=1,48
      A=S
      IS= SQRT(A)+1
      DO 10 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 30
      IF(P(K).GT.IS)  GOTO 20
   10 CONTINUE
   20 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 40
   30 S=S+D(J)
      CONTINUE

   40 KAN=0
      DO 60 J=LP+1,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 50 IL=Q-2,1,-1
   50 KNOD=DMOD(DBLE(1+IL*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
         CALL F77_CHAN_OUT_WORD(P(J),OUTFIR)
         STOP
      ENDIF
      IF (KAN.EQ.100) THEN
         CALL F77_CHAN_OUT_WORD(0,OUTFIR)
         KAN=0
      ENDIF
   60 CONTINUE
      CALL F77_CHAN_OUT_WORD(0,OUTFIR)
      STOP
      END
```

```
      PROGRAM TRECI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000)
      INTEGER KAN,KNO,Q
      DOUBLE PRECISION QD,KNOD
      INTEGER D(48),I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER OUTFIR,INFIR

      OUTFIR=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUTFIR)

      INFIR=F77_CHAN_IN_PORT(0)
      CALL F77_CHAN_IN_WORD(LP,INFIR)
      CALL F77_CHAN_IN_WORD(LD,INFIR)

      DO 30 I=1,100000
      DO 30 J=1,48
      A=S
      IS= SQRT(A)+1
      DO 10 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 30
      IF(P(K).GT.IS)  GOTO 20
   10 CONTINUE
   20 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 40
   30 S=S+D(J)
      CONTINUE

   40 KAN=0
      DO 60 J=LP+2,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 50 IL=Q-2,1,-1
   50 KNOD=DMOD(DBLE(1+IL*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
         CALL F77_CHAN_OUT_WORD(P(J),OUTFIR)
         STOP
      ENDIF
      IF (KAN.EQ.100) THEN
         CALL F77_CHAN_OUT_WORD(0,OUTFIR)
         KAN=0
      ENDIF
   60 CONTINUE
      CALL F77_CHAN_OUT_WORD(0,OUTFIR)
      STOP
      END
```

```
      PROGRAM CETVRTI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000)
      INTEGER KAN,KNO,Q
      DOUBLE PRECISION QD,KNOD
      INTEGER D(48),I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER OUTFIR,INFIR

      OUTFIR=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUTFIR)

      INFIR=F77_CHAN_IN_PORT(0)
      CALL F77_CHAN_IN_WORD(LP,INFIR)
      CALL F77_CHAN_IN_WORD(LD,INFIR)

      DO 30 I=1,100000
      DO 30 J=1,48
      A=S
      IS= SQRT(A)+1
      DO 10 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 30
      IF(P(K).GT.IS)  GOTO 20
   10 CONTINUE
   20 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 40
   30 S=S+D(J)
      CONTINUE

   40 KAN=0
      DO 60 J=LP+3,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 50 IL=Q-2,1,-1
   50 KNOD=DMOD(DBLE(1+IL*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
         CALL F77_CHAN_OUT_WORD(P(J),OUTFIR)
         STOP
      ENDIF
```

13. DODATAK C

PROGRAM ZA PROVERU HIPOTEZE O ALTERNIRAJUĆEM FAKTORIJELU

- KONFIGURACIONI FAJL
- PROGRAM "PRVI", POSAO PRVOG PROCESORA
- PROGRAM "DRUGI" , POSAO DRUGOG PROCESORA
- PROGRAM "TREĆI" , POSAO TREĆEG PROCESORA
- PROGRAM "ČETVRTI", POSAO ČETVRTOG PROCESORA

```
      PROGRAM PRVI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000),Q,BR
      DOUBLE PRECISION KNOD,QD
      INTEGER TIMEB,TIMEE,TIME,D(48),KAN,I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER INSEC,OUTSEC,INTTHE,OUTTHE,INFOU,OUTFOU


      OPEN (23,FILE='OBAVESTENJA.K3' )

      INSEC=F77_CHAN_IN_PORT(2)
      OUTSEC=F77_CHAN_OUT_PORT(2)
      CALL F77_CHAN_IN_WORD(KAN,INSEC)
      INTHE=F77_CHAN_IN_PORT(3)
      OUTTHE=F77_CHAN_OUT_PORT(3)
      CALL F77_CHAN_IN_WORD(KAN,INTHE)
      INFOU=F77_CHAN_IN_PORT(4)
      OUTFOU=F77_CHAN_OUT_PORT(4)
      CALL F77_CHAN_IN_WORD(KAN,INFOU)


      WRITE(6,10)
      WRITE(23,10)
   10 FORMAT(1X,'KOMUNICIRAMO')

      WRITE(6,20)
      WRITE(23,20)
   20 FORMAT(1X,'UNESI INDEKS PRVOG I POSLEDNJEG PRIM-BROJA')

      READ(5,30) LP
      READ(5,30 ) LD
   30 FORMAT(I8)

      CALL F77_CHAN_OUT_WORD(LP,OUTSEC)

      CALL F77_CHAN_OUT_WORD(LD,OUTSEC)

      CALL F77_CHAN_OUT_WORD(LP,OUTTHE)

      CALL F77_CHAN_OUT_WORD(LD,OUTTHE)

      CALL F77_CHAN_OUT_WORD(LP,OUTFOU)
```

```
      IS= SQRT(A)+1
      DO 50 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 70
      IF(P(K).GT.IS)  GOTO 60
   50 CONTINUE
   60 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 80
   70 S=S+D(J)
      CONTINUE


   80 WRITE(6,90)
      WRITE(23,90)
   90 FORMAT(1X,'NAPRAVIO PROSTE')

      KAN=0
      BR=0
      DO 1001 J=LP,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 100 IL=Q-2,1,-1
      KNOD=DMOD(DBLE(1+IL*KNOD),QD)
  100 KNOD=DMOD(DBLE(-1+(IL-1)*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
        WRITE(6,110) P(J),1
        WRITE(23,110) P(J),1
  110   FORMAT(1X,'KONTRAPRIMER:',I15,I15)
        GOTO 509
      ENDIF
      IF (KAN.EQ.100) THEN
        CALL F77_CHAN_IN_WORD(KAN,INSEC)
        IF (KAN.NE.0) THEN
           WRITE(6,70) KAN,2
           WRITE(23,70) KAN,2
           GOTO 509
        ENDIF

        CALL F77_CHAN_IN_WORD(KAN,INTHE)
        IF (KAN.NE.0) THEN
           WRITE(6,110) KAN,3
           WRITE(23,110) KAN,3
           GOTO 509
        ENDIF

        CALL F77_CHAN_IN_WORD(KAN,INFOU)
        IF (KAN.NE.0) THEN
           WRITE(6,110) KAN,4
           WRITE(23,110) KAN,4
           GOTO 509
        ENDIF
        BR=BR+400
        WRITE(6,120) BR
        WRITE(23,120) BR
  120   FORMAT(1X,'DO SADA URADJENO: ',I15)
        KAN=0
```

```
      WRITE(23,110) KAN,2
    ENDIF


    CALL F77_CHAN_IN_WORD(KAN,INTHE)
    IF (KAN.NE.0) THEN
      WRITE(6,110) KAN,3
      WRITE(23,110) KAN,3
    ENDIF

    CALL F77_CHAN_IN_WORD(KAN,INFOU)
    IF (KAN.NE.0) THEN
      WRITE(6,110) KAN,4
      WRITE(23,110) KAN,4
    ENDIF
140 CALL ICLOCK(TIMEE)
    TIME= TIMEE-TIMEB
    WRITE(6,150) P(LD),TIME
    WRITE(23,150) P(LD),TIME
150 FORMAT(1X,'POSAO ZAVRSEN',/,I15,/,'  Vreme: ',I15)


    CLOSE(23)
    STOP
    END
```

```
! konfiguracioni fajl programa za proveru
! - hipoteze o alternirajucem levom faktorijelu
! definisanje procesora
processor  host
processor  first
processor  second
processor  third
processor  fourth

! definisanje kanala koji povezuju procesore
wire  ?  host[0]   first[0]
wire  ?  first[1]  second[0]
wire  ?  first[2]  third[0]
wire  ?  first[3]  fourth[0]

! definisanje procesa
! informacije o broju ulazno-izlaznih kanala
! informacije o memoriji
task afserver  ins=1  outs=1
task filter   ins=2  outs=2  data=10k
task prvi   ins=5  outs=5  data=2500k
task drugi    ins=1  outs=1  data=2500k
task treci    ins=1  outs=1  data=2500k
task cetvrti  ins=1  outs=1  data=2500k

! dodeljivanje procesa procesorima
place  afserver   host
place  filter     first
place  prvi       first
place  drugi      second
place  treci      third
place  cetvrti    fourth

! definisanje kanala koji povezuju procese
connect  ?  filter[0]     afserver[0]
connect  ?  afserver[0]   filter[0]
connect  ?  filter[1]     prvi[1]
connect  ?  prvi[1]       filter[1]
connect  ?  drugi[0]      prvi[2]
connect  ?  prvi[2]       drugi[0]
connect  ?  treci[0]      prvi[3]
connect  ?  prvi[3]       treci[0]
connect  ?  cetvrti[0]    prvi[4]
connect  ?  prvi[4]       cetvrti[0]
```

```
      PROGRAM DRUGI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000)
      INTEGER KAN,KNO,Q
      DOUBLE PRECISION QD,KNOD
      INTEGER D(48),I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER OUTFIR,INFIR

      OUTFIR=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUTFIR)

      INFIR=F77_CHAN_IN_PORT(0)
      CALL F77_CHAN_IN_WORD(LP,INFIR)
      CALL F77_CHAN_IN_WORD(LD,INFIR)

      DO 30 I=1,100000
      DO 30 J=1,48
      A=S
      IS= SQRT(A)+1
      DO 10 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 30
      IF(P(K).GT.IS)  GOTO 20
   10 CONTINUE
   20 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 40
   30 S=S+D(J)
      CONTINUE

   40 KAN=0
      DO 60 J=LP+1,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 50 IL=Q-2,1,-1
      KNOD=DMOD(DBLE(1+IL*KNOD),QD)
   50 KNOD=DMOD(DBLE(-1+(IL-1)*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
         CALL F77_CHAN_OUT_WORD(P(J),OUTFIR)
         STOP
      ENDIF
      IF (KAN.EQ.100) THEN
         CALL F77_CHAN_OUT_WORD(0,OUTFIR)
         KAN=0
      ENDIF
   60 CONTINUE
      CALL F77_CHAN_OUT_WORD(0,OUTFIR)
      STOP
      END
```

```
      PROGRAM TRECI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000)
      INTEGER KAN,KNO,Q
      DOUBLE PRECISION QD,KNOD
      INTEGER D(48),I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER OUTFIR,INFIR

      OUTFIR=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUTFIR)

      INFIR=F77_CHAN_IN_PORT(0)
      CALL F77_CHAN_IN_WORD(LP,INFIR)
      CALL F77_CHAN_IN_WORD(LD,INFIR)

      DO 30 I=1,100000
      DO 30 J=1,48
      A=S
      IS= SQRT(A)+1
      DO 10 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 30
      IF(P(K).GT.IS)  GOTO 20
   10 CONTINUE
   20 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 40
   30 S=S+D(J)
      CONTINUE

   40 KAN=0
      DO 60 J=LP+2,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 50 IL=Q-2,1,-1
      KNOD=DMOD(DBLE(1+IL*KNOD),QD)
   50 KNOD=DMOD(DBLE(-1+(IL-1)*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
         CALL F77_CHAN_OUT_WORD(P(J),OUTFIR)
         STOP
      ENDIF
      IF (KAN.EQ.100) THEN
         CALL F77_CHAN_OUT_WORD(0,OUTFIR)
         KAN=0
      ENDIF
   60 CONTINUE
      CALL F77_CHAN_OUT_WORD(0,OUTFIR)
      STOP
      END
```

```
      PROGRAM CETVRTI
      INCLUDE 'C:\TF2V0\CHAN.INC'
      INTEGER S,P(100000)
      INTEGER KAN,KNO,Q
      DOUBLE PRECISION QD,KNOD
      INTEGER D(48),I,J,K,LP,LD,N,IS
      DATA (P(I),I=1,46)/2,3,5,7,11,13,17,19,23,29,
     *31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,
     *109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,
     *193,197,199/
      DATA D/10,2,4,2,4,6,2,6,4,2,4,6,6,2,6,4,2,6,4,6,8,4,2,4,
     *2,4,8,6,4,6,2,4,6,2,6,6,4,2,4,6,2,6,4,2,4,2,10,2/
      DATA S,N/211,46/
      INTEGER OUTFIR,INFIR

      OUTFIR=F77_CHAN_OUT_PORT(0)
      CALL F77_CHAN_OUT_WORD(1,OUTFIR)

      INFIR=F77_CHAN_IN_PORT(0)
      CALL F77_CHAN_IN_WORD(LP,INFIR)
      CALL F77_CHAN_IN_WORD(LD,INFIR)

      DO 30 I=1,100000
      DO 30 J=1,48
      A=S
      IS= SQRT(A)+1
      DO 10 K=5,N
      IF(MOD(S,P(K)).EQ.0) GOTO 30
      IF(P(K).GT.IS)  GOTO 20
   10 CONTINUE
   20 N=N+1
      P(N)=S
      IF(N.GE.LD) GOTO 40
   30 S=S+D(J)
      CONTINUE

   40 KAN=0
      DO 60 J=LP+3,LD,4
      KAN=KAN+1
      Q=P(J)
      KNOD=Q
      QD=DBLE(Q)
      DO 50 IL=Q-2,1,-1
      KNOD=DMOD(DBLE(1+IL*KNOD),QD)
   50 KNOD=DMOD(DBLE(-1+(IL-1)*KNOD),QD)
      KNO=KNOD
      IF (KNO.EQ.0) THEN
         CALL F77_CHAN_OUT_WORD(P(J),OUTFIR)
         STOP
      ENDIF
      IF (KAN.EQ.100) THEN
         CALL F77_CHAN_OUT_WORD(0,OUTFIR)
         KAN=0
      ENDIF
   60 CONTINUE
      CALL F77_CHAN_OUT_WORD(0,OUTFIR)
      STOP
      END
```