Druga domaća zadaća

Igra Connect4

1. Implementacija

Upute: U ovom odjeljku potrebno je opisati ključne dijelove funkcionalnosti koristeći isječke programa i snimke zaslona. Obavezno uključite sljedeće elemente s odgovarajućim komentarima:

- *Isječak programa koji prikazuje pripremu poslova na glavnom (master) procesu.*

```
void getAvailableJobs(const vector<vector<char>>& board, int depth, vector<vector<int>>& jobs, vector<int> currentPath = {}) {
    if (depth == 0) {
        jobs.push_back(currentPath);
        return;
    }

    for (int column = 0; column < BOARD_WIDTH; ++column) {
        vector<vector<char>> newBoard = board;
        int player = currentPath.size() % 2 == 0 ? PLAYER : AI;
        pair<int, int> move = makeMove(player, newBoard, column);
        if (move.first == -1) {
            continue;
        }
        currentPath.push_back(column);
        getAvailableJobs(newBoard, depth - 1, jobs, currentPath);
        currentPath.pop_back();
    }
}
```

```
makeMove(PLAYER, board, colNumber);
printBoard(board);
if (checkWin(colNumber, board, PLAYER) == -1) {
    cout << "Player wins!" << endl;
    gameOver = true;
    break;
}

vector<vector<int>> jobs;
getAvailableJobs(board, TASK_DEPTH, jobs);

// cout << "Master generated " << jobs.size() << " jobs" << endl;
map<int, double> results;

if (jobs.size() == 0) {
    cout << "Draw!" << endl;
    gameOver = true;
    break;
}
```

Funkcija getAvailableJobs koristi rekurziju za generiranje svih mogućih nizova poteza do određene dubine. Glavna svrha funkcije je popuniti jobs vektor s nizovima poteza koji predstavljaju sve moguće scenarije igre do zadane dubine. Ovime se osigurava da AI ima popis svih mogućih poteza koji vode do različitih pozicija na ploči, što je korisno za donošenje odluka o najboljem potezu.

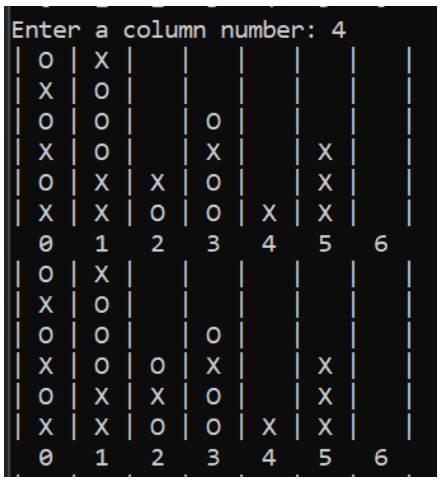
- *Isječke programa koji pokazuju kako se zadaci prenose s glavnog (master) procesa na radničke (worker) procese.*

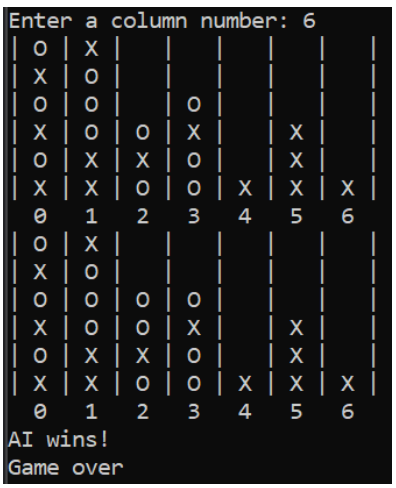
```cpp
if (status.MPI_TAG == WORKERS_REPORTING_FOR_DUTY_TAG) { // Worker is reporting for duty
    cout << "Worker " << status.MPI_SOURCE << " is assigned for duty\n";
    MPI_Recv(nullptr, 0, MPI_INT, status.MPI_SOURCE, WORKERS_REPORTING_FOR_DUTY_TAG, MPI_COMM_WORLD, &status);
    if (jobs.size() > 0) {
        // send up to 100 jobs to workers
        vector<vector<int>> jobsToSend;
        for (int i = 0; i < 100 && jobs.size() > 0; i++) {
            jobsToSend.push_back(jobs.back());
            jobs.pop_back();
        }
        sendJob(board, jobsToSend, status.MPI_SOURCE);
    }
    else {
        unemployedWorkers.push_back(status.MPI_SOURCE);
    }
}
```

```cpp
void sendJob(vector<vector<char>> board, vector<vector<int>> job, int worker) {
    vector<char> flattenBoard = {};
    for (int i = 0; i < BOARD_HEIGHT; ++i) {
        for (int j = 0; j < BOARD_WIDTH; ++j) {
            flattenBoard.push_back(board[i][j]);
        }
    }

    MPI_Send(flattenBoard.data(), BOARD_HEIGHT * BOARD_WIDTH, MPI_CHAR, worker, JOB_ASSIGNED_BOARD_TAG, MPI_COMM_WORLD);

    int jobSize = job.size();
    MPI_Send(&jobSize, 1, MPI_INT, worker, JOB_ASSIGNED_FULL_PATH_SIZE_TAG, MPI_COMM_WORLD);
    for (int i = 0; i < job.size(); i++) {
        int pathSize = job[i].size();
        MPI_Send(&pathSize, 1, MPI_INT, worker, JOB_ASSIGNED_PATH_SIZE_TAG, MPI_COMM_WORLD);
        MPI_Send(job[i].data(), pathSize, MPI_INT, worker, JOB_ASSIGNED_PATH_TAG, MPI_COMM_WORLD);
    }
}
```

Na početku prijenosa prvo pošaljemo trenutno stanje „spljoštene“ ploče zatim šaljemo 100 (ili manje ako ih nema dovoljno) poslova. To radimo na način da prvo pošaljemo broj poslova koji ćemo poslati, a zatim za svaki posao šaljemo njegovu duljinu i zatim njega

- *Snimku zaslona koja prikazuje posljednja dva koraka igre u kojoj računalo pobjeđuje*

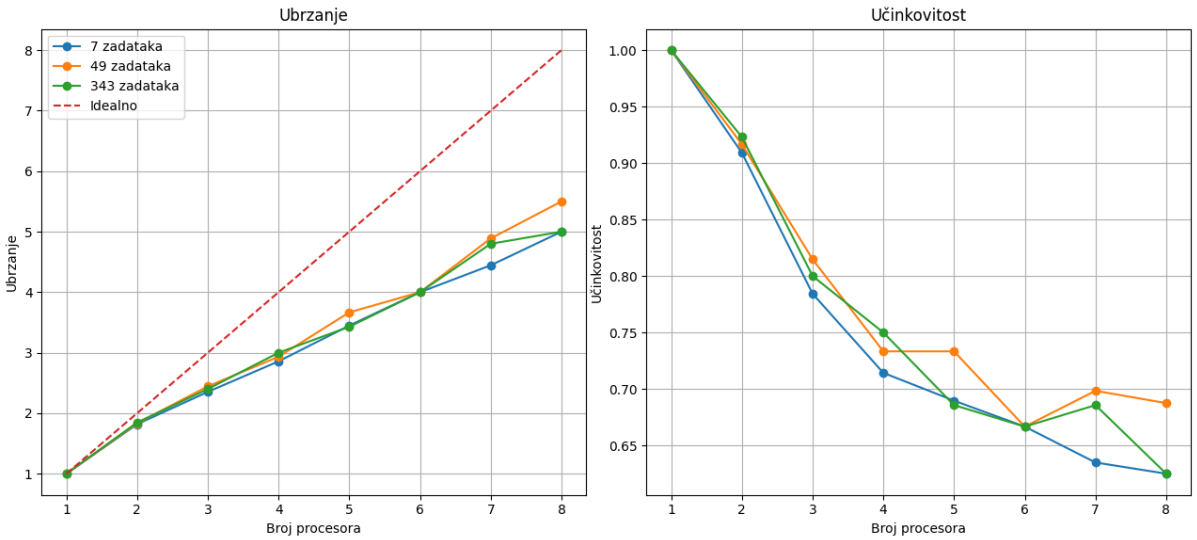




.

2. Kvantitativna analiza

*Upute: U ovom dijelu potrebno je priložiti **tablice s rezultatima mjerenja te grafove ubrzanja i učinkovitosti** za tri različita scenarija: kada paralelni algoritam ima 7, 49 i 343 zadatka (uz aglomeraciju na dubini 1, 2 i 3). Mjerenja treba provesti tako da je najmanje mjereno trajanje (za 8 procesora) reda veličine barem **nekoliko sekundi** (definirajte potrebnu dubinu pretraživanja). Uz grafove, dodajte kratki komentar koji opisuje kako broj zadataka utječe na ubrzanje i učinkovitost (uzevši u obzir utjecaj zrnatosti zadataka, komunikacijskog overhead-a, te udjela programa koji se ne može paralelizirati).*



Ubrzanje

Graf na lijevoj strani prikazuje ubrzanje programa za različite brojeve procesora (P) i različite zadatke (7, 49 i 343 zadataka). Ubrzanje se definira kao omjer vremena izvršavanja s jednim procesorom (T1) i vremena izvršavanja s P procesora (TP).

- **Idealno Ubrzanje:** Idealna crvena isprekidana linija prikazuje idealno ubrzanje, gdje je ubrzanje linearno i proporcionalno broju procesora (S(P) = P).
- **Ubrzanje:** Za sve tri skupine zadataka (7, 49, 343 zadataka), ubrzanje raste s povećanjem broja procesora, ali ne postiže idealno linearno ubrzanje zbog paralelnih režijskih troškova i ograničenja skaliranja.

Učinkovitost

Graf na desnoj strani prikazuje učinkovitost programa za različite brojeve procesora i različite zadatke. Učinkovitost se definira kao omjer ubrzanja i broja procesora (E(P) = S(P) / P).

- **Pad učinkovitosti:** Učinkovitost opada s povećanjem broja procesora za sve skupine zadataka. To je očekivano jer se paralelni režijski troškovi povećavaju s brojem procesora, što smanjuje učinkovitost.
- **Različiti zadaci:** Učinkovitost je veća za zadatke s manje poslova (7 i 49 zadataka) pri manjim brojevima procesora, ali kako broj procesora raste, učinkovitost brže pada zbog ograničenih mogućnosti paralelizacije.

Komentar

Broj zadataka značajno utječe na ubrzanje i učinkovitost paralelnog programa zbog različitih faktora kao što su zrnatost zadataka, komunikacijski overhead i udio programa koji se ne može paralelizirati. Zrnatost zadataka odnosi se na veličinu pojedinačnih zadataka u odnosu na cjelokupni posao. Fini zrnat zadatak znači više manjih zadataka, dok grubi zrnat zadatak znači manje većih zadataka. Veći zadaci imaju manji relativni komunikacijski overhead jer se komunikacija između procesora događa rjeđe. Ovo omogućuje efikasniju upotrebu procesorskih resursa i bolje skaliranje s brojem procesora. Manji zadaci generiraju veći komunikacijski overhead jer se komunikacija između procesora događa češće. Ovo može rezultirati manjim učinkom paralelizacije i bržim opadanjem učinkovitosti s povećanjem broja procesora. Dodatno treba napomenuti da udio programa koji se može paralelizirati ima bitan utjecaj na performanse. Što je taj udio veći, više toga se može izvoditi istovremeno pa tim i brže.