Izračun porazdelitve temperature v 2D prerezih

Projektna naloga pri predmetu Napredna računalniška orodja

Kristijan Danov 23221364, Rade Blagojević 23221382

Januar, 2025

Kazalo vsebine

Kazalo slik

1 Uvod

1.1 Prenos toplote v 2D prerezu

V tem projektu smo obravnavali časovno ustaljen problem prenosa toplote v 2D prerezu, kjer smo za dane robne pogoje izračunali temperaturni gradient prereza. V tem primeru smo predpostavili neodvisno toplotno prevodnost in robne pogoje ter smo zanemarili notranjo generacijo toplote. Tak primer prenosa toplote po trdini je definiran kot

$$\frac{\partial}{\partial x}(k\frac{\partial T}{\partial x}) + \frac{\partial}{\partial y}(k\frac{\partial T}{\partial y}) + q = 0$$

1.2 MKR Metoda reševanja

Za reševanje kopleksnih primerov diferencialnih enačb pogosto posežem k numeričnim metodam saj analitično reševanje ni mogoče. Nekatere od najbolj znanih metod numeričnega reševanja so:

- metoda končnih razlik (MKR)
- metoda končnih elementov (MKE)
- metoda končnih volumnov (MKV)
- metoda robnih elementov (MRE)

Projekt smo naredili spomočno metode končnih razlik. Diferencialno enačbo iskane funkcije v danem prostoru rešujemo numerično tako, da odvode funkcije aproksimiramo s diferenčne sheme.
Pri tej metodi moramo obravnavano območje popisati s strukturirano mrežo pravokotnikov. V robnih točkah pa definiramo vnaprej predpisane pogoje.

1.2.1 Robni pogoji pri aproksimaciji temperature spomočjo metode MKR

Prestop toplote na notranjem kotu

$$2(T_{m-1,n} + T_{m,n+1}) + (T_{m+1,n} + T_{m+1,n-1}) + 2\frac{h\Delta x}{k}T_{ext} - 2(3 + \frac{h\Delta x}{k})T_{m,n} = 0$$

Prestop toplote na robu

$$2(T_{m-1,n} + T_{m,n+1} + T_{m,n-1}) + 2\frac{h\Delta x}{k}T_{ext} - 2(\frac{h\Delta x}{k} + 2)T_{m,n} = 0$$

Prestop toplote na zunanjem kotu

$$2(T_{m-1,n} + T_{m,n-1}) + 2\frac{h\Delta x}{k}T_{ext} - 2(\frac{h\Delta x}{k} + 1)T_{m,n} = 0$$

Toplotni tok na robu

$$(2T_{m-1,n} + T_{m,n-1} + T_{m,n+1}) + 2\frac{q\Delta x}{k} - 4T_{m,n} = 0$$

Vozlišče v notranjosti

$$T_{m-1,n} + T_{m+1,n} + T_{m,n-1} + T_{m,n+1} - 4T_{m,n} = 0$$

1.2.2 Rešitev sistema enačb

Enačb imamo toliko, kot imamo vozlišč. Enačbe u matrični obliki zapišemo

$$[A][T] = [b]$$

oziroma

$$\begin{bmatrix} a_{11} & a_{12} & a_{1N} \\ a_{21} & a_{22} & a_{2N} \\ ... & ... & ... \\ a_{N1} & a_{N2} & a_{NN} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ ... \\ T_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_N \end{bmatrix}$$

2 C++

Ta problem bomo rešili v C++ programskega jezika.

2.1 Deklaracija na spremenljivki, struct Point in KE

Najprej bomo deklarirali pomembne spremenljivke in struct za točke in celice ali mreze.

```
//Struct za tocke in mreze
struct Point{
    int ID;
    double x;
    double y;
    bool isInner=false;
};

struct KE{
    int ID;
    int a;
    int b;
    int c;
    int d;
};

int main() {

    //Matrix A in b
    vector<vector<double>> A;
    vector<double> b;
    vector<double> T;
    double dx = 1.25;
    double dy = 1.25;
    double h = 100;
    double k = 24;
    //Matlab primer dela s ovi
    // double dx = 1;
    // double k = 1;
    const int n = 5754;
    constexpr size_t SIZE = static_cast<size_t>(n);
```

Slika 1: Struct za točke in mreže

2.2 Branje podatkov

Podatki za naš problem so podani v datoteki primer2mreza.txt, katero beremo. Podatke o točkah vstavimo pa v **vector<Point> points**. Iz podatkov mreže pa generiramo adjacency matriko, katera nam pomaga pri določanju okolice za vsako točko.

```
//Preberi tocke
for(int i = 0; i<n; i++){
    getline(infile, temp);
    replace(temp.begin(), temp.end(), ';', ' ');
    replace(temp.begin(), temp.end(), ',', ' ');

    string idTemp;
    string xTemp;
    string yTemp;

    istringstream iss(temp);
    iss >> idTemp;
    iss >> xTemp;
    iss >> yTemp;

    Point tempP;
    tempP.ID = stoi(idTemp);
    tempP.x = stod(xTemp);
    tempP.y = stod(yTemp);

    points.push_back(tempP);
}
```

```
//Preberi stevilo mreze
istringstream issM(temp);
issM >> temp2;
issM >> temp2;
int nMreze = stoi(temp2);

//Preberi mreza i popolni adjacency matrix
while (getline(infile, temp)){
    if (trim(temp).empty()) break;

    replace(temp.begin(), temp.end(), ';', ' ');
    replace(temp.begin(), temp.end(), ',', ' ');

    string idTemp;
    string id1Temp;
    string id2Temp;
    string id3Temp;
    string id4Temp;

    istringstream iss(temp);
    iss >> idTemp;
    iss >> id1Temp;
    iss >> id2Temp;
    iss >> id3Temp;
    iss >> id4Temp;

    int pid = stoi(idTemp);
    int p1 = stoi(id1Temp);
    int p2 = stoi(id2Temp);
    int p3 = stoi(id3Temp);
    int p4 = stoi(id4Temp);

    KE temp;
    temp.ID = pid;
    temp.a = p1;
    temp.b = p2;
    temp.c = p3;
    temp.d = p4;

    elementi.push_back(temp);

    adjM[p1].insert(p2);
    adjM[p1].insert(p4);

    adjM[p2].insert(p1);
    adjM[p2].insert(p3);

    adjM[p3].insert(p2);
    adjM[p3].insert(p4);

    adjM[p4].insert(p1);
    adjM[p4].insert(p3);
}
```

```
vector<int> T1;
vector<int> T2;
vector<int> q3;
vector<int> T4;
vector<int> T5ex;

//Preberi pogoje
//Pogoj 1
while (getline(infile, temp)){
    if (trim(temp).empty()) break;
    int id = stoi(temp);
    T1.push_back(id);
}
```

Slika 2: Branje podatkov iz datoteke primer2mreza.txt

2.3 Preverjanje notranjih točk

Preveriti je potrebno tudi, pozicijo točk, ali je točka notranja, kar nam je pomembno v nadaljevanju pri generaciji matrik A in b. To pa naredimo s kodo na *Slika 3*.

```
//Proveri dali e inner
int index = 0;
for(auto tocka : adjM){
    int counter = 0;
    for(auto elem : tocka){
        counter++;
    }

    if(counter==4){
        points[index].isInner = true;
    }

    index++;
}
```

Slika 3: Preverjanje notranjih točk

2.4 Inicializiranje matrik A, b, T

Za optimizacijo in skrajševanje časa izvedbe programa, najprej inicializiramo A, b, T. T pa polnimo s 100, kar pa je naša začetna predpostavka.

```
//Inicijaliziram A, b, T
vector<double> row;

for(int j=0; j<n; j++){
    row.push_back(0);
}

for (int i=0; i<n; i++){
    T.push_back(100);
    b.push_back(0);
    A.push_back(row);
}
```

Slika 4: Inicializacija matrik A, b, T

2.5 Ispolnjevanje matrik A, b

Za reševanje problema potrebno je matrike A in b isponiti z robnimi pogoji in enačbami za notranje točke.

2.5.1 Pogoj 1

Pogoj 1 - podadene točke imajo temperaturo $400^o C$

```
//Sestavamo A in b
for(int i = 0; i<n; i++){
    if(count(T1.begin(), T1.end(), i) > 0){
        // printf("Bang T1\n");
        A[i][i] = 1;
        b[i] = 400;
```

Slika 5: Pogoj 1

2.5.2 Pogoj 2

Pogoj 2 - podadene točke imajo temperaturo $100^o C$

```
}else if(count(T2.begin(), T2.end(), i) > 0){
    // printf("Bang T2\n");
    A[i][i] = 1;
    b[i] = 100;
```

Slika 6: Pogoj 2

2.5.3 Pogoj 3

Pogoj 3 nam pove da v določenih točkah na robu imamo podan toplotni tok. Za to smo uporabili naslednjo enačbo.

$$(2T_{m-1,n} + T_{m,n-1} + T_{m,n+1}) + 2\frac{q\Delta x}{k} - 4T_{m,n} = 0$$

Točko $T_{m-1,n}$ smo spomočjo isIner ugotovili, če je notranja točka ter smo jo množili z 2, sosednje točke pa smo množili z 1. Kjer je q=0, četrti člen je enak 0.

```
}else if(count(q3.begin(), q3.end(), i) > 0){
    // printf("Bang q3\n");
    for(auto elem : adjM[i]){
        if(points[elem].isInner)
            A[i][elem] = 2;
        else
            A[i][elem] = 1;
    }
    A[i][i]=-4;
    b[i] = 0;
```

Slika 7: Pogoj 3

2.5.4 Pogoj 4

Pogoj 4 - podadene točke imajo temperaturo 600^oC

```
}else if(count(T4.begin(), T4.end(), i) > 0){
    // printf("Bang T4\n");
    A[i][i] = 1;
    b[i] = 600;
```

Slika 8: Pogoj 4

2.5.5 Pogoj 5

Pogoj 5 - Prestop toplote.
Najprej preverimo ali je točka na robu ali pa je notranja. To preverimo s isInner, kar nam poda True za notranje točke. Za notranje točke pa uporabimo sledečo enačbo

$$2(T_{m-1,n} + T_{m,n+1}) + (T_{m+1,n} + T_{m+,n-1}) + 2\frac{h\Delta x}{k}T_{ext} - 2(3 + \frac{h\Delta x}{k})T_{m,n} = 0$$

Za točke na robu moramo uporabiti drugo enačbo, in sicer:

$$2(T_{m-1,n} + T_{m,n+1} + T_{m,n-1}) + 2\frac{h\Delta x}{k}T_{ext} - 2(\frac{h\Delta x}{k} + 2)T_{m,n} = 0$$

```
}else if(count(T5ex.begin(), T5ex.end(), i) > 0){
    // printf("Bang T5\n");
    //preveri rob ali notranjem
    if(points[i].isInner){
        //notranjem kot
        for(auto elem : adjM[i]){
            if(points[elem].isInner)
                A[i][elem] = 2;
            else
                A[i][elem] = 1;
        }
        A[i][i] = -2*(3+(h*dx)/k);

    }else{
        //rob
        for(auto elem : adjM[i]){
            A[i][elem] = 2;
        }
        A[i][i] = -2*(2+(h*dx)/k);
    }

    b[i] = -2*(h*dx*200)/k;
```

Slika 9: Pogoj 5

2.5.6 Notranje točke

Vse točke, ki nimajo podanega nekega pogoja so notranje. Za njih pa velja enačba

$$T_{m-1,n} + T_{m+1,n} + T_{m,n-1} + T_{m,n+1} - 4T_{m,n} = 0$$

```
}else{
    //brez pogoji
        // printf("Inner\n");
        A[i][i] = -4;
        for(auto elem : adjM[i]){
            A[i][elem] = 1;
        }
        b[i] = 0;

}
```

Slika 10: Enačba za notranje točke

2.6 Gauss-Seidl metoda

Pri pojektu smo uporabili Gauss-Seidelovo metodo podobno, kot pri domači nalogi 4, kjer smo paralelizirali kalkulacije po vrsti. V paralelizaciji smo pa uporabili maksimalno število threadov.

```
# ifdef _OPENMP
    cout<<"PARALEL"<<endl;
    int max_thread = omp_get_max_threads();
    omp_set_num_threads(max_thread);
# endif


    double d;
    int ii, jj;
    for (int iitt=0; iitt<1000; iitt++){
        printf("Iteracija no.%d \n", iitt);

        #pragma omp parallel shared(A, b, T) private(jj, ii, d)
        {
            #pragma omp for
            for(jj=0; jj<n; jj++){
                d = b[jj];

                for(ii=0; ii<n; ii++){
                    if(jj!=ii){
                        d = d - A[jj][ii] * T[ii];
                    }

                    T[jj] = d / A[jj][jj];
                }
            }
        }
    }
```

Slika 11: Gauss-Seidl

2.7 Shranjevanje rešitev v VTK format

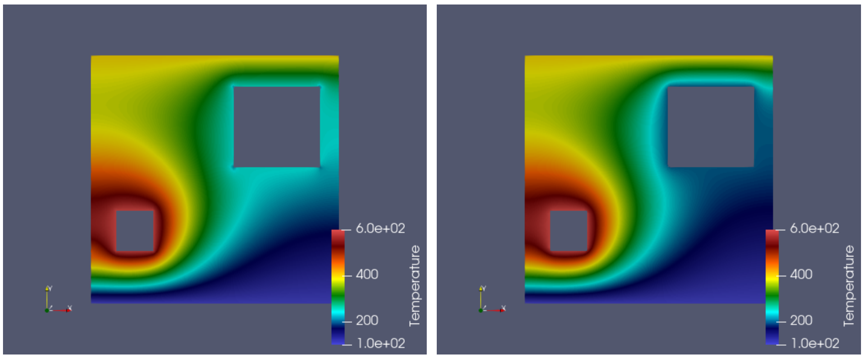
Rešitev smo shranili v .vtk datoteko, da bi jo lahko uvozili v program ParaView ter vizualizirali našo rešitev. Format rezultata je zasnovan na rezultat_vtk.vtk iz primera.

```
ofstream out("rez.vtk");

out<<"# vtk DataFile Version 3.0"<<endl;
out<<"Mesh_1"<<endl;
out<<"ASCII"<<endl;
out<<"DATASET UNSTRUCTURED_GRID"<<endl;
out<<"POINTS "<<n<<" float"<<endl;
for(int i = 0; i<n; i++){
    out<<points[i].x<<" "<<points[i].y<<" 0"<<endl;
}

out<<"CELLS "<<elementi.size()<<" 27580"<<endl;
for(auto el: elementi){
    out<<"4 "<<el.a<<" "<<el.b<<" "<<el.c<<" "<<el.d<<" "<<endl;
}

out<<"CELL_TYPES "<<elementi.size()<<endl;
for(auto el: elementi){
    out<<"9"<<endl;
}

out<<endl<<"POINT_DATA "<<n<<endl;
out<<"SCALARS Temperature float 1"<<endl;
out<<"LOOKUP_TABLE default"<<endl;
for(int i = 0; i<n; i++){
    out<<T[i]<<endl;
}
out<<endl;
out.close();
```

Slika 12: Generacija .vtk datoteke

3 ParaView

Generiran vtk file smo vizualizirali v programu ParaView.
Nalogo smo tudi za primerjavo rešili v MATLAB programskemu jeziku. Opomba: primer rešen v MATLAB-u, v kod je bilo deklarirano, da je dx=1 in k=1, kar v našem primeru ni točno, kajti v našem primeru je dx=1.25 in k=24, zato se rešitvi razlikujeta.
Nalogo smo tudi za primerjavo rešili v MatLab programskemu jeziku.(desno)

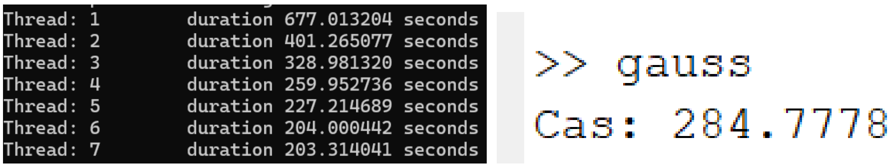


Slika 13: Primerjava rešitev naloge C++ in MATLAB v ParaView

4 Analiza

4.1 Primerjava hitrosti izvajanja prograva v C++ in MATLAB

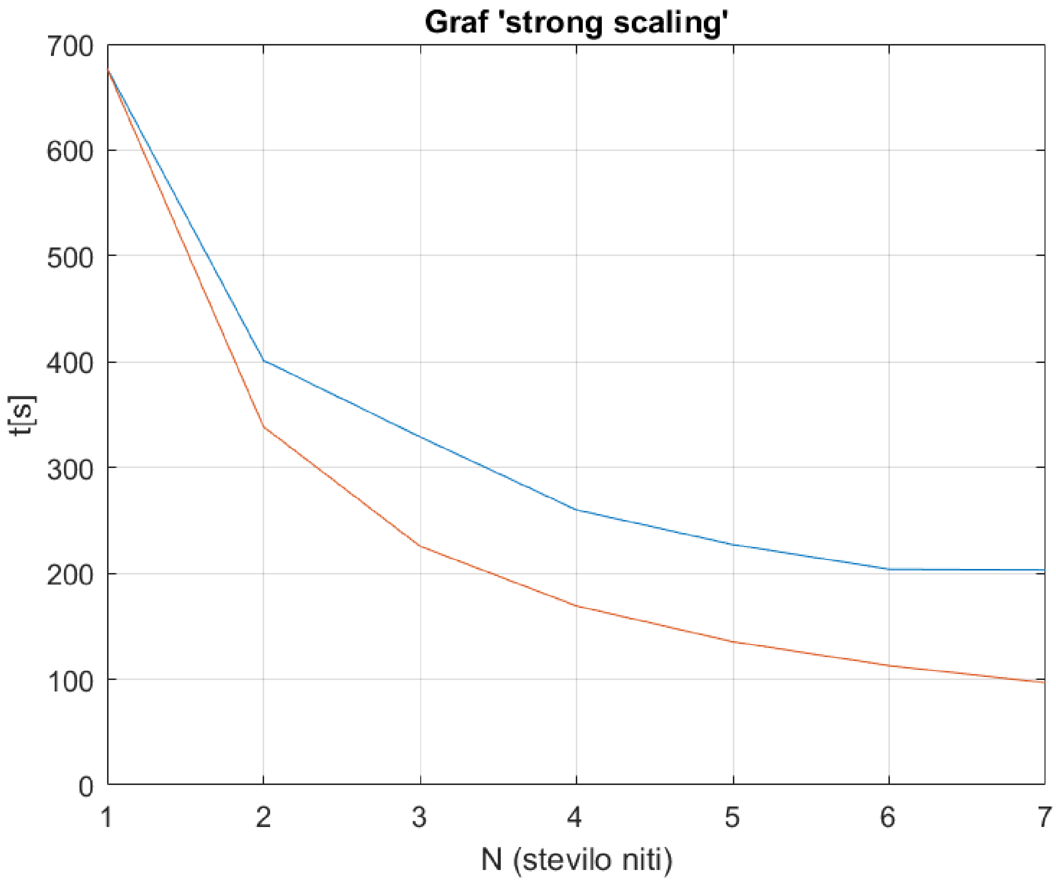
Za Gauss-Seidel metodo smo naredili 1000 iteracij ter istočasno preverili koliko časa potrebujemo za izvedbo. To smo naredili s 1, 2, 3, 4, 5, 6, 7 threadov v C++ in serijski oziroma s 1 thread v MATLAB (kjer Parallel Computing Toolbox ni delal in nismo mogli uporabiti parafor)
Rezultat ki ga dobimo pri serijskem reševanju v C++ je 677s, v MATLAB-u pa je 285s. Iz rezultata pa je razvidno da je MATLAB hitrejši, ker bolj efikasno uporablja memorijo ter ima vgrajeno optimizacijo. Z boljšo optimizacijo našega programa bi lahko v C++ dobili hitreše rezultat, ker je nižji programski jezik in "lightweight".



Slika 14: Primerjava hitrosti programov

4.1.1 Strong scaling

Rezultete časov komputacije različnega števila threadov(niti) lahko primerjamo v grafu strong scaling. Vidimo, da odstopamo od idealne (oranžne) črte, kar je normalno. Naš cilj je, da našo izmerjeno linijo (modro) čim bolj približamo idealni. Ideja idealne linije je, da lahko z uporabo N threadov omogočimo N-krat hitrejše izvajanje našega programa. V resnici to ni mogoče.



Slika 15: Strong scaling graf