UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I INFORMATIKU

Implementacija AES algoritma tehnikom paralelnog programinja u C programskom jeziku

- Računarstvo visokih performansi -

Student:
Ivana Milutinović 50m/22

Profesor:
dr Lidija Fodor

Novi Sad, jul 2024.

Sadržaj

1. Uvod

Paralelno programiranje predstavlja ključnu tehniku za poboljšanje performansi programa različite namene, omogućavajući istovremeno izvršavanje više zadataka na više procesora ili jezgara. U svetu savremenih računara, gde se zahtevi za obradu podataka i brzo izvršavanje stalno povećavaju, paralelno programiranje postaje neophodno za efikasno korišćenje dostupnih resursa. Paralelni pristup omogućava bolje iskorišćavanje hardverskih mogućnosti, smanjuje vreme obrade i povećava efikasnost programa.

MPI (*Message Passing Interface*) biblioteka je standard za paralelno programiranje koji omogućava komunikaciju između procesa koji se izvode na različitim čvorovima u distribuiranom sistemu. Pored toga što implementacija ove biblioteke postoji i za druge programske jezike poput Jave, Pajtona i drugih, korišćenje MPI biblioteke u programskom jeziku C omogućava programerima da pišu programe na niskom nivou sa visokim performansama koji mogu iskoristiti snagu superračunara i klastera. Ova kombinacija omogućava postizanje izuzetno visokih performansi i skalabilnosti, što je ključno za rešavanje složenih problema u oblastima kao što su naučne simulacije, kriptografija i analiza velikih podataka.

2. Opis problema

Advanced Encryption Standard (*AES*) predstavlja varijantu **Rijndael** algoritma za enkripciju. Ovo je "*block cipher*" algoritam, što znači da su tekst i ključevi koji se koriste predstavljeni u formi matrice 4×4. Naime, u zavisnosti od velčine ključa, razlikuju se tri vrste ovog algoritma. U ovoj implementaciji, biće korišćeni ključevi veličine 128 bita, što je ujedno i standardna veličina cifarskog bloka koji se koristi u svim njegovim varijantama. Pored enkriptovanja, biće implementiran i obrnuti proces - dekripcija.

Sam proces enkripcije odvija se u 10 rundi, dok u slučaju varijanti sa ključem od 192 i 256 bita imamo 12, odnosno 14 rundi. AES je, za sada, neprobojan šifarski algoritam, koji je usvojen i od strane američke vlade, a u narednom delu ukratko će biti opisano njegovo izvršavanje.

Koraci algoritma:

1. Generisanje ključeva za svaku od rundi iz cifarskog ključa, na osnovu "AES key schedule"-a
2. Inicijalno dodavanje prvog ključa, primenom XOR operatora na bitove svakog bajta matrice stanja, pri čemu matrica stanja predstavlja matričnu reprezentaciju poruke koja se šifruje
3. Ponavljanje narednih koraka 9 puta (petlja):
    - Zamena svakog bajta matrice stanja pomoću S-Box tabele
    - Levo cirkularno šiftovanje elemenata po vrstama (nulta vrsta za nula mesta, prva vrsta za jedno mesto. . .)
    - Kreiranje novih kolona od postojećih, pojedinačno, primenom linearne operacije kojom se kombinuju sva 4 bajta
    - Dodavanje odgovarajućeg ključa za datu rundu
4. Finalna runda koja zajedno sa prethodnim daje 10 rundi ukupno:
    - Zamena svakog bajta matrice stanja, pomoću S-Box tabele
    - Levo cirkularno šiftovanje elemenata po vrstama (nulta vrsta za nula mesta, prva vrsta za jedno mesto. . .)
    - Dodavanje odgovarajućeg ključa za datu rundu

Proces dekripcije odvija se u obrnutom redosledu, pomoću procedura koje rade obrnuti posao od njihovih "parnjaka" namenjenih za enkripciju. Primerice, umesto šiftovanja elemenata ulevo, implementirana je operacija šiftovanja elemenata udesno. Slično, korak koji se tiče zamene bajtova, u ovom procesu vrši se korišćenjem inverzne S-Box tabele.

Pojašnjenje pojedinačnih koraka:

→ Dodavanje ključa matrici stanja, za posmatranu rundu, vrši se nad pojedinačnim bajtovima. Naime, svaki bajt sabira se sa odgovaraju´cim bajtom matrice kojom je predstavljen taj ključ. Obzirom da se ovde radi o operacijama nad bajtovima, tj. nad pojedinačnim bitovima, sabiranje se vrši korišćenjem XOR operatora

→ Zamena bajtova pomoću S-Box tabele radi se tako što se za svaki bajt koji posmatramo u matrici stanja, izračunava vrednost nižeg i višeg *nibla* (podatak veličine 4 bita) . Potom, pomoću višeg nibla određujemo vrstu, a pomoću nižeg kolonu S-Box matrice, u čijem preseku se nalazi bajt kojim zamenjujemo posmatrani.

→ Korak sa levim cirkularnim šiftovanjem vrši se tako što se elementi *i*-te vrste pomeraju za *i* pozicija ulevo. Dakle, elementi nulte vrste se neće pomerati, elementi prve će se pomeriti za jedno mesto, druge za dva mesta i poslednje za tri mesta ulevo.

→ Uz operaciju šiftovanja bajtova, korak koji se tiče kreiranja novih kolona linearnim operacijama najviše doprinosi difuziji algoritma. Difuzija u kontekstu cifarskih algoritama odnosi se na to da čak i male promene koje algoritam čini nad "plain-text"-om, bivaju oslikane drastičnim promenama na "cipher-text"-u koji se dobija na izlazu. Ovim se potencijalnom napadaču znatno otežava dešifrovanje. Izračunavanje novih kolona (bajtova) radi se na sledeći način:

- ❖ svaka kolona pojedinačno sačinjena je od četiri bajta, te od nje kreiramo vektor kolone sa koordinatama koje se odnose na pojedinačne bajtove.
- ❖ ovim vektorom kolone množimo odgovarajuću 4 × 4 matricu nad Galoa poljem, čiji elementi predstavljaju koeficijente linearne kombinacije. Svaka od četiri dobijene linearne kombinacije predstavlja novu vrednost pojedinačnih bajtova.
- ❖ od te četiri vrednosti kreiramo novu kolonu. Ovaj proces ponavlja se za svaku od kolona pojedinačno
- ❖ Inverzna operacija izvodi se na isti, analogan način, pri čemu je jedina razlika u koeficijentima od kojih je sačinjena matrica koja se množi.

Generisanje ključeva: AES koristi "key schedule" u generisanju ključeva. Naime, "key schedule" predstavlja, u opštem slučaju, neki cifarski algoritam koji na osnovu inicijalnog ključa izračunava pojedinačne ključeve za svaku od rundi. U ovom slučaju, izračunava se 10 ključeva, plus još jedan dodatni. Da bi sam postupak bio jasniji, definišu se sledeće oznake:

- N je dužina ključa izražena brojem reči dužine 32 bita (u ovom slučaju $N = 4$)
- $K_0, K_1, \ldots, K_{N-1}$ su 4 reči dužine 32 bita, za početni ključ
- R je broj potrebnih ključeva (ovde je $R = 11$)
- *rcon* je niz konstanti koje se koriste za računanje ključeva
- $W_0, W_1, \ldots, W_{4R-1}$ su reči dužine 32 bita od kojih će biti konstruisani svi ključevi
- "*rotWord*" je operacija levog cirkularnog šiftovanja po bajtovima
- "*subWord*" je primena operacije zamene svakog pojedinačnog bajta, na osnovu S-Box tabele

Sada, potrebno je izračunati 44 bajta od kojih će biti konstruisano 11 potrebnih ključeva. Ovaj postupak opisan je narednim "key scheduler'' algoritmom . Za $i = 0 \ldots 4R - 1$:

$$W_i = \begin{cases} K_i, & \text{if } i < N \\ W_{i-N} \oplus subWord(rotWord(W_{i-1})) \oplus rcon_{i/N}, & \text{if } i \geq N \wedge i \equiv 0 \pmod N \\ W_{i-N} \oplus subWord(W_{i-1}), & \text{if } i \geq N, N > 6 \wedge i \equiv 4 \pmod N \\ W_{i-N} \oplus W_{i-1}, & \text{otherwise.} \end{cases}$$

Za generisanje svakog pojedinačnog ključa koristi se po 4 elementa niza *W*. U ovoj implementaciji, koristi se samo jedna matrica za reprezentaciju ključa i ona se modifikuje (popunjava bajtovima novog ključa) u svakoj rundi. Vredi istaći da prve 4 reči niza *W*, na osnovu kojih se konstruiše prvi ključ, zapravo predstavljaju delove polaznog cifarskog ključa.

3. Sekvencijalna implementacija

U narednom delu opisane su osnovne metode koje se koriste za izvršavanje algoritma. Pored navedenih implementiran je i niz pomoćnih metoda koje, radi konciznosti, ovde nisu navedene.
*`n – MATRIX_SIZE`

- **`constructStateMatrix(char* msg)`** - za poruku *msg*, dužine 128 bita, koja se šifruje konstruiše matricu stanja tako što raspoređuje njene bajtove u odgovarajućem redosledu; ovo je matrica 4 × 4 i nad njom se vrše sve operacije šifrovanja
- **`stateMatrixToCipher(int state[n][n])`** - metoda koja na osnovu matrice stanja iz parametra *state* konstruiše cifarski tekst koji vraća kao string; numerička vrednost svakog bajta interpretira se kao karakter, pre dodavanja u string; ovo je rezultujući tekst
- **`shiftRows(int src[n][n])`** - radi levo cirkularno šiftovanje svake vrste matrice stanja koja je prosleđena parametrom *src*; njemu inverzni metod za dekripciju, `invShiftRows(int src[4][4], int rows, int cols)`, radi analogno, šiftovanje udesno
- **`subBytesStep(int src[n][n])`** - koristi se definisanom matricom **S_BOX** koja predstavlja S-Box tabelu, u svrhu izvršavanja koraka zamene bajtova matrice *src*; inverzni `invSubBytesStep(int src[4][4], int rows, int cols)` koristi se inverznom S-Box tabelom (matrica **INV_S_BOX**) u okviru procesa dekripcije
- **`mixColumns(int src[n][n], int inv)`** – izračunava nove vrednosti za svaku od kolona prosleđene matrice *src*, pomoću opisanih linearnih transformacija
- **`keySchedule()`** - metoda kojom se, prema "AES key schedule" algoritmu, izračunavaju 44 bajta potrebna za generisanje svih 11 ključeva; bajtovi se smeštaju u niz *W*
- **`aesKeyMatrix(int keyBytes[n])`** – parametar *keyBytes* predstavlja niz od 4 reči veličine 32 bita na osnovu kojih se konstruiše matrica ključa; ove reči dobavljaju se iz niza *W*; svaki od bajtova se smešta na odgovarajuće mesto u matrici *aesRoundKeyMatrix* koja predstavlja ključ u odgovarajućoj formi, spreman za korišćenje
- **`addRoundKey(int key[n][n], int state[n][n])`** – izvršava korak dodavanja ključa predstavljenog parametrom *key* u matricu stanja iz parametra *state*, pomoću XOR operacije nad bitovima

Implementacija algoritma enkripcije/dekripcije je testirana na primeru poruke koja se učitava iz fajla *message.txt* i smešta u promenljivu *plainText*. Kao rezultat izvršavanja programa, za proces enkripcije se u konzoli ispisuje učitana poruka za šifrovanje, njena matrica stanja i enkriptovani *cipherText* sa svojom finalnom matricom stanja. U procesu dekripcije, radi provere ispravnosti algoritma, ispisuje se matrica stanja i tekst dobijeni dekripcijom koji su identični početnoj matrici i učitanoj poruci.

Vreme potrebno za izvršavanje programa na lokalnom računaru je **0.015 sekundi**.

4. Opis paralelizacije

U prethodnoj dve sekcije su dati opis problema i postupak njegovog rešavanja, a zatim i sekvencijalna implementacija istog na osnovu koje je dalje rađena paralelizacija. Primena paralelizacije biće objašnjena u 2 dela: enkripcije i dekripcije.

Enkripcija

Procesi rade zajedno kako bi enkriptovali tekstualni ulaz. Prvo, inicijalizuju se konstantne vrednosti i konvertuje glavni ključ u niz od 4 32-bitne reči pomoću funkcija *rconsInit()* i *cipherTo4WordKey().* Nakon toga, generišu se ključevi runde za AES enkripciju pomoću for petlje i funkcije *keySchedule().*

U fazi pripreme, tekst se učitava iz fajla *message.txt* na procesu sa rangom 0 (master), prikazuje njegova veličina i matrica stanja originalne poruke. Veličina teksta se zatim šalje svim procesima korišćenjem **MPI_Bcast(),** nakon čega se tekst deli među procesima koristeći **MPI_Scatter().** Svaki proces konstruiše svoje početno stanje matrice koristeći lokalni deo teksta i funkciju *constructStateMatrix(),* nakon čega dodaje inicijalni ključ inicijalne runde na stanje matrice sa *addRoundKey().*

Runde enkripcije se realizuju između procesa spram podela rundi. Svaki proces izvršava određen broj rundi (*subBytesStep, shiftRows, i opciono mixColumns*) koristeći svoje lokalno stanje matrice i lokalne ključeve runde. Na kraju poslednje runde, dodaje se poslednji ključ runde na stanje matrice, nakon čega se rezultati prikupljaju sa *MPI_Gather()* i formira konačni kriptovani tekst.

Nakon enkripcije, na master procesu se prikazuju finalno stanje matrice i kriptovani tekst.

Dekripcija

Inicijalno, master proces označava početak dekripcije navođenjem adekvatne poruke.

Nakon toga, kriptovani tekst se deli među sve procese pomoću *MPI_Scatter(),* pri čemu svaki proces dobija deo kriptovanog teksta *cipherText*. Lokalni deo kriptovanog teksta se zatim konvertuje u početno stanje matrice *stateMatrixI* korišćenjem funkcije *constructStateMatrix().*

Dekripcija se vrši kroz petlju koja prolazi kroz sve runde, počevši od poslednje ka prvoj (round se kreće od NUM_ROUNDS - 1 do 0). Za svaku rundu, procesi uzimaju inverzne ključeve runde prema njihovom redosledu i primenjuju ih na matricu stanja koristeći funkcije *aesKeyMatrix()* i *addRoundKey().* U svakoj iteraciji, osim poslednje, takođe se izvršava i obrnuti *mixColumns()* korak, a zatim se primenjuju obrnuti *invShiftRows()* i *invSubBytesStep()* koraci.

Nakon završetka svih rundi dekripcije, rezultati se prikupljaju sa *MPI_Gather()* samo na procesu sa rangom 0. Ovde se formira dekriptovani tekst *decryptedTextF* koji se dalje prikazuje u obliku stanja matrice i konačnog dekriptovanog rezultata pomoću funkcija *constructStateMatrix()* i *stateMatrixToCipher().* Na kraju, oslobađaju se svi alocirani resursi kako bi se osigurala ispravna upotreba memorije.

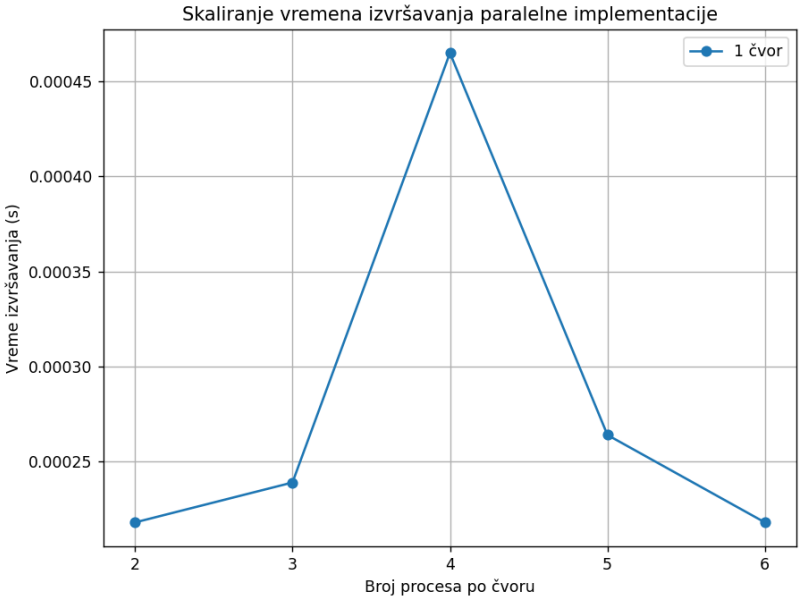
6. Rezultati

U okviru seminarskog rada, rezultati testiranja uključuju tabelarni prikaz vremena izvršavanja implementiranog AES algoritma za enkripciju i dekripciju. Ovi rezultati pružaju detaljan uvid u performanse algoritma pri različitom broju procesa, omogućavajući direktno poređenje između serijske i paralelne implementacije.

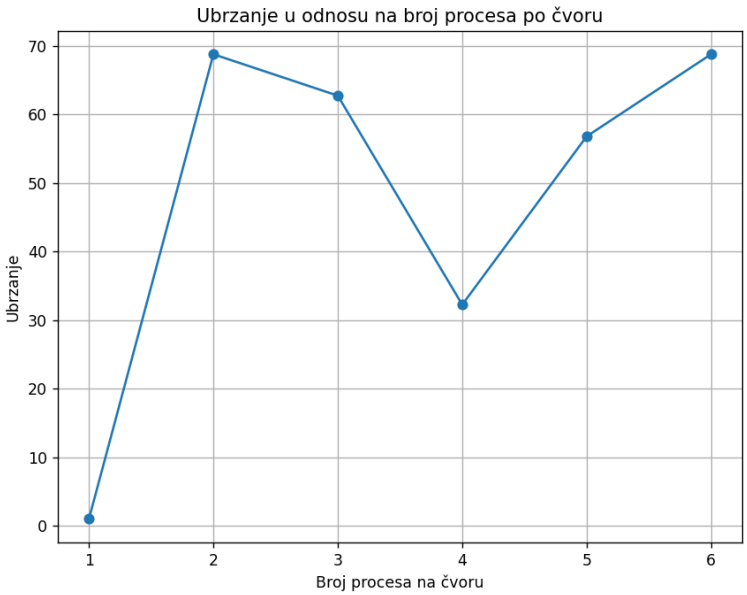
Tabela sadrži informacije o vremenima izvršavanja za različite brojeve procesa, što pomaže u proceni skalabilnosti algoritma i identifikaciji potencijalnih uskih grla ili efekata ubrzanja. Pored toga, grafik skaliranja i ubrzanja vizualno prikazuje kako se vreme izvršavanja menja sa porastom broja procesa, pružajući uvid u efikasnost paralelne implementacije i potencijalne granice ubrzanja u zavisnosti od problema i resursa na raspolaganju.

Broj cvorova	Broj procesa po cvoru	Vreme
1	1	0.015000
1	2	0.000218
1	3	0.000239
1	4	0.000465
1	5	0.000264
1	6	0.000218

Slika 1: Tabela vremena izvršavanja



Slika 2: Grafik skaliranja



Slika 3: Grafik ubrzanja

7. Zaključak

U izradi projekta fokus je bio na paralelizaciji AES algoritma za enkripciju i dekripciju koristeći MPI biblioteku za paralelno programiranje u C programskom jeziku, a sve sa ciljem efikasnog rukovanja porukama od 128 bita preko više procesa. Kroz testiranje i analizu, uočena su značajna poboljšanja u performansama i skaliranju, posebno kada su za broj procesa na čvoru korišćene vrednosti 2 i 6. Rezultati su pokazali da paralelno računanje može značajno ubrzati kriptografske operacije, čineći ih pogodnijim za aplikacije u realnom vremenu i za obradu velikih količina podataka.

Ubrzanje postignuto sa 2 i 6 procesa bilo je značajno. Distribuiranjem radnog opterećenja među više procesa, efektivno je smanjeno vreme izvršavanja, naglašavajući prednosti paralelnog procesiranja u kriptografskim zadacima. Ovo ubrzanje pokazuje da su troškovi upravljanja više procesa nadmašeni dobitkom u računarskoj efikasnosti, posebno za zadatke koji zahtevaju intenzivne proračune kao što su enkripcija i dekripcija. Implementacija je iskoristila sposobnosti MPI za efikasno slanje ključeva i deljenje podataka, osiguravajući ravnomerno opterećenje među procesima.

Testovi skalabilnosti dodatno su potvrdili efikasnost ovog pristupa. Algoritam za enkripciju i se dobro skaliralirao sa povećanjem broja procesa, održavajući konzistentno poboljšanje performansi. Ova skalabilnost je ključna za aplikacije koje zahtevaju obradu velikih količina podataka ili za sisteme koji rade u distribuiranim okruženjima. Pozitivni rezultati za 2 i 6 procesa sugerišu da implementacija može da se nosi sa različitim opterećenjima i da se prilagodi različitim računarskim resursima, čineći je univerzalnim rešenjem za bezbednu obradu podataka.

Obzirom da ovaj projekat demonstrira potencijal paralelnog procesiranja u poboljšanju performansi kriptografskih operacija, postignuto ubrzanje i skalabilnost naglašavaju važnost optimizacije algoritama za enkripciju i dekripciju za paralelno izvršavanje, otvarajući put ka efikasnijem i sigurnijem rukovanju podacima u različitim aplikacijama.