Univerza v Mariboru

FERI

Programiranje za superračunalnik

Borko Bošković, Janez Brest

9. december

2024

Uvod

Vsebina

- Učinkovito reševanje problemov
- Amdahlov zakon
- Gustafsonov zakon
- Problem LABS

Učinkovito reševanje problemov

- Načrtovanje in implementacija učinkovitih algoritmov
  - Časovna zahtevnost - O($log\ n$), O(n^2)
- Izbira tehnologij
  - Programski jezik (C++, Java, Python, C#)
  - Programska okolja in knjižnice
- Paralelno reševanje problema
  - Niti, OpenMP, MPI, CUDA, OpenCL

Učinkovito reševanje problemov

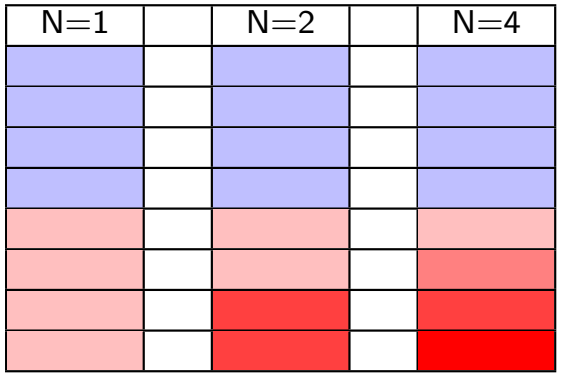
- Kvaliteta programske kode
- Strojna oprema
  - Osebni računalnik
  - Prenosni računalnik
  - Grafična kartica
  - Superračunalnik

Orodja

- GNU C++
- CMake
- GNU GDB
- GIT
- Profilirnik valgrind
- Onlinegdb
- Google Colab
- Singularity

Amdahlov zakon

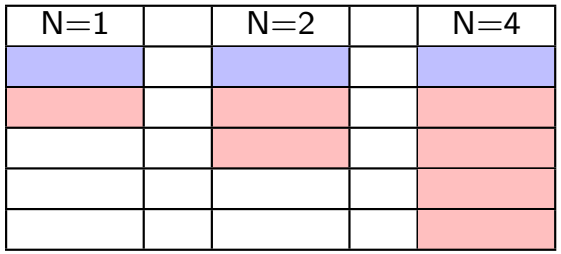
- Teoretična pohitritev enake naloge na boljšem sistemu (z večimi procesorji).
- $S = \frac{1}{1-P+\frac{P}{N}}$, kjer P predstavlja razmerje programa ki je paraleliziran in N število procesorjev.
- V teoretičnem primeru da imamo neskončno procesorjev, se pohitritev približuje $\frac{1}{1-P}$.



- $P = 0,5$
- N=1: $S = \frac{1}{1-0,5+\frac{0,5}{1}} = 1$ (sekvenčni: 8 celic, paralelni: 8 celic)
- N=2: $S = \frac{1}{1-0,5+\frac{0,5}{2}} = 1,33$ (sekvenčni: 8 celic, paralelni: 6 celic)
- N=4: $S = \frac{1}{1-0,5+\frac{0,5}{4}} = 1,6$ (sekvenčni: 8 celic, paralelni: 5 celic)

Gustafsonov zakon

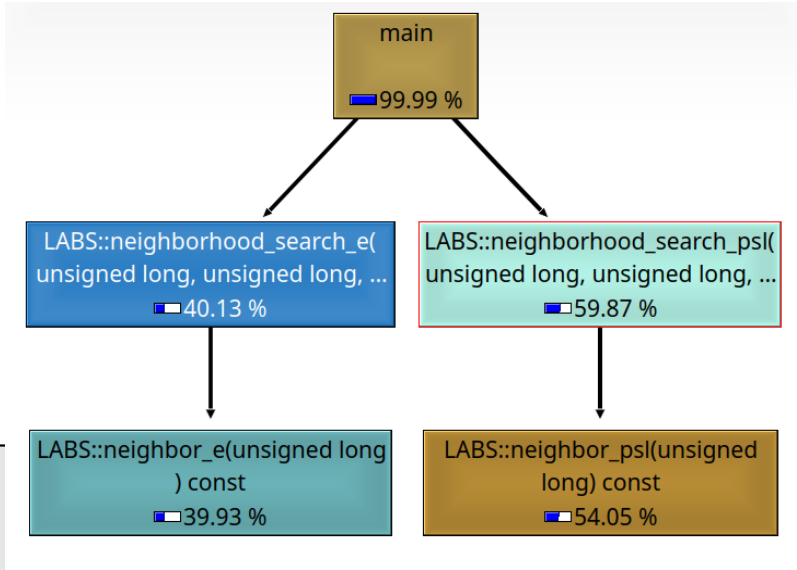
- Pohitritev glede na fiksno obremenitev enega procesorja.
- $S = N + (1 - N) \times s$, kjer N predstavlja število procesorjev in s delež programa, ki se izvaja paralelno.



- $s = 0,5$
- N=1: $S = 1 + (1 - 1) * 0,5 = 1$ (sekvenčni: 2 celici, paralelni: 2 celici)
- N=2: $S = 2 + (1 - 2) * 0,5 = 1,5$ (sekvenčni: 2 celic, paralelni: 3 celici)
- N=4: $S = 4 + (1 - 4) * 0,5 = 2,5$ (sekvenčni: 2 celic, paralelni: 5 celici)

Naloge

1. Izračunajte število ovrednotenj zaporedij na sekundo za program, ki izvaja paralelno statični metodi `search_psl` in `search_e`. Pri tem upoštevajte, hitrost programa prikazanega na primeru, deleže izvajanja funkcij prikazanih na sliki in da ima sistem 64 jeder.




```
$ ./labs_neighborhood_search 42 10000000 42
Searching ...
E: 125 F: 7.056 speed: 1.8622e+07 eval/sec
Searching ...
PSL: 5 speed: 1.18483e+07 eval/sec
```

Problem LABS

```
class LABS{
public:
    enum value { p=+1, n=-1};
    LABS(const size_t L): L(L), seq(L,p), c(L,0),
          e(numeric_limits<int>::max()), psl(numeric_limits<int>::max()) {};
    LABS(const LABS & l): L(l.L), seq(l.seq), c(l.c), e(l.e), psl(l.psl) {}
    LABS& operator=(const LABS & l);
    inline double get_mf() const { return (L*L)/(2.0*e); }
    inline int get_e() const { return e; }
    inline int get_psl() const { return psl; }
    void random(mt19937 & rand);
    void evaluate_e();
    void evaluate_psl();
    static LABS random_search_e(const size_t seed, const size_t n, const size_t L);
    static LABS random_search_psl(const size_t seed, const size_t n, const size_t L);

private:
    const size_t L;
    vector<value> seq;
    vector<int> c;
    int e, psl;
};
```

Ovrednotenje sekvenc

```
void LABS::evaluate_e(){
    e = 0;
    for (size_t k=1; k<L; k++) {
        c[k]=0;
        for (size_t i=0; i<=L-k-1; i++) c[k] += seq[i]*seq[i+k];
        e += c[k]*c[k];
    }
}
```

```
void LABS::evaluate_psl(){
    psl = 0;
    for (size_t k=1; k<L; k++) {
        c[k]=0;
        for (size_t i=0; i<=L-k-1; i++) c[k] += seq[i]*seq[i+k];
        if(abs(c[k]) > psl) psl = abs(c[k]);
    }
}
```

Iskanje sekvenc

```
LABS LABS::random_search_e(const size_t seed, const size_t n, const size_t L){
    LABS current(L), best(L);
    mt19937 rand(seed);
    best.random(rand);
    best.evaluate_e();
    for(size_t i=0; i<n; i++){
        current.random(rand);
        current.evaluate_e();
        if(current.get_e() < best.get_e()) best = current;
    }
    return best;
}
```

```
LABS LABS::random_search_psl(const size_t seed, const size_t n, const size_t L){
    LABS current(L), best(L);
    mt19937 rand(seed);
    best.random(rand);
    best.evaluate_psl();
    for(size_t i=0; i<n; i++){
        current.random(rand);
        current.evaluate_psl();
        if(current.get_psl() < best.get_psl()) best = current;
    }
    return best;
}
```

Merjenje časa

```
        cout<<"Searching ..."<<endl;
        auto start = system_clock::now();
        LABS best = LABS::random_search_e(seed,n,D);
        auto end = system_clock::now();
        auto elapsed = duration_cast<milliseconds>(end - start);
        cout<<"E: "<<best.get_e()<<" F: "<<best.get_mf();
        cout<<" speed: "<<n/(elapsed.count()/1000.0)<<" eval/sec"<<endl;
```

```
        cout<<"Searching ..."<<endl;
        start = system_clock::now();
        best = LABS::random_search_psl(seed,n,D);
        end = system_clock::now();
        elapsed = duration_cast<milliseconds>(end - start);
        cout<<"PSL: "<<best.get_psl();
        cout<<" speed: "<<n/(elapsed.count()/1000.0)<<" eval/sec"<<endl;
```

Merjenje časa

```
cmake_minimum_required(VERSION 3.5)

project(labs_neighborhood_search LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

add_executable(labs_neighborhood_search main.cpp)
```

```
$ cd naloge/labs_random_search/
$ cmake -DCMAKE_BUILD_TYPE=Release .
$ make
$ ./labs_random_search 1 10000000 63
Searching ...
E: 595 MF: 3.33529 runtime: 131.379 sec
Searching ...
PSL: 7 runtime: 137.815 sec
$ ./labs_random_search 2 10000000 63
Searching ...
E: 547 MF: 3.62797 runtime: 132.58 sec
Searching ...
PSL: 7 runtime: 137.831 sec
```

Ovrednotenje soseda

```
int LABS::neighbor_e(const size_t i) const{
    int e = 0, ck;
    const size_t lmt = std::max(L-i,i+1);
    size_t k=1;
    for(;k<lmt; k++) {
        ck = c[k];
        if(i+k<L) ck -= 2*seq[i]*seq[k+i];
        if(k<=i) ck -= 2*seq[i-k]*seq[i];
        e += ck*ck;
    }
    for (; k<L; k++) e += c[k]*c[k];
    return e;
}
```

Izbira soseda

```
void LABS::update_e(const size_t i, const int e){
    const size_t lmt = max(L-i,i+1);
    size_t k=1;
    for (;k<lmt; k++){
        int ck = c[k];
        if(i+k<L) ck -= 2*seq[i]*seq[k+i];
        if(k<=i) ck -= 2*seq[i-k]*seq[i];
        c[k] = ck;
    }
    this->e = e;
    seq[i] = (value)(-seq[i]);
    #ifndef NDEBUG
    int update_e = e;
    evaluate_e();
    if(e != update_e) throw string("Wrong E!");
    #endif
}
```

Iskanje s pomočjo sosedov

```
LABS LABS::neighborhood_search_e(const size_t seed, const size_t n, const size_t L){
    LABS current(L), best(L);
    mt19937 rand(seed);
    best.random(rand);
    best.evaluate_e();
    current = best;
    size_t nfes=0, best_neighbor;
    int best_neighbor_e, e;
    while(nfes < n){
        best_neighbor_e = numeric_limits<int>::max();
        for(size_t i=0; i<L; i++){
            e = current.neighbor_e(i);
            if(e < best_neighbor_e){
                best_neighbor = i;
                best_neighbor_e = e;
            }
        }
        nfes+=L;
        if(best_neighbor_e >= current.get_e()){
            current.random(rand);
            current.evaluate_e();
            nfes++;
        }
        else{
            current.update_e(best_neighbor,best_neighbor_e);
        }
        if(current.get_e() < best.get_e()) best = current;
    }
    return best;
}
```

Merjenje časa

```
$ cd naloge/labs_random_search/
$ cmake -DCMAKE_BUILD_TYPE=Debug .
$ make
$ ./labs_neighborhood_search 1 10000000 63
Searching ...
Wrong E!
```

```
$ cd naloge/labs_random_search/
$ cmake -DCMAKE_BUILD_TYPE=Release .
$ make
$ ./labs_neighborhood_search 1 100000000 63
Searching ...
E: 543 MF: 3.6547 runtime: 7.136 sec
Searching ...
PSL: 13 runtime: 10.087 sec
$ ./labs_neighborhood_search 2 100000000 63
Searching ...
E: 499 MF: 3.97695 runtime: 7.278 sec
Searching ...
PSL: 12 runtime: 10.197 sec
```

Naloge

1. Implementirajte algoritem, kjer je dolžina sprehoda $8L$ in v vsakem koraku izberemo najboljšega soseda, ki ga še nismo obiskali.
2. Implementirajte iskanje, ki za naslednjo rešitev sprehoda izbere prvo rešitev, ki je boljša od trenutne rešitve. Če boljšega zaporedja ne najdete, trenutno zaporedje dobi vrednost najboljšega soseda. Dolžino sprehoda omejite na $8L$.
3. Na osnovi števila ovrednotenj na sekundo, določite koliko ovrednotenj program potrebuje pri L=513, da se bo izvajal 5 sekund. Nato programe zaženite 25 krat in primerjajte povprečne vrednosti F in PSL. Določite najboljši algoritem za oba kriterija.

MPI

- Message Passing Interface
- Knjižnica za porazdeljeno računanje
- Pošiljanje sporočil
- Več instanc oz. procesov programa
- Vsaka instanca programa izvaja različno pot istega programa

Komunikacija od točke do točke

```
1  char sporocilo[100];
2  int mrank, oznaka=1;
3  MPI_Status status;
4
5  MPI_Comm_rank(MPI_COMM_WORLD, &mrank) // Ugotovimo rnak procesa
6  if(mrank == 0){
7      strcpy(sporocilo, "Sporocilo");
8      // Sporocilo posljemo procesu z rankom 1
9      MPI_Send(sporocilo,strlen(msporocilo)+1,MPI_CHAR,1,oznaka,MPI_COMM_WORLD);
10 }
11 else if (mrank == 1){
12     // Preberemo sporocilo, ki ga je poslal proces z rankom 0
13     MPI_Recv(sporocilo,100,MPI_CHAR,0,oznaka,MPI_COMM_WORLD,&status);
14 }
```

Prevajanje, zagon in testiranje programa

```
cmake_minimum_required(VERSION 3.5)

project(mpi_labs_neighborhood_search LANGUAGES CXX)
find_package(MPI REQUIRED)
SET(CMAKE_CXX_FLAGS "${CMAKE_XX_FLAGS} -fopenmp")

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
add_executable(mpi_labs_neighborhood_search main.cpp)
include_directories(mpi_labs_neighborhood_search ${MPI_INCLUDE_PATH})
target_link_libraries(mpi_labs_neighborhood_search ${MPI_CXX_LIBRARIES})

include(CTest)
set(TARGET_E 1 1 1 1 2 2 7 3 8 12 13 5 10 6 19 15 24 32 25 29 26)
set(TARGET_PSL 1 1 1 1 1 1 2 1 2 2 2 1 2 1 2 2 2 2 2 2 2)
foreach(L RANGE 4 20)
    list(GET TARGET_E ${L} target_e)
    list(GET TARGET_PSL ${L} target_psl)
    add_test(NAME Test${L} COMMAND bash -c "./mpi_labs_neighborhood_search 42 10000000 ${L} >
    add_test(NAME Test_E_${L} COMMAND bash -c "grep -q '^E: ${target_e}' out_${L}.txt")
    add_test(NAME Test_PSL_${L} COMMAND bash -c "grep -q '^PSL: ${target_psl}' out_${L}.txt")
endforeach()

set(L 513)
foreach(seed RANGE 1 25)
    add_test(NAME L${L}_${seed} COMMAND bash -c "./mpi_labs_neighborhood_search ${seed} 15000
endforeach()
```

Prevajanje in zagon programa

```
$ cd naloge/mpi_send_recv/
$ cmake -DCMAKE_BUILD_TYPE=Release . && make
$ mpirun --use-hwthread-cpus -N 6 mpi_labs_neighborhood_search 42 1000000 101
Slave 1 E: 970 speed: 1.14943e+07 eval/sec
Slave 2 E: 1046 speed: 9.90099e+06 eval/sec
Slave 3 E: 1046 speed: 9.52381e+06 eval/sec
...
Slave 5 PSL: 11 speed: 8.77193e+06 eval/sec
```

Testiranje programa

```
$ make test
Running tests...
Test project build
      Start  1: Test4
 1/76 Test  #1: Test4 ..............................   Passed    0.05 sec
      Start  2: Test_E_4
 2/76 Test  #2: Test_E_4 ...........................   Passed    0.00 sec
      Start  3: Test_PSL_4
 ...
75/76 Test #75: L513_24 ............................   Passed    8.22 sec
      Start 76: L513_25
76/76 Test #76: L513_25 ............................   Passed    8.22 sec

100% tests passed, 0 tests failed out of 76

Total Test time (real) = 205.72 sec
```

Model gospodar-suženj

```
1  int main(int argc, char *argv[]){
2      int size, rank;
3      MPI_Init(&argc, &argv);  // Inicializacija okolja MPI
4      MPI_Comm_size(MPI_COMM_WORLD, &size); // Stevilo procesov
5      MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Rank procesa
6      try{
7          if(rank == 0) master(size); // Gospodar - zbiratelj informacij
8          else slave(argc,argv,rank); // Suznji
9      }
10     catch (string err) {
11             cerr<<err<<std::endl;
12             return 1;
13     }
14     MPI_Finalize(); // Koncamo okolje MPI
15     return 0;
16 }
```

Suženj

```
1  void slave(const int argc, char * argv[], const int rank){
2      const int tag_e=1, tag_psl=2;
3      if(argc < 4) throw string("Three arguments are required: seed NFEs D!");
4      const size_t seed =atoi(argv[1]), NFEs = atoi(argv[2]), D = atoi(argv[3]);
5      auto start = system_clock::now();
6      LABS best = LABS::neighborhood_search_e(seed+rank,NFEs,D);
7      auto end = system_clock::now();
8      auto elapsed = duration_cast<milliseconds>(end - start);
9      int best_e = best.get_e();
10     double speed = NFEs/(elapsed.count()/1000.0);
11     MPI_Send(&best_e,1, MPI_INT, 0, tag_e, MPI_COMM_WORLD); // Posljemo E z oznako 1
12     MPI_Send(&speed,1, MPI_DOUBLE, 0, tag_e, MPI_COMM_WORLD); // Posljemo hitrost z oznako 1
13
14     start = system_clock::now();
15     best = LABS::neighborhood_search_psl(seed+rank,NFEs,D);
16     end = system_clock::now();
17     elapsed = duration_cast<milliseconds>(end - start);
18     int best_psl = best.get_psl();
19     speed = NFEs/(elapsed.count()/1000.0);
20     MPI_Send(&best_psl,1, MPI_INT, 0, tag_psl, MPI_COMM_WORLD); // Posljemo PSL z oznako 2
21     MPI_Send(&speed,1, MPI_DOUBLE, 0, tag_psl, MPI_COMM_WORLD); // Posljemo hitrost z oznako 2
22 }
```

Gospodar

```
1  void master(const size_t size){
2      MPI_Status status;
3      int best_e, best_psl;
4      double speed;
5      const int tag_e=1, tag_psl=2;
6      for(size_t i=1; i<size; i++){
7          // Prejmemo E z oznako 1
8          MPI_Recv(&best_e, 1, MPI_INT, i, tag_e, MPI_COMM_WORLD, &status);
9          // Prejmemo hitrost z oznako 1
10         MPI_Recv(&speed, 1, MPI_DOUBLE, i, tag_e, MPI_COMM_WORLD, &status);
11         std::cout<<"Slave "<<i<<" E: "<<best_e<<" speed: "<<speed<<" eval/sec"<<std::endl;
12     }
13     for(size_t i=1; i<size; i++){
14         MPI_Recv(&best_psl, 1, MPI_INT, i, tag_psl, MPI_COMM_WORLD, &status);
15         // Prejmemo PSL z oznako 2
16         MPI_Recv(&speed, 1, MPI_DOUBLE, i, tag_psl, MPI_COMM_WORLD, &status);
17         // Prejmemo hitrost z oznako 2
18         std::cout<<"Slave "<<i<<" PSL: "<<best_psl<<" speed: "<<speed<<" eval/sec"<<std::endl
19     }
20 }
```

Zagon programa

```
$ cmake -DCMAKE_BUILD_TYPE=Release . && make
$ mpirun --use-hwthread-cpus -N 6 mpi_labs_neighborhood_search 42 1000000 101
Slave 1 E: 970 speed: 1.14943e+07 eval/sec
Slave 2 E: 1046 speed: 9.90099e+06 eval/sec
Slave 3 E: 1046 speed: 9.52381e+06 eval/sec
...
Slave 5 PSL: 11 speed: 8.77193e+06 eval/sec
```

Naloge

1. Spremenite program tako, da se bo zaustavitveni pogoj vsakega procesa enak $\frac{NFEs}{N}$. S pomočjo tega programa določite faktor pohitritve po Amdahlovem zakonu.
2. V program dodajte zaustavitveni pogoj: kvaliteta rešitve. Ko proces doseže ali preseže izbrano kvaliteo rešitve, konča iskanje.

Asinhrona komunikacija - suženj

```
1  void slave(const int argc, char * argv[], const int rank){
2      const int tag_e=1, tag_psl=2;
3      if(argc < 4) throw string("Three arguments are required: seed NFEs D!");
4      const size_t seed =atoi(argv[1]), NFEs = atoi(argv[2]), D = atoi(argv[3]);
5      auto start = system_clock::now();
6      LABS best = LABS::search_e(seed+rank,NFEs,D);
7      auto end = system_clock::now();
8      auto elapsed = duration_cast<milliseconds>(end - start);
9      int best_e = best.get_e();
10     double speed = NFEs/(elapsed.count()/1000.0);
11     MPI_Request req[4];
12     MPI_Isend(&best_e,1, MPI_INT, 0, tag_e, MPI_COMM_WORLD,&req[0]); // Posljemo E, oznaka 1
13     MPI_Isend(&speed,1, MPI_DOUBLE, 0, tag_e, MPI_COMM_WORLD,&req[1]); // Posljemo hitrost, o
14     start = system_clock::now();
15     best = LABS::search_psl(seed+rank,NFEs,D);
16     end = system_clock::now();
17     elapsed = duration_cast<milliseconds>(end - start);
18     int best_psl = best.get_psl();
19     speed = NFEs/(elapsed.count()/1000.0);
20     MPI_Isend(&best_psl,1, MPI_INT, 0, tag_psl, MPI_COMM_WORLD,&req[2]); // Posljemo PSL, ozn
21     MPI_Isend(&speed,1, MPI_DOUBLE, 0, tag_psl, MPI_COMM_WORLD,&req[3]); // Posljemo hitros,
22     MPI_Status status;
23     for(size_t i=0; i<4; i++) MPI_Wait(&req[i], &status);
24 }
```

Asinhrona komunikacija - gospodar 1/2

```
1  void master(const size_t size){
2      struct Buffer{
3          Buffer() : e(0), psl(0), e_speed(0), psl_speed(0), e_flag(true),
4              e_speed_flag(true), psl_flag(true), psl_speed_flag(true)  {}
5          int e, psl;
6          double e_speed, psl_speed;
7          bool e_flag, e_speed_flag, psl_flag, psl_speed_flag;
8          MPI_Request e_req, e_speed_req, psl_req, psl_speed_req;
9      };
10     MPI_Status status;
11     int count=16, flag;
12     const int tag_e=1, tag_psl=2;
13     vector<Buffer> buffer(size);
14     for(size_t i=1; i<size; i++){
15         MPI_Irecv(&buffer[i].e,1,MPI_INT,i,tag_e,MPI_COMM_WORLD,&buffer[i].e_req);
16         MPI_Irecv(&buffer[i].e_speed,1,MPI_DOUBLE,i,tag_e,MPI_COMM_WORLD,&buffer[i].e_speed_r
17         MPI_Irecv(&buffer[i].psl,1,MPI_INT,i,tag_psl,MPI_COMM_WORLD,&buffer[i].psl_req);
18         MPI_Irecv(&buffer[i].psl_speed,1,MPI_DOUBLE,i,tag_psl,MPI_COMM_WORLD,&buffer[i].psl_s
19     }
```

Asinhrona komunikacija - gospodar 2/2

```
1     while(count > 0){
2         for(size_t i = 1; i<size; i++){
3             MPI_Test(&(buffer[i].e_req), &flag, &status);
4             if(flag && buffer[i].e_flag){
5                 std::cout<<"E:"<<buffer[i].e<<std::endl;
6                 buffer[i].e_flag = false; count --;
7             }
8             MPI_Test(&buffer[i].e_speed_req, &flag, &status);
9             if(flag && buffer[i].e_speed_flag){
10                std::cout<<"E speed:"<<buffer[i].e_speed<<std::endl;
11                buffer[i].e_speed_flag = false; count --;
12            }
13            MPI_Test(&buffer[i].psl_req, &flag, &status);
14            if(flag && buffer[i].psl_flag){
15                std::cout<<"PSL:"<<buffer[i].psl<<std::endl;
16                buffer[i].psl_flag = false; count --;
17            }
18            MPI_Test(&buffer[i].psl_speed_req, &flag, &status);
19            if(flag && buffer[i].psl_speed_flag){
20                std::cout<<"PSL speed:"<<buffer[i].psl_speed<<std::endl;
21                buffer[i].psl_speed_flag = false; count --;
22            }
23        }
24    }
25 }
```

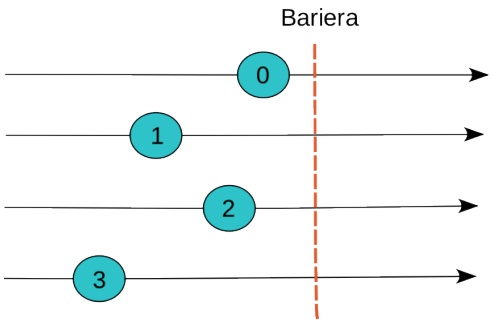
Asinhrona komunikacija - zagon programa

```
$ mpirun --use-hwthread-cpus -N 6 mpi_labs_neighborhood_search 42 1000000 101
E:1046
E speed:1.26582e+07
E speed:1.13636e+07
E:1050
E:1006
E speed:1.11111e+07
E:970
E speed:9.70874e+06
E speed:9.52381e+06
E:1046
PSL speed:8.69565e+06
PSL:12
PSL:11
PSL speed:8.77193e+06
PSL:12
PSL speed:8.06452e+06
```

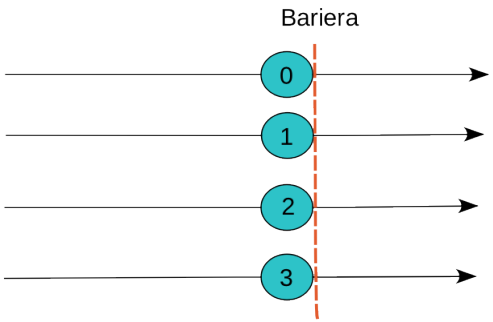
Naloge

1 V program dodajte zaustavitveni pogoj: čas iskanja (`runtime`). Ko proces doseže ali preseže določen čas, konča iskanje. Zaustavljanje implementirajte tako, da en proces zaspi za določen čas. Ko se zbudi, pošlje ostalim procesom sporočilo, naj končajo z iskanjem.

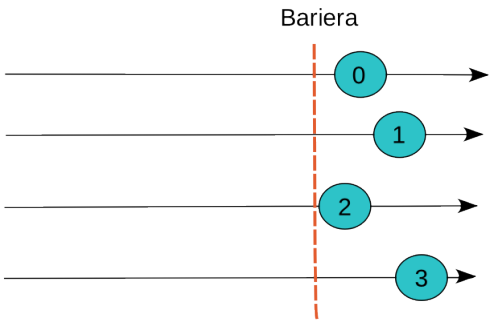
Bariere



(a) Procesi pred sinhronizacijo.



(b) Procesi v sinhronizacijski točki.



(c) Procesi po sinhronizaciji.

Bariere - suženj

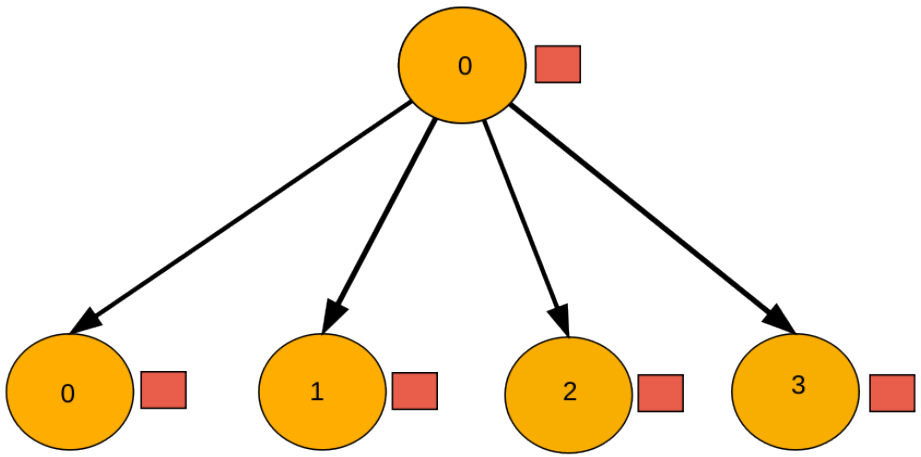
```
1  void slave(const int argc, char * argv[], const int rank){
2      const int tag_e=1, tag_psl=2;
3      if(argc < 4) throw string("Three arguments are required: seed NFEs D!");
4      const size_t seed =atoi(argv[1]), NFEs = atoi(argv[2]), D = atoi(argv[3]);
5      auto start = system_clock::now();
6      LABS best = LABS::neighborhood_search_e(seed+rank,NFEs,D);
7      auto end = system_clock::now();
8      auto elapsed = duration_cast<milliseconds>(end - start);
9      int best_e = best.get_e();
10     double speed = NFEs/(elapsed.count()/1000.0);
11     MPI_Send(&best_e,1, MPI_INT, 0, tag_e, MPI_COMM_WORLD); // Posljemo E z oznako 1
12     MPI_Send(&speed,1, MPI_DOUBLE, 0, tag_e, MPI_COMM_WORLD); // Posljemo hitrost z oznako 1
13
14     MPI_Barrier(MPI_COMM_WORLD); // Cakamo na ostale procese
15
16     start = system_clock::now();
17     best = LABS::neighborhood_search_psl(seed+rank,NFEs,D);
18     end = system_clock::now();
19     elapsed = duration_cast<milliseconds>(end - start);
20     int best_psl = best.get_psl();
21     speed = NFEs/(elapsed.count()/1000.0);
22     MPI_Send(&best_psl,1, MPI_INT, 0, tag_psl, MPI_COMM_WORLD); // Posljemo PSL z oznako 2
23     MPI_Send(&speed,1, MPI_DOUBLE, 0, tag_psl, MPI_COMM_WORLD); // Posljemo hitrost z oznako 2
24 }
```

Bariere - gospodar

```
1  void master(const size_t size){
2      MPI_Status status;
3      int best_e, best_psl;
4      double speed;
5      const int tag_e=1, tag_psl=2;
6      std::cout<<"F"<<std::endl;
7      for(size_t i=1; i<size; i++){
8          // Prejmemo E z oznako 1
9          MPI_Recv(&best_e, 1, MPI_INT, i, tag_e, MPI_COMM_WORLD, &status);
10         // Prejmemo hitrost z oznako 1
11         MPI_Recv(&speed, 1, MPI_DOUBLE, i, tag_e, MPI_COMM_WORLD, &status);
12         std::cout<<"Slave "<<i<<" E: "<<best_e<<" speed: "<<speed<<" eval/sec"<<std::endl;
13     }
14
15     MPI_Barrier(MPI_COMM_WORLD); // Cakamo na ostale procese
16     std::cout<<"PSL"<<std::endl;
17
18     for(size_t i=1; i<size; i++){
19         MPI_Recv(&best_psl, 1, MPI_INT, i, tag_psl, MPI_COMM_WORLD, &status);
20         // Prejmemo PSL z oznako 2
21         MPI_Recv(&speed, 1, MPI_DOUBLE, i, tag_psl, MPI_COMM_WORLD, &status);
22         // Prejmemo hitrost z oznako 2
23         std::cout<<"Slave "<<i<<" PSL: "<<best_psl<<" speed: "<<speed<<" eval/sec"<<std::endl
24     }
25 }
```

Komunikacija enega z ostalimi

```
1  MPI_Bcast(
2      void* data,                 // Podatki, ki jih posiljamo
3      int count,                  // Stevilo elementov, ki jih posiljamo
4      MPI_Datatype datatype,      // Tip podatkov, ki jih posiljamo
5      int root,                   // Rank korenskega vozlisca
6      MPI_Comm communicator)      // Komunikator
```

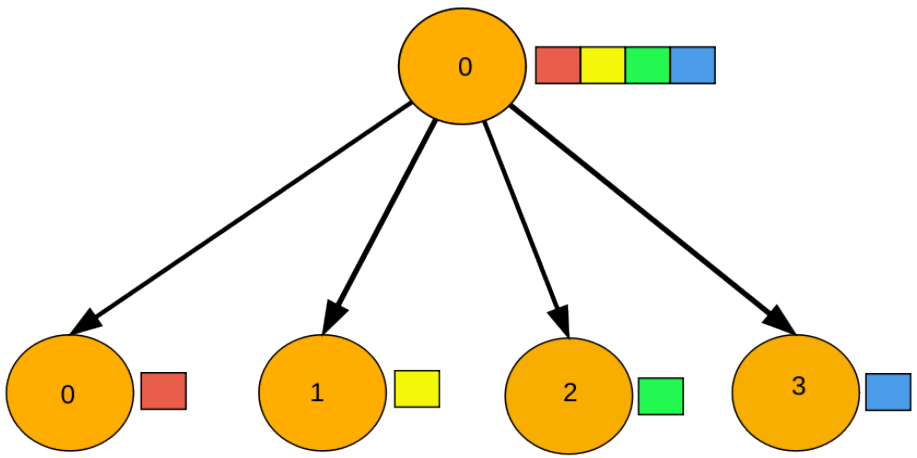


Komunikacija enega z ostalimi

```
1  int main(int argc, char *argv[]){
2      int size, rank;
3      size_t seed, NFEs, D;
4      MPI_Init(&argc, &argv);  // Inicializacija okolja MPI
5      MPI_Comm_size(MPI_COMM_WORLD, &size); // Stevilo procesov
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Rank procesa
7      try{
8          if(rank == 0){
9              if(argc < 4) throw string("Three arguments are required: seed NFEs D!");
10             seed =atoi(argv[1]);
11             NFEs = atoi(argv[2]);
12             D = atoi(argv[3]);
13         }
14         MPI_Bcast(&seed, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
15         MPI_Bcast(&NFEs, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
16         MPI_Bcast(&D, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
17         seed += rank;
18         if(rank == 0) master(size); // Gospodar - zbiratelj informacij
19         else slave(rank,seed,NFEs,D); // Suznji
20     }
21     catch (string err) {
22             cerr<<err<<std::endl;
23             return 1;
24     }
25     MPI_Finalize(); // Koncamo okolje MPI
26     return 0;
27 }
```

Pošiljanje elementov polja

```
1  MPI_Scatter(
2      void* send_data,                // Podatki, ki jih posiljamo
3      int send_count,                 // Stevilo elementov, ki jih posiljamo dolocenemu procesu
4      MPI_Datatype send_datatype,     // Tip podatkov, ki jih posiljamo
5      void* recv_data,                // Kazalec kamor shranimo prejete podatke
6      int recv_count,                 // Stevilo elementov, ki jih proces prejeme
7      MPI_Datatype recv_datatype,     // Podatkovni tip prejetih podatkov
8      int root,                       // Rank korenskega vozlisca
9      MPI_Comm communicator)          // Komunikator
```

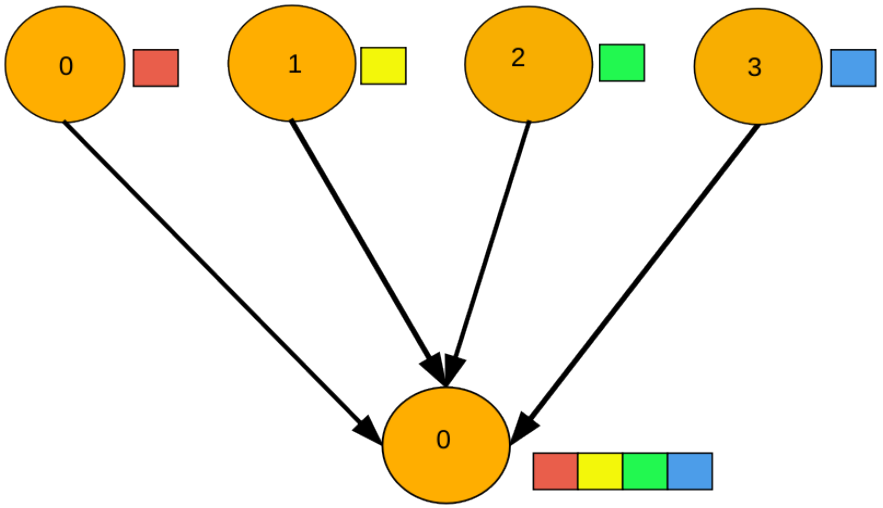


Pošiljanje elementov polja

```
1  int main(int argc, char *argv[]){
2      int size, rank;
3      size_t NFEs, D, my_seed;
4      std::vector<size_t> seed;
5      MPI_Init(&argc, &argv);  // Inicializacija okolja MPI
6      MPI_Comm_size(MPI_COMM_WORLD, &size); // Stevilo procesov
7      MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Rank procesa
8      seed.resize(size);
9      try{
10         if(rank == 0){
11             if(argc < 4) throw string("Three arguments are required: seed NFEs D!");
12             seed[0] =atoi(argv[1]);
13             NFEs = atoi(argv[2]);
14             D = atoi(argv[3]);
15             for(size_t i=1; i<size; i++) seed[i] = seed[i-1]+1;
16         }
17         MPI_Scatter(&seed[0],1,MPI_UNSIGNED_LONG,&my_seed,1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD
18         MPI_Bcast(&NFEs, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
19         MPI_Bcast(&D, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
20         if(rank == 0) master(size); // Gospodar - zbiratelj informacij
21         else slave(rank,my_seed,NFEs,D); // Suznji
22     }
23     catch (string err) {
24             cerr<<err<<std::endl;
25             return 1;
26     }
27     MPI_Finalize(); // Koncamo okolje MPI
28     return 0;
29 }
```

Prejemanje elementov v polja

```
1  MPI_Gather(
2      void* send_data,                 // Podatki, ki jih posiljamo
3      int send_count,                  // Stevilo elementov, ki jih posiljamo dolocenemu procesu
4      MPI_Datatype send_datatype,      // Tip podatkov, ki jih posiljamo
5      void* recv_data,                 // Kazalec kamor shranimo prejete podatke
6      int recv_count,                  // Stevilo elementov, ki jih proces prejeme
7      MPI_Datatype recv_datatype,      // Podatkovni tip prejetih podatkov
8      int root,                        // Rank korenskega vozlisca
9      MPI_Comm communicator)           // Komunikator
```



Prejemanje elementov v polje - suženj

```
1  void slave(const int rank, size_t seed, const size_t NFEs, const size_t D,
2             int best_e[], double speed_e[], int best_psl[], double speed_psl[]){
3      const int tag_e=1, tag_psl=2;
4      auto start = system_clock::now();
5      LABS best = LABS::neighborhood_search_e(seed,NFEs,D);
6      auto end = system_clock::now();
7      auto elapsed = duration_cast<milliseconds>(end - start);
8      int s_best_e = best.get_e();
9      double s_speed_e = NFEs/(elapsed.count()/1000.0);
10     // Posljemo gospodarju
11     MPI_Gather(&s_best_e,1, MPI_INT, best_e, 1, MPI_INT, 0, MPI_COMM_WORLD);
12     MPI_Gather(&s_speed_e,1, MPI_DOUBLE, speed_e, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
13
14     MPI_Barrier(MPI_COMM_WORLD); // Cakamo na ostale procese
15
16     start = system_clock::now();
17     best = LABS::neighborhood_search_psl(seed,NFEs,D);
18     end = system_clock::now();
19     elapsed = duration_cast<milliseconds>(end - start);
20     int s_best_psl = best.get_psl();
21     double s_speed_psl = NFEs/(elapsed.count()/1000.0);
22     // Posljemo gospodarju
23     MPI_Gather(&s_best_psl,1, MPI_INT, best_psl, 1, MPI_INT, 0, MPI_COMM_WORLD);
24     MPI_Gather(&s_speed_psl,1, MPI_DOUBLE, speed_psl, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
25 }
```

Prejemanje elementov v polje - gospodar

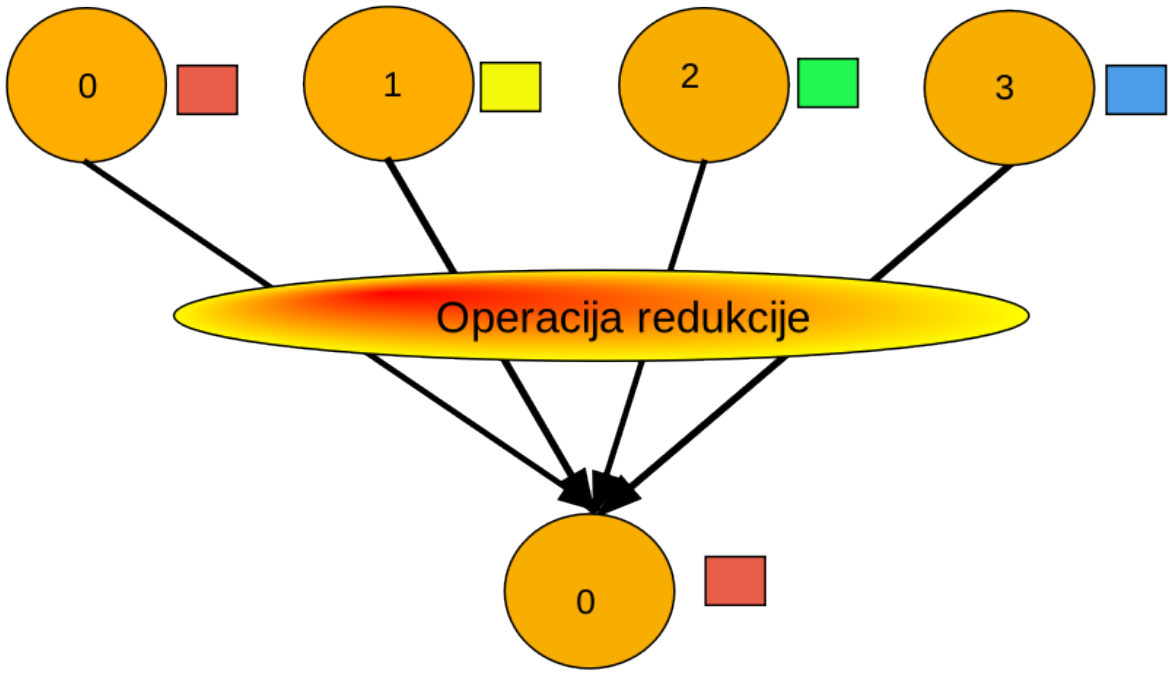
```
1  void master(int best_e[], double speed_e[], int best_psl[], double speed_psl[], const size_t s
2      MPI_Status status;
3      int m_best_e=0, m_best_psl=0;
4      double m_speed_e=0, m_speed_psl=0;
5      std::cout<<"F"<<std::endl;
6      // Prejmemo rezultate
7      MPI_Gather(&m_best_e,1, MPI_INT, best_e, 1, MPI_INT, 0, MPI_COMM_WORLD);
8      MPI_Gather(&m_speed_e,1, MPI_DOUBLE, speed_e, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
9      for(size_t i=1; i<size; i++)
10         std::cout<<"Slave "<<i<<" E: "<<best_e[i]<<" speed: "<<speed_e[i]<<" eval/sec"<<std::e
11
12     MPI_Barrier(MPI_COMM_WORLD); // Cakamo na ostale procese
13     std::cout<<"PSL"<<std::endl;
14
15     // Prejmemo rezultate
16     MPI_Gather(&m_best_psl,1, MPI_INT, best_psl, 1, MPI_INT, 0, MPI_COMM_WORLD);
17     MPI_Gather(&m_speed_psl,1, MPI_DOUBLE, speed_psl, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
18     for(size_t i=1; i<size; i++)
19        std::cout<<"Slave "<<i<<" PSL: "<<best_psl[i]<<" speed: "<<speed_psl[i]<<" eval/sec"<<s
20 }
```

Prejemanje elementov v polje - zagon

```
1  $ mpirun  -n 6 mpi_labs_gather 1 1000000 80
2  F
3  Slave 1 E: 511 speed: 1.66667e+07 eval/sec
4  Slave 2 E: 487 speed: 1.66667e+07 eval/sec
5  Slave 3 E: 547 speed: 2.5e+07 eval/sec
6  Slave 4 E: 535 speed: 1.66667e+07 eval/sec
7  Slave 5 E: 519 speed: 1.66667e+07 eval/sec
8  PSL
9  Slave 1 PSL: 10 speed: 1.11111e+07 eval/sec
10 Slave 2 PSL: 9 speed: 1.11111e+07 eval/sec
11 Slave 3 PSL: 9 speed: 1.42857e+07 eval/sec
12 Slave 4 PSL: 9 speed: 1e+07 eval/sec
13 Slave 5 PSL: 8 speed: 1.11111e+07 eval/sec
```

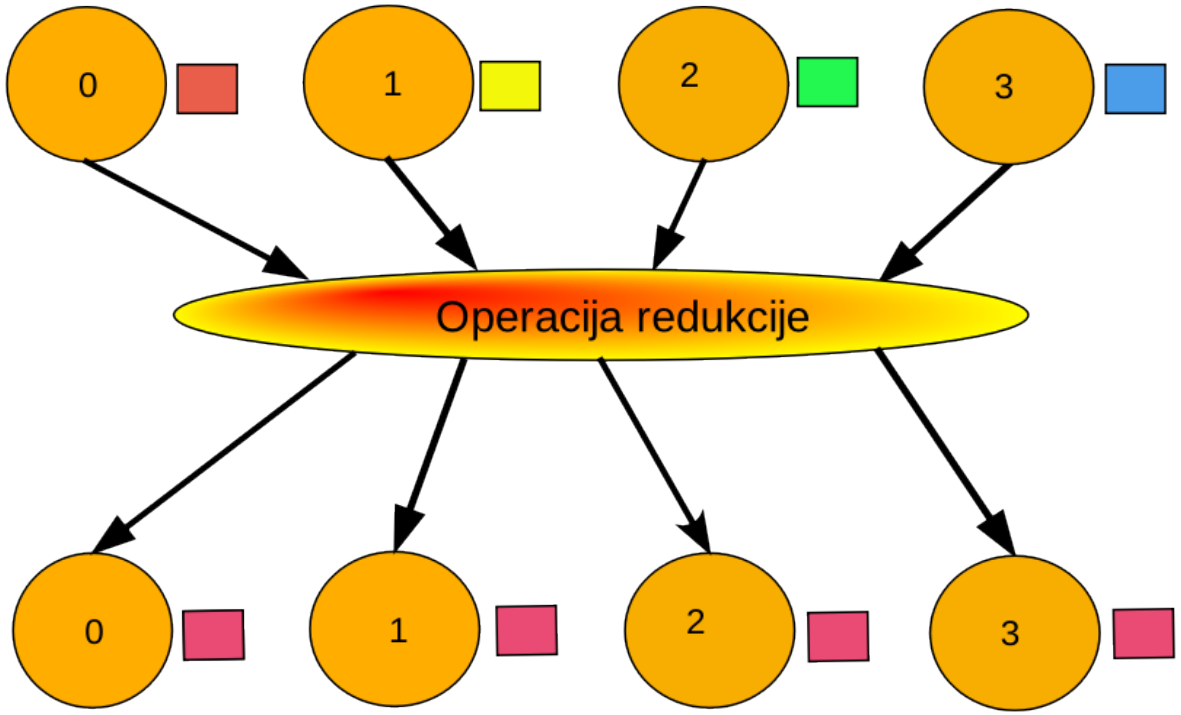
Redukcija

```
1  MPI_Reduce(
2      void* send_data,            // Podatki, ki jih posiljamo
3      void* recv_data,            // Kazalec kamor shranimo prejete podatke
4      int count,                  // Stevilo elementov v prejetih podatkih
5      MPI_Datatype datatype,      // Tip podatkov
6      MPI_Op op,                  // Operacija redukcije
7      int root,                   // Rank korenskega vozlisca
8      MPI_Comm communicator)      // Komunikator
```



Redukcija

```
1  MPI_Allreduce(
2      void* send_data,          // Podatki, ki jih posiljamo
3      void* recv_data,          // Kazalec kamor shranimo prejete podatke
4      int count,                // Stevilo elementov v prejetih podatkih
5      MPI_Datatype datatype,    // Tip podatkov
6      MPI_Op op,                // Operacija redukcije
7      MPI_Comm communicator)    // Komunikator
```



Redukcija

```
void slave(const int argc, char * argv[], const int rank){
    const int tag_e=1, tag_psl=2;
    if(argc < 4) throw string("Three arguments are required: seed NFEs D!");
    const size_t seed =atoi(argv[1]), NFEs = atoi(argv[2]), D = atoi(argv[3]);
    auto start = system_clock::now();
    LABS best = LABS::neighborhood_search_e(seed+rank,NFEs,D);
    auto end = system_clock::now();
    auto elapsed = duration_cast<milliseconds>(end - start);
    int best_e = best.get_e();
    double speed = NFEs/(elapsed.count()/1000.0), sum_speed;
    MPI_Send(&best_e,1, MPI_INT, 0, tag_e, MPI_COMM_WORLD); // Posljemo E z oznako 1
    MPI_Send(&speed,1, MPI_DOUBLE, 0, tag_e, MPI_COMM_WORLD); // Posljemo hitrost z oznako 1
    MPI_Reduce(&speed, &sum_speed, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    std::cout<<"Slave "<<rank<<" total speed: "<<sum_speed<<" eval/sec"<<std::endl;

    start = system_clock::now();
    best = LABS::neighborhood_search_psl(seed+rank,NFEs,D);
    end = system_clock::now();
    elapsed = duration_cast<milliseconds>(end - start);
    int best_psl = best.get_psl();
    speed = NFEs/(elapsed.count()/1000.0);
    MPI_Send(&best_psl,1, MPI_INT, 0, tag_psl, MPI_COMM_WORLD); // Posljemo PSL z oznako 2
    MPI_Send(&speed,1, MPI_DOUBLE, 0, tag_psl, MPI_COMM_WORLD); // Posljemo hitrost z oznako 2
    MPI_Allreduce(&speed, &sum_speed, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    std::cout<<"Slave "<<rank<<" total speed: "<<sum_speed<<" eval/sec"<<std::endl;
}
```

Redukcija

```
void master(const size_t size){
    MPI_Status status;
    int best_e, best_psl;
    double speed, sum_speed;
    const int tag_e=1, tag_psl=2;
    std::cout<<"F"<<std::endl;
    for(size_t i=1; i<size; i++){
        // Prejmemo E z oznako 1
        MPI_Recv(&best_e, 1, MPI_INT, i, tag_e, MPI_COMM_WORLD, &status);
        // Prejmemo hitrost z oznako 1
        MPI_Recv(&speed, 1, MPI_DOUBLE, i, tag_e, MPI_COMM_WORLD, &status);
        std::cout<<"Slave "<<i<<" E: "<<best_e<<" speed: "<<speed<<" eval/sec"<<std::endl;
    }
    speed = 0;
    MPI_Reduce(&speed, &sum_speed, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    std::cout<<"Master total speed: "<<sum_speed<<std::endl;
    std::cout<<std::endl<<"PSL"<<std::endl;

    for(size_t i=1; i<size; i++){
        MPI_Recv(&best_psl, 1, MPI_INT, i, tag_psl, MPI_COMM_WORLD, &status);
        // Prejmemo PSL z oznako 2
        MPI_Recv(&speed, 1, MPI_DOUBLE, i, tag_psl, MPI_COMM_WORLD, &status);
        // Prejmemo hitrost z oznako 2
        std::cout<<"Slave "<<i<<" PSL: "<<best_psl<<" speed: "<<speed<<" eval/sec"<<std::endl
    }
    speed = 0;
    MPI_Allreduce(&speed, &sum_speed, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    std::cout<<"Master total speed: "<<sum_speed<<std::endl;
}
```

Redukcija

```
$ mpirun -n 6 mpi_labs_reduce 1 1000000 80
F
Slave 5 total speed: 4.64929e-310 eval/sec
Slave 3 total speed: 4.64123e-310 eval/sec
Slave 1 E: 612 speed: 1.43266e+06 eval/sec
Slave 2 E: 624 speed: 1.5015e+06 eval/sec
Slave 3 E: 588 speed: 1.43678e+06 eval/sec
Slave 1 total speed: 4.66313e-310 eval/sec
Slave 4 E: 632 speed: 1.43266e+06 eval/sec
Slave 5 E: 644 speed: 1.47929e+06 eval/sec
Master total speed: 7.2829e+06 eval/sec

PSL
Slave 2 total speed: 4.66978e-310 eval/sec
Slave 4 total speed: 4.68008e-310 eval/sec
Slave 1 PSL: 10 speed: 1.24844e+06 eval/sec
Slave 2 PSL: 10 speed: 1.25e+06 eval/sec
Slave 3 PSL: 9 speed: 1.25156e+06 eval/sec
Slave 4 PSL: 9 speed: 1.25156e+06 eval/sec
Slave 5 PSL: 9 speed: 1.25471e+06 eval/sec
Master total speed: 6.25627e+06 eval/sec
Slave 3 total speed: 6.25627e+06 eval/sec
Slave 1 total speed: 6.25627e+06 eval/sec
Slave 4 total speed: 6.25627e+06 eval/sec
Slave 5 total speed: 6.25627e+06 eval/sec
Slave 2 total speed: 6.25627e+06 eval/sec
```

Naloge

1. Dopolnite program tako, da se bodo najboljša zaporedja v času iskanja pošiljala korenskemu vozlišču. Le ta pa bo med prejetimi zaporedji poiskal najboljše in ga izpisal na zaslon.
2. Dopolnite program tako, da bo vsako vozlišče preskovalo določen iskalni podprostor. Iskalne prostore določite s pomočjo prvih elementov zaporedja.