Multiprocesorski sistemi

Uvod

Matija Dodović, Marko Mišić

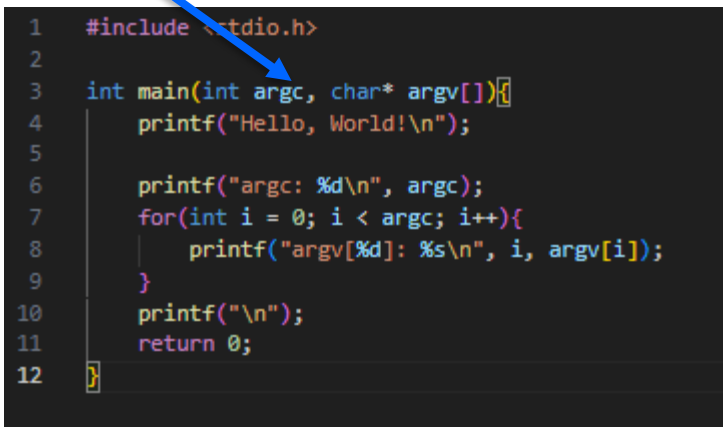
13S114MUPS, 13E114MUPS, 13M114MUPS

2023/2024.

Uvod u Linux operativni sistem

C program

- Parametri ulazne „main“ funkcije
  - Preko njih se može zadati složenost problema



```c
#include <stdio.h>

int main(int argc, char* argv[]){
    printf("Hello, World!\n");

    printf("argc: %d\n", argc);
    for(int i = 0; i < argc; i++){
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    printf("\n");
    return 0;
}
```

Prevodjenje: `gcc -o hello_world -O3 hello_world.c`

`Naziv egzekutabilnog programa`

`Stepen optimizacije`

C program

- Parametri ulazne „main“ funkcije
  - Preko njih se može zadati složenost problema

```c
#include <stdio.h>

int main(int argc, char* argv[]){
    printf("Hello, World!\n");

    printf("argc: %d\n", argc);
    for(int i = 0; i < argc; i++){
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    printf("\n");
    return 0;
}
```

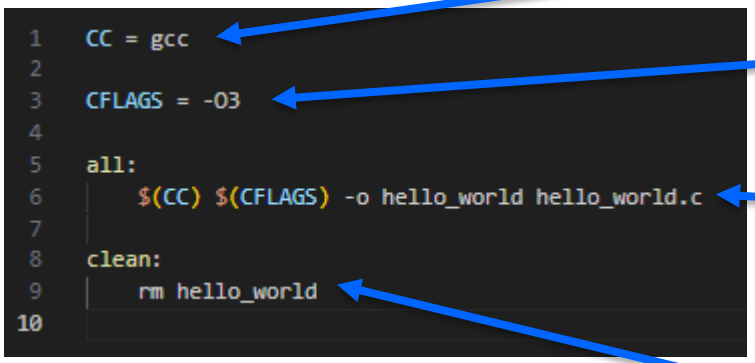
Pokretanje: `./hello_world param1 param2 param3`

Ispis:

```
Hello, World!
argc: 4
argv[0]: ./hello_world
argv[1]: param1
argv[2]: param2
argv[3]: param3
```

Makefile i basch

- Jednostavan Makefile



```
1  CC = gcc
2
3  CFLAGS = -O3
4
5  all:
6      $(CC) $(CFLAGS) -o hello_world hello_world.c
7
8  clean:
9      rm hello_world
10
```

Kompajler

Flegovi

Pravilo za prevodjenje

Pravilo za „čišćenje“

Preporuka imati ga uvek

Pokretanjem pravila all*: `make all`

izvršava: `gcc -O3 -o hello_world hello_world.c`

* Pošto je to prvo pravilo, može se pokrenuti i samo sa `make`

Makefile i basch skripta

- Basch skripta služi za izvršavanje sekvenci komandi

```
1   #!/bin/bash
2   ./hello_world param1 param2 param3
3   ./hello_world 1 2 3
```

Pokretanje: `./run.sh`

Ispis:

```
Hello, World!
argc: 4
argv[0]: ./hello_world
argv[1]: param1
argv[2]: param2
argv[3]: param3

Hello, World!
argc: 4
argv[0]: ./hello_world
argv[1]: 1
argv[2]: 2
argv[3]: 3
```

```c
#include <stdio.h>

int main(int argc, char* argv[]){
    printf("Hello, World!\n");

    printf("argc: %d\n", argc);
    for(int i = 0; i < argc; i++){
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    printf("\n");
    return 0;
}
```

Merenje vremena

```c
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[]) {
    int i, j;
    int n = 0;
    int len = argc > 1 ? atoi(argv[1]) : 1;









    for (i = 0; i < 1000 * len; i++) {
        for (j = 0; j < 1000000; j++) {
            n = n + 1;
        }
    }


    printf("len = %d, n = %d\n", len, n);


    return 0;
}
```

Merenje vremena

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

int main(int argc, char *argv[]) {
    int i, j;
    int n = 0;
    int len = argc > 1 ? atoi(argv[1]) : 1;

    struct timeval start, end;
    long seconds, useconds;
    double elapsed_time;

    // Start timing
    gettimeofday(&start, NULL);

    for (i = 0; i < 1000 * len; i++) {
        for (j = 0; j < 1000000; j++) {
            n = n + 1;
        }
    }

    // End timing
    gettimeofday(&end, NULL);

    // Calculate elapsed time in microseconds
    seconds  = end.tv_sec  - start.tv_sec; // seconds
    useconds = end.tv_usec - start.tv_usec; // microseconds
    elapsed_time = ((seconds) * 1000.0 + useconds/1000.0);  // Convert to milliseconds

    printf("len = %d, n = %d\n", len, n);
    printf("Elapsed time: %.3f ms\n", elapsed_time);

    return 0;
}
```

Pokretanje:

```
#!/bin/bash
./intensive_loop 1
./intensive_loop 5
./intensive_loop 10
```

Ispis:

```
len = 1, n = 1000000000
Elapsed time: 661.826 ms
len = 5, n = 705032704
Elapsed time: 3148.606 ms
len = 10, n = 1410065408
Elapsed time: 6283.174 ms
```

Primeri paralelnih programa

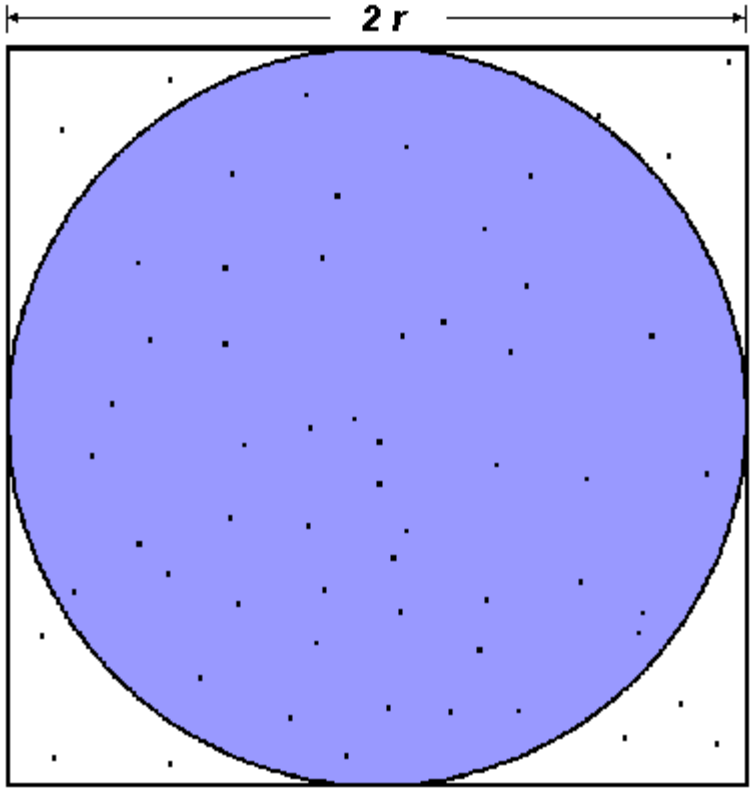
Primeri paralelnih programa

- Računanje broja PI
- Obrada 2D niza (array processing)
- *Simple heat equation* problem
- *1-D Wave Equation* problem

Računanje broja PI (1)

- Jedan jednostavan algoritam za približno računanje broja PI
  - Upisati krug u kvadrat
  - Slučajno generisati tačke u datom kvadratu
  - Utvrditi broj tačaka u kvadratu koje su istovremeno i u krugu
  - PI se može približno izračunati po formuli
    - PI = 4.0 * (broj tačaka u krugu)/(ukupan broj tačaka)
  - Što se više tačaka generiše, aproksimacija je bolja

Računanje broja PI (2)



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

Računanje broja PI (3)

- Sekvencijalni kod
  - npoints = 10000
    circle_count = 0
    do j = 1,npoints
      generate 2 random numbers between 0 and 1
      xcoordinate = random1 ; ycoordinate = random2
      if (xcoordinate, ycoordinate) inside circle
      then circle_count = circle_count + 1
    end do

    PI = 4.0*circle_count/npoints

Računanje broja PI (4)

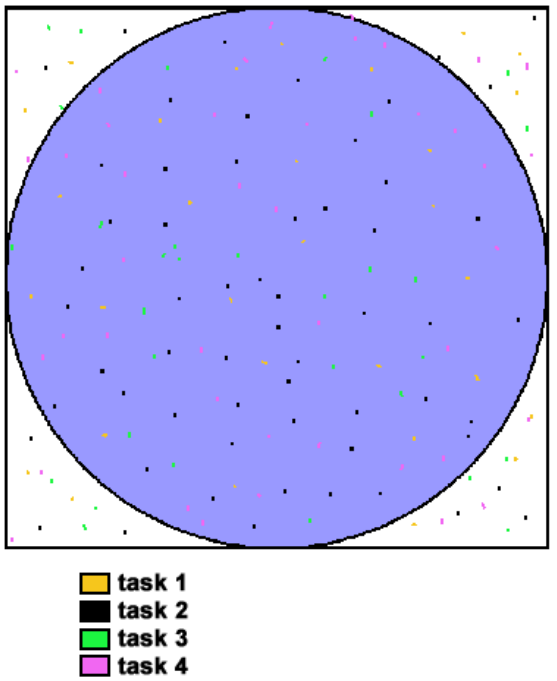
- Većina vremena u izvršavanju ovog koda bi bila potrošena na izvršavanje petlje
- Još jedan od *embarrassingly parallel* problema
  - Računski zahtevan, minimalna komunikacija i I/O
- Paralelna strategija
  - Razbiti petlju na delove koje će izvršavati zadaci
  - Svaki zadatak obavlja svoj deo računanja petlje
  - Ne postoje zavisnosti po podacima, pa nema ni komunikacije među zadacima
  - Koristi se SPMD model
    - Jedan zadatak se ponaša kao master i sakuplja rezultate

Računanje broja PI (5)

- npoints = 10000
  circle_count = 0
  p = number of tasks
  num = npoints/p
  find out if I am MASTER or WORKER
  do j = 1,num
    generate 2 random numbers between 0 and 1
    xcoordinate = random1 ; ycoordinate = random2
    if (xcoordinate, ycoordinate) inside circle
    then circle_count = circle_count + 1
  end do

Računanje broja PI (6)

- if I am MASTER
  receive from WORKERS their circle_counts
  compute PI (use MASTER and WORKER calculations)
  else if I am WORKER
  send to MASTER circle_count
  endif

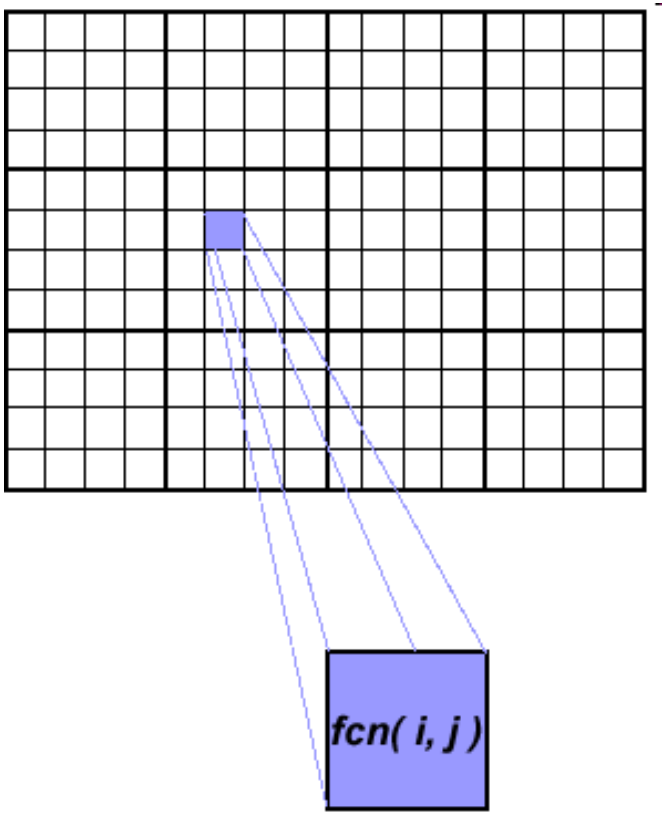


Obrada 2D niza (1)

- Radi se obrada nad elementima 2D niza, a računski posao nad svakim elementom je nezavisan u odnosu na druge
- Sekvencijalni program računa jedan element u jednom trenutku idući sekvencijalno kroz niz
- Ceo problem je računski veoma zahtevan
- Problem je tzv. *embarrassingly parallel*, jer je izračunavanje jednog elementa nezavisno od drugih elemenata

Obrada 2D niza (2)

- Sekvencijalno rešenje
  - do j = 1,n
    do i = 1,n
      a(i,j) = fcn(i,j)
    end do
    end do

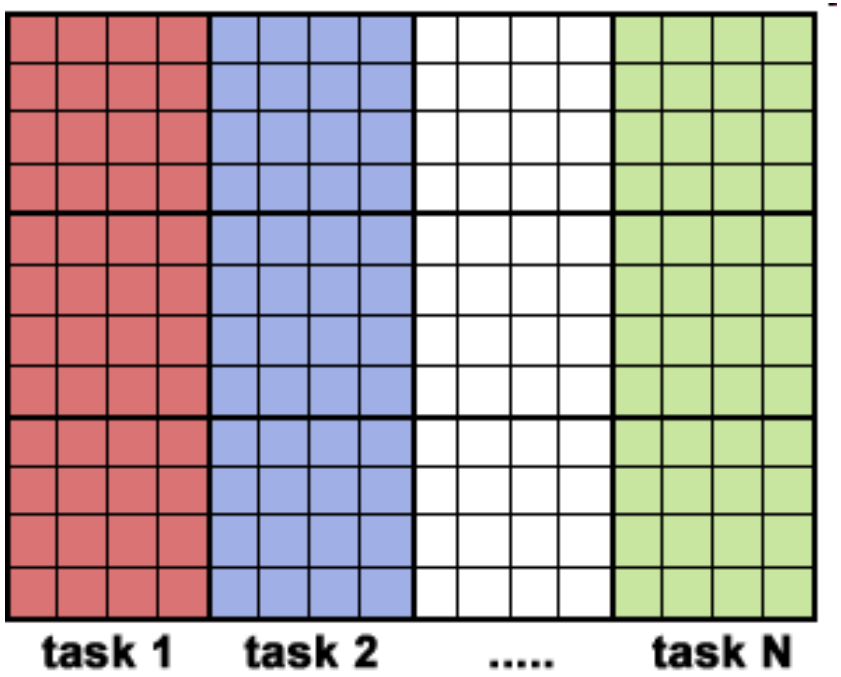


Obrada 2D niza (3)

- Paralelno rešenje
  - Elementi niza se podele tako da svaki procesor dobije jednu podmatricu na obradu
  - Prilikom izbora šeme za distribuciju podmatrica, gleda se da maksimizuje korišćenje keš memorije
  - Komunikacija između zadataka nije potrebna zbog karakteristika problema
  - Nakon što se matrica distribuira, svaki zadatak izvršava deo petlje, u zavisnosti koji deo podataka je dobio na obradu

Obrada 2D niza (4)

- Primer koda
  - do j = mystart, myend
    do i = 1,n
      a(i,j) = fcn(i,j)
    end do
    end do



Obrada 2D niza (5)

- Jedno moguće rešenje
  - Implementirati SPMD model
  - Master zadatak inicijalizuje niz,
    šalje podatke radnicima i prima rezultate
  - Zadatak-radnik prima podatke,
    izvršava svoj deo računanja i šalje rezultate masteru
  - Ovakvo rešenje se može oblikovati
    prema pseudo-kodu datom na sledećoj stranici

Obrada 2D niza (6)

- find out if I am MASTER or WORKER

  if I am MASTER

    initialize the array
    send each WORKER info on part of array it owns
    send each WORKER its portion of initial array

  receive from each WORKER results

Obrada 2D niza (7)

- else if I am WORKER
  receive from MASTER info on part of array I own
  receive from MASTER my portion of initial array

  # calculate my portion of array
  do j = my first column,my last column
  do i = 1,n
    a(i,j) = fcn(i,j)
  end do
  end do

  send MASTER results

  endif

Obrada 2D niza (8)

- Prethodno rešenje prikazuje statičko balansiranje opterećenja
  - Svaki zadatak dobija podjednaku količinu posla
  - Ukoliko su procesori različitih karakteristika, može doći do gubitka procesorskog vremena u čekanju
  - Statičko balansiranje opterećenje obično nije problem ukoliko zadaci izvršavaju podjednak posao na identičnim mašinama
  - Ukoliko postoji problem balansiranosti opterećenja, može se koristiti šema bazena poslova (*pool of tasks*) za rešavanje problema

Obrada 2D niza (9)

- Depo poslova (*task pool*)
  - Postoje dve vrste procesa: master i proces-radnik
- Master proces
  - Sadrži i održava depo poslova koje radnici treba da obave
  - Šalje radniku posao kada to ovaj zahteva
  - Sakuplja rezultate od procesa-radnika
  - Po potrebi, i master može da se uključi u obavljanje posla

Obrada 2D niza (10)

- Proces-radnik
  - Dobija poslove od master procesa
  - Izvršava računanje
  - Šalje rezultate master procesu
- Proces-radnik ne zna pre vremena izvršavanja koji deo posla će obaviti i koliko poslova
- Dinamičko balansiranje opterećenja se dešava u vreme izvršavanja
  - Brži procesi-radnici dobijaju više posla da obave

Obrada 2D niza (11)

- find out if I am MASTER or WORKER

  if I am MASTER

    do until no more jobs
      send to WORKER next job
      receive results from WORKER
    end do

    tell WORKER no more jobs

Obrada 2D niza (12)

- else if I am WORKER

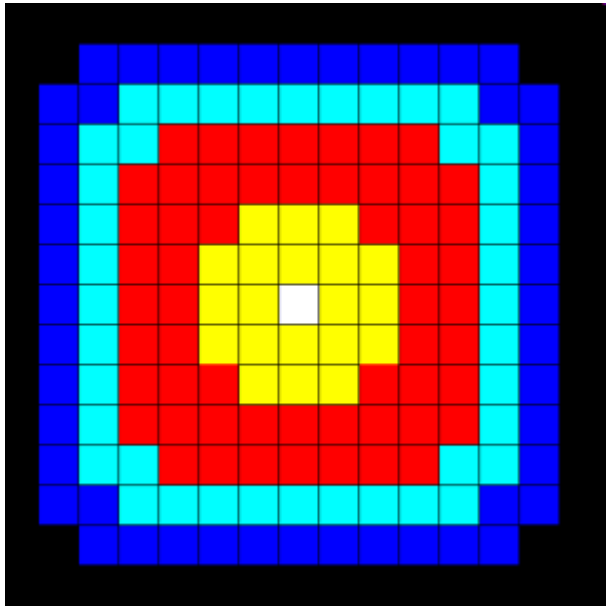
```
  do until no more jobs

    receive from MASTER next job
       calculate array element: a(i,j) = fcn(i,j)
    send results to MASTER

  end do
endif
```

Obrada 2D niza (13)

- Depo poslova
  - U datom primeru, svaki zadatak je smatrao jedan element niza za poseban posao (*job*)
  - Primer *fine-grain* implementacije
  - *Fine-grain* implementacije podrazumevaju više režijskog vremena za komunikaciju, kako bi se smanjilo rasipanje procesorskog vremena
  - Optimalnije rešenje bi bilo da svaki posao koji se šalje sadrži veću količinu računskog posla
    - Količina posla zavisi od konkretnog problema

Simple heat equation problem (1)

- Većina paralelnih problema zahteva komunikaciju među zadacima
  - Postoji čitava klasa problema koji zahtevaju komunikaciju sa "susednim" zadacima
- *Simple heat equation* problem opisuje promenu temperature na nekom prostoru kroz vreme, kada su poznati početna temperatura i granični uslovi



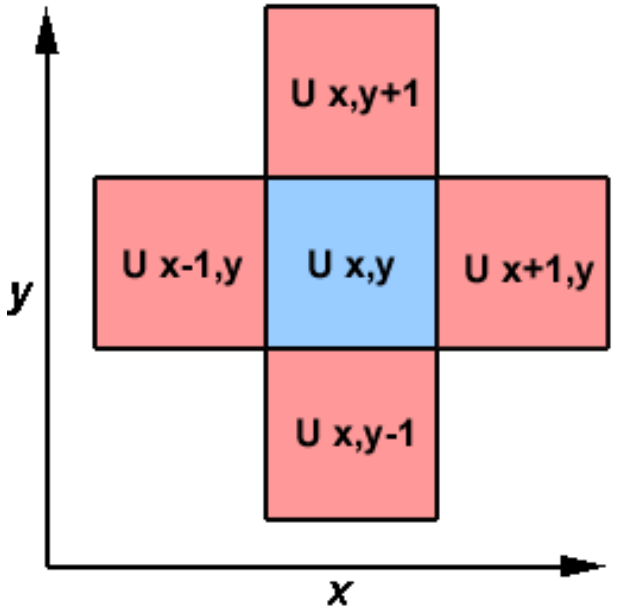
Simple heat equation problem (2)

- Postoji konačna šema za rešavanje ovog problema numerički na kvadratnoj površini
  - Početna temperatura na granicama je nula, a u sredini je visoka
  - Granična temperatura se uvek drži na nuli
  - Kvadratna površina se deli na mrežu manjih kvadrata
    - Predstavljaju se matricom čiji elementi predstavljaju temperaturu odgovarajućeg polja u kvadratu
    - Koristi se *time stepping* algoritam – izračunaju se temperature svih polja u datom vremenskom trenutku, pa se prelazi na sledeći vremenski trenutak

Simple heat equation problem (3)

- Izračunavanje vrednosti jednog elementa zavisi od vrednosti susednih elemenata

$$U_{x,y} = U_{x,y} + C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) + C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$



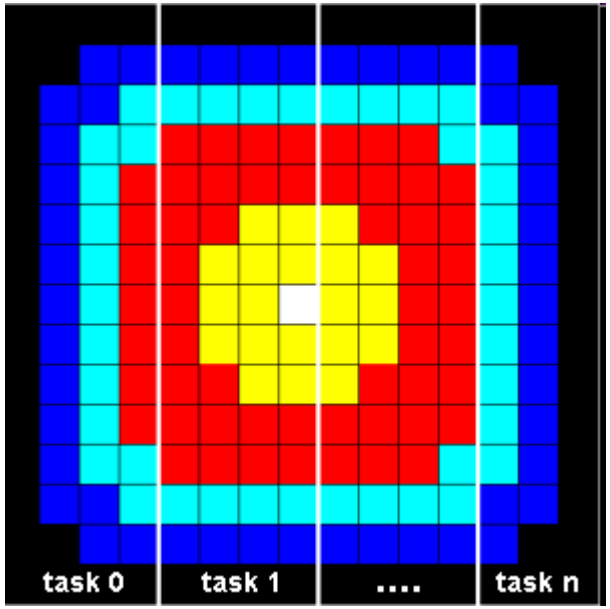
Simple heat equation problem (4)

- Sekvencijalno rešenje
  - do iy = 2, ny – 1
    do ix = 2, nx – 1

```
u2(ix, iy) =
  u1(ix, iy)  +
    cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2.*u1(ix,iy)) +
    cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2.*u1(ix,iy))

end do
end do
```

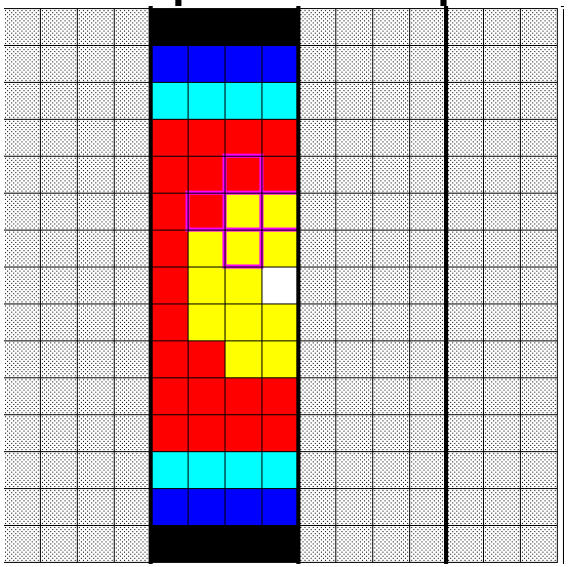
Simple heat equation problem (5)

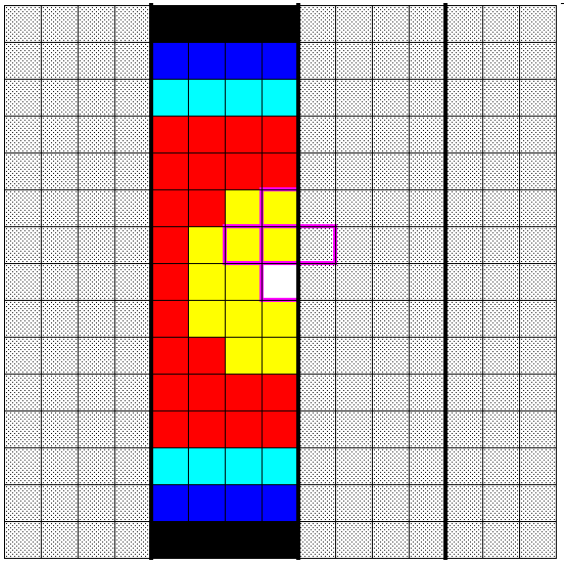
- Jedno paralelno rešenje
  - Implementirati SPMD model
  - Cela matrica se deli na i svaki zadatak dobija deo posla
  - Master proces šalje podatke procesima-radnicima, proverava konvergenciju rešenja i sakuplja rezultate
  - Procesi-radnici računaju podatke, komuniciraju po potrebi sa susednim procesima i šalju rezultate masteru



Simple heat equation problem (6)

- Postoje zavisnosti po podacima
  - Unutrašnji elementi koji pripadaju zadatku ne zavise od drugih zadataka
  - Granični elementi zahtevaju komunikaciju, jer su im potrebni podaci iz susednih zadataka





Simple heat equation problem (7)

- find out if I am MASTER or WORKER
  if I am MASTER
    initialize array
    send each WORKER starting info and subarray
    do until all WORKERS converge
      gather from all WORKERS convergence data
      broadcast to all WORKERS convergence signal
    end do
    receive results from each WORKER

Simple heat equation problem (8)

- else if I am WORKER
  receive from MASTER starting info and subarray
  do until solution converged
    update time
    send neighbors my border info
    receive from neighbors their border info
      update my portion of solution array
    determine if my solution has converged
      send MASTER convergence data
      receive from MASTER convergence signal
  end do
  send MASTER results
  endif

Simple heat equation problem (9)

- U prethodnom rešenju je pretpostavljeno da je komunikacija između zadataka blokirajuća
  - Blokirajuća komunikacija podrazumeva da zadatak ne nastavlja dalje sa izvršavanjem dok god primalac ne preuzme podatke
- U prethodnom rešenju susedni zadaci razmene podatke, a zatim svaki od njih izračuna svoj deo matrice
- Vreme računanja se može često smanjiti korišćenjem neblokirajuće komunikacije
  - Neblokirajuća komunikacija dozvoljava da se nastavi sa radom dok komunikacija još traje
  - Svaki zadatak može da izračuna svoje unutrašnje elemente dok se vrši razmena podataka o graničnim elementima, a zatim da izračuna vrednost svojih graničnih elemenata

Simple heat equation problem (10)

- find out if I am MASTER or WORKER
  if I am MASTER
    initialize array
    send each WORKER starting info and subarray
    do until all WORKERS converge
      gather from all WORKERS convergence data
      broadcast to all WORKERS convergence signal
    end do
    receive results from each WORKER

Simple heat equation problem (11)

- else if I am WORKER
  receive from MASTER starting info and subarray
  do until solution converged
    update time
    non-blocking send neighbors my border info
    non-blocking receive neighbors border info
    update interior of my portion of solution array
    wait for non-blocking communication complete
    update border of my portion of solution array

Simple heat equation problem (12)

- determine if my solution has converged
  send MASTER convergence data
  receive from MASTER convergence signal

  end do
  send MASTER results
  endif

1-D Wave Equation problem (1)

- Treba izračunati amplitudu talasa
  duž uniformne, vibrirajuće žice
  nakon odgovarajućeg vremenskog trenutka
- Računanje uključuje
  - Vrednost amplitude talasa na *y* osi
  - Poziciju tačke koja se prati duž *x* ose
  - Presečne tačke talasa duž žice
  - Promenu amplitude talasa
    u diskretnim vremenskim intervalima

1-D Wave Equation problem (2)

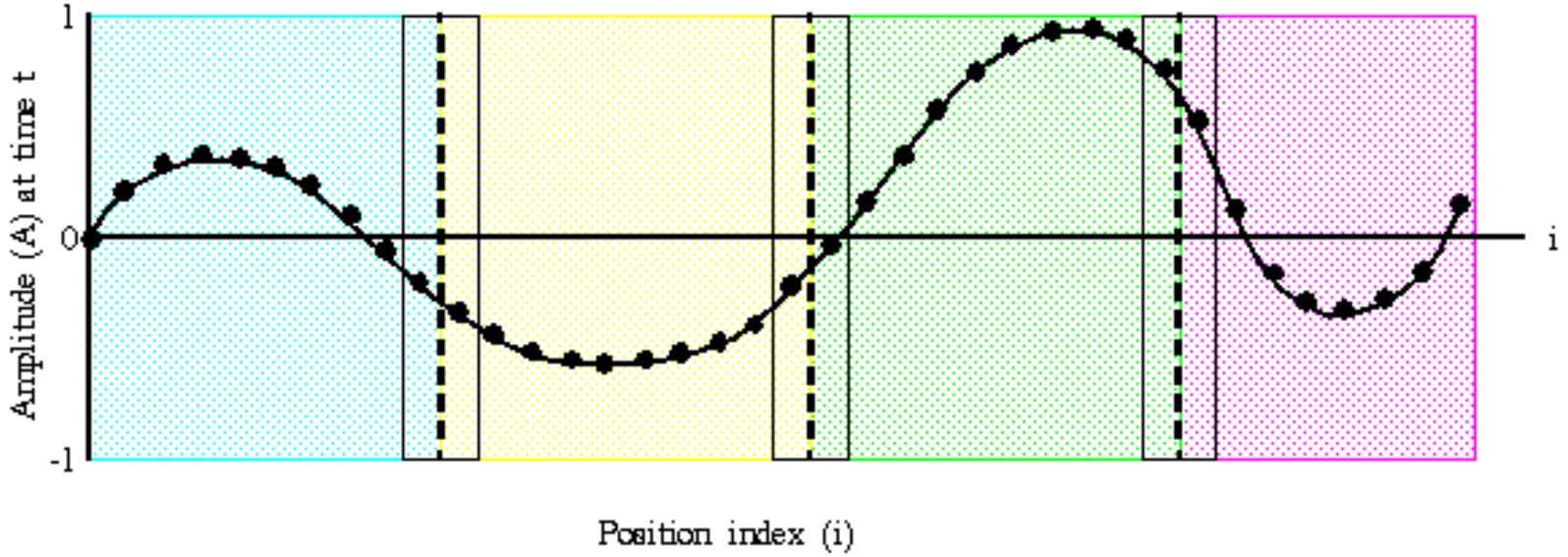
- Amplituda talasa u jednoj tački se izračunava po formuli
  - A(i,t+1) =
    (2.0 * A(i,t)) - A(i,t-1)+ (c * (A(i-1,t) - (2.0 * A(i,t)) + A(i+1,t)))
- Amplituda talasa u nekoj tački zavisi
  i od susednih tačaka
  i od amplitude tačke u prethodnim vremenskim trenucima
  - Paralelno rešenje će zahtevati komunikaciju

1-D Wave Equation problem (3)

- Moguće rešenje
  - Implementirati SPMD model
  - Talas predstaviti pomoću niza amplituda u izabranim tačkama
  - Niz amplituda podjednako podeliti i poslati podnizove svim zadacima koji rade
  - Obrada svake tačke zahteva podjednako vreme, tako da svi zadaci treba da dobiju jednak broj tačaka
  - Obratiti pažnju da zadaci dobiju što veći kontinualni deo niza, kako bi se izbegla nepotrebna komunikacija

1-D Wave Equation problem (4)

- Komunikacija potrebna na granicama podnizova
  - "susedni" zadaci će morati da razmenjuju podatke



1-D Wave Equation problem (5)

- find out number of tasks and task identities

```
#Identify left and right neighbors
left_neighbor = mytaskid – 1
right_neighbor = mytaskid +1
if mytaskid = first then left_neigbor = last
if mytaskid = last then right_neighbor = first

find out if I am MASTER or WORKER
if I am MASTER
  initialize array
  send each WORKER starting info and subarray
else if I am WORKER
  receive starting info and subarray from MASTER
endif
```

1-D Wave Equation problem (6)

- #Update values for each point along string

```
#In this example the master participates in calculations
do t = 1, nsteps
  send left endpoint to left neighbor
  receive left endpoint from right neighbor
  send right endpoint to right neighbor
  receive right endpoint from left neighbor

#Update points along line
  do i = 1, npoints
    newval(i) = (2.0 * values(i)) - oldval(i)
    + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
  end do
end do
```

1-D Wave Equation problem (7)

- #Collect results and write to file
  if I am MASTER
    receive results from each WORKER
    write results to file
  else if I am WORKER
    send results to MASTER
  endif

Izvori o paralelnom programiranju

Literatura i izvori

- Blaise Barney,
  Introduction to Parallel Computing,
  https://computing.llnl.gov/tutorials/parallel_comp/,
  2016.
- Ian Foster,
  Designing and Building Parallel Programs
- Grama, Gupta, Karypis, Kumar,
  Introduction to Parallel Computing
- Culler, Singh, Gupta,
  Parallel Computer Architecture
  (A Hardware/Software Approach), 1998.