UNIVERZA V LJUBLJANI

Fakulteta za strojništvo

Izračun porazdelitve temperature v 2D prerezih

Projektna naloga pri predmetu Napredna računalniška orodja

Avtorja

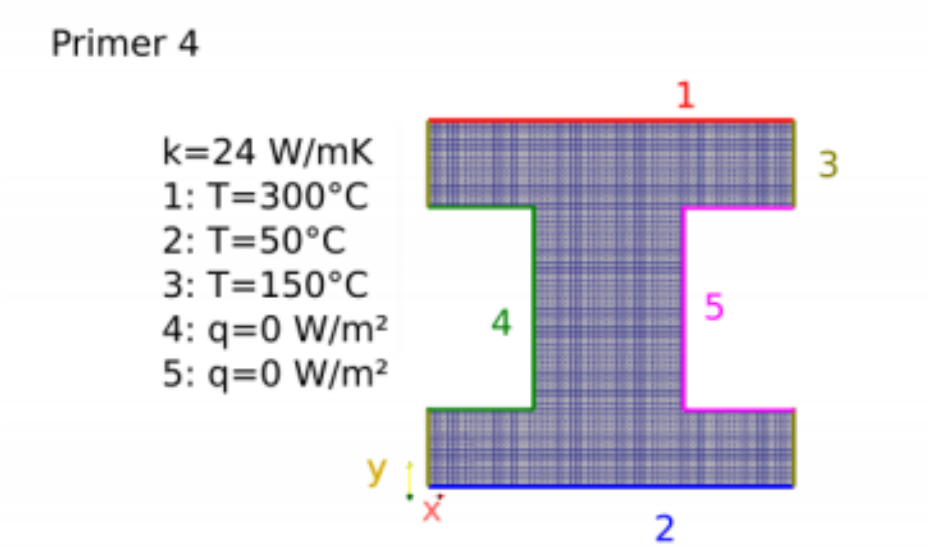
Krajnik Anže 23221333
Krznar Nejc 23221154

18. December 2024

Kazalo

1 Definicija problema

Za podani 2D prerez je potrebno izračunati porazdelitev temperature, ki bo zadoščala podanim robnim pogojem, ki so prikazani na Sliki1.



Slika 1: Robni pogoji za obravnavani prerez

2 Teoretična podlaga reševanja problema

2.1 Sestaljanje sistema enačb

Na robovih 1, 2 in 3 imamo podano vrednost temperature, katero samo pripišemo robnim točkam. Robova 4 in 5 pa sta podana s toplotnim tokom, kjer je bilo treba pogoje na robnih točkah za pisati preko diferenčne enačbe 1 za toplotni tok.

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

Za notranje točke pa uporabimo enačbo 2.

$$T_{m-1,n} + T_{m+1,n} + T_{m,n-1} + T_{m,n+1} - 4T_{m,n} = 0 \tag{2}$$

Na koncu dobimo toliko enačb, kot imamo vozlišč. Ko sestavimo enačbe za vsako vozlišče, dobimo:

$$a_{11}T_1 + a_{12}T_2 + a_{13}T_3 + \cdots + a_{1N}T_N = C_1 \tag{3}$$
$$a_{21}T_1 + a_{22}T_2 + a_{23}T_3 + \cdots + a_{2N}T_N = C_1 \tag{4}$$
$$a_{31}T_1 + a_{32}T_2 + a_{33}T_3 + \cdots + a_{3N}T_N = C_1 \tag{5}$$
$$\vdots$$
$$a_{N1}T_1 + a_{N2}T_2 + a_{N3}T_3 + \cdots + a_{NN}T_N = C_1 \tag{6}$$
$$\tag{7}$$

Enačbe nato zapišemo v matrični obliki,

$$[A][T] = [b] \tag{8}$$

kjer je A matrika koeficientov, T vektor neznanih temperatur in b vektor konstant:

$$[A] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}, \quad [T] = \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_N \end{bmatrix}, \quad [b] = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

Matrično reševanje je bilo izvedeno po Gauss-Seideli metodi.

2.2 Reševanje sistema enačb

Gauss-Seidlova metoda je iterativna metoda za reševanje sistema linearnih enačb.
Vsaka komponenta rešitve T_i se iterativno posodobi z naslednjo enačbo:

$$T_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} T_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} T_j^{(k)} \right),$$

kjer je k indeks iteracije.

Metoda se ponavlja, dokler razlika med zaporednimi iteracijami ne postane manjša od prednastavljene tolerance:

$$\|\mathbf{T}^{(k+1)} - \mathbf{T}^{(k)}\| < \epsilon.$$

3 Izdelava programa

Prvo je bilo reševanje problema razdeljeno na posamezne korake, ki jih mora program opraviti, da iz vhodnih podatkov dobimo rešitev.

1. Branje vhodnih podatkov iz datoteke
2. Sestavljanje matrike koeficientov in vektorja neznank
3. Reševanje sistema enačb
4. Izpis rezultatov v vtk datoteki

3.1 Uporabljene knjižnice

Knjižnica *iostream* je temeljna za standardni vnos in izpis podatkov preko konzole, medtem ko *fstream* omogoča delo z datotekami za trajno shranjevanje podatkov. *Sstream* se uporablja za manipulacijo znakovnih nizov kot podatkovnih tokov in pretvorbe med različnimi tipi, *vector* pa je dinamična podatkovna struktura, ki deluje kot prilagodljiva tabela. *Cmath* vsebuje pomembne matematične funkcije in konstante za različne izračune, *algorithm* ponuja zbirko optimiziranih standardnih algoritmov za urejanje in iskanje, *chrono* se uporablja za natančno merjenje časa in časovno upravljanje v programu, medtem ko *omp.h* omogoča paralelno programiranje za izboljšano učinkovitost z izkoriščanjem več procesorskih jeder.

3.2 Branje vhodnih podatkov iz datoteke

V prvem delu program prebere vhodne podatke iz podane datoteke primer4mreza.txt. Datoteka vključuje x in y koordinate vozlišč, celice ter robne pogoje v posameznem robnem vozlišču. Preden preberemo vhodne podatke, definiramo spremenljivke in vektorje v katere bomo shranili prebrane podatke. Nato datoteko odpremo z uporabo `std::ifstream`, kjer preberemo željene vrstice z `std::getline`. Podatki o vozliščih (ID, koordinati `x` in `y`) se razčlenijo z uporabo `std::istringstream` in shranijo v vektorja `X` in `Y`. Podobno se podatki o celicah, ki vključujejo ID-je pripadajočih vozlišč, shranijo v vektor `celice`. Robni pogoji se preberejo glede na njihov tip (temperatura ali toplotni tok) in se shranijo v `vrednosti_robnih_pogojev`, medtem ko se ID-ji pripadajočih vozlišč shranijo v `vozlisca_robnih_pogojev`.

3.3 Sestavljanje matrike koeficientov in vektorja konstant

Preden sestavimo sistem enačb moramo vsakemu vozlišču določiti sosede, ki jih bomo uporabili za formiranje diferenčnih enačb.
Postopek za določanje sosednosti vozlišč poteka po naslednjih korakih:

1. **Ustvarimo vektor sosedov:** Za vsako vozlišče (`node_id`) se ustvari vektor `node_i_neighbours`, ki vsebuje štiri pozicije. Na začetku so vrednosti privzete na -1.
2. **Izračun položaja sosednjih vozlišč:** Relativni položaj soseda glede na trenutno vozlišče se določi na podlagi razlik v x in y koordinatah.
3. **Shranjevanje sosednjih vozlišč:** Če je položaj veljaven, se identifikator sosednjega vozlišča shrani v `node_i_neighbours`.
4. **Shranjevanje rezultata:** Po preverjanju vseh celic se vektor sosedov shrani v glavni vektor `sosednja_vozlisca`.

Za numerično reševanje problema koda sestavi matriko sistema A in vektor b po naslednjih korakih:

1. **Inicializacija:** Matrika A dimenzij $n \times n$ in vektor b dolžine n se ustvarita z vrednostmi 0.
2. **Podatki o sosednjih vozliščih:** Iz vektorja `sosednja_vozlisca` pridobimo podatke o sosednjih vozlišč.
3. **Notranja vozlišča:** Če ima vozlišče vse štiri sosede, se v matriko A zapišejo vrednosti, ki ustrezajo diferenčni shemi enačbe za notranje vozlišče
4. **Robna vozlišča:** Vozlišča na robu mreže se obravnavajo posebej glede na tip robnega pogoja:
    - **Dirichletov pogoj (Poznana T):** Na diagonali matrike A se zapiše 1.0, vektor b pa dobi vrednost robnega pogoja.

- **Neumannov pogoj (Poznan q):** Matrika A in vektor b se prilagodita, da vključita vplive robnih pogojev v skladu z diferenčno shemo enačbe za tolpotni tok.

Na koncu matriko A in vektor b shranimo v `.csv` datoteki za reševanje sistema enačb z uporabo MATLAB

3.4 Reševanje sistema enačb z Red-Black Gauss-Seidlovo metodo

Koda uporablja **Red-Black Gauss-Seidlovo metodo** za iterativno reševanje sistema linearnih enačb $A \cdot T = b$, kjer iščemo vrednosti vektorja T. Metoda je pospešena s paralelizacijo z OpenMP, kar omogoča hkratno izvajanje na več procesorskih jederih. Postopek poteka po naslednjih korakih:

1. **Inicializacija**
   - Ustvari se začetni vektor T, kjer so vsi elementi nastavljeni na vrednost 100.
   - Nastavi se število iteracij (v tem primeru 1000), ki jih bomo izvedli.
2. **Paralelizacija**
   - Uporablja se 10 procesorskih jeder, ki jih nastavimo z ukazom `omp_set_num_threads(10)`.
   - Omogočeno je dinamično dodeljevanje nalog med različna jedra z `omp_set_dynamic(1)`.
3. **Iterativno reševanje**
   Uporabili smo Red-Black Gauss-Seidlovo metodo. Ta razdeli nalogo na dva ločena koraka (rdeče in črne vozle), kar omogoča bolj učinkovito paralelno računanje. Paralelizacija z `OpenMP` zagotavlja, da se posodobitev vsakega vozla (rdečega ali črnega) izvede neodvisno, kar zmanjšuje čas izvajanja in povečuje zmogljivost pri večprocesorskih sistemih.

   (a) **Rdeči vozli (parni indeksi):** Na začetku vsake iteracije se najprej posodobi $T[j]$ za vse parne indekse ($j\%2 = 0$). Posodobitev vrednosti uporablja elemente matrike A, vektorja b in trenutni vektor T (iz prejšnje iteracije).

```
#pragma omp parallel for shared(A, b, T, n)
for (int jj = 0; jj < n; jj++) {
    if (jj % 2 == 0) { // Rdeči vozli
        double d = b[jj];
        for (int ii = 0; ii < n; ii++) {
            if (jj != ii) {
                d -= A[jj][ii] * T[ii];
            }
        }
        T[jj] = d / A[jj][jj];
    }
}
```

(b) **Črni vozli (neparni indeksi):** V drugi fazi iteracije se posodobi $T[j]$ za vse neparne indekse ($j\%2 \neq 0$).

```
#pragma omp parallel for shared(A, b, T, n)
for (int jj = 0; jj < n; jj++) {
    if (jj % 2 != 0) { // Črni vozli
        double d = b[jj];
        for (int ii = 0; ii < n; ii++) {
            if (jj != ii) {
                d -= A[jj][ii] * T[ii];
            }
        }
        T[jj] = d / A[jj][jj];
    }
}
```

Po vsaki iteraciji se T posodobi za naslednji cikel iteracij.

4. **Merjenje časa**
   - Izmeri se čas izvajanja za vseh 1000 iteracij.
   - Na koncu se izpiše skupni čas, ki je bil potreben za rešitev sistema enačb.

3.5 Izpis rezultatov v vtk datotek

Koda zapisuje rezultate simulacije v datoteko formata VTK, ki omogoča vizualizacijo podatkov o mreži in temperaturnih vrednostih v orodjih, kot je ParaView. Postopek zapisovanja poteka po naslednjih korakih:

1. **Ustvarjanje in odpiranje datoteke:**
   - Ustvari se nova datoteka z imenom `rezultat_vtkcpp.vtk`.
2. **Zapis osnovnih informacij o mreži:**
   - V glavo datoteke se zapiše različica VTK in uporabljen format podatkov (ASCII).
   - Podatki o vozliščih (`POINTS`): Zapiše se število vozlišč in njihove (x, y) koordinate. Vse točke so v 2D ravnini, pri čemer je $z = 0$.
3. **Zapis celic:**
   - Opisane so povezave med vozlišči (`CELLS`):
     - Zapiše se število celic.
     - Za vsako celico se navedejo identifikatorji (indeksi) štirih povezanih vozlišč.

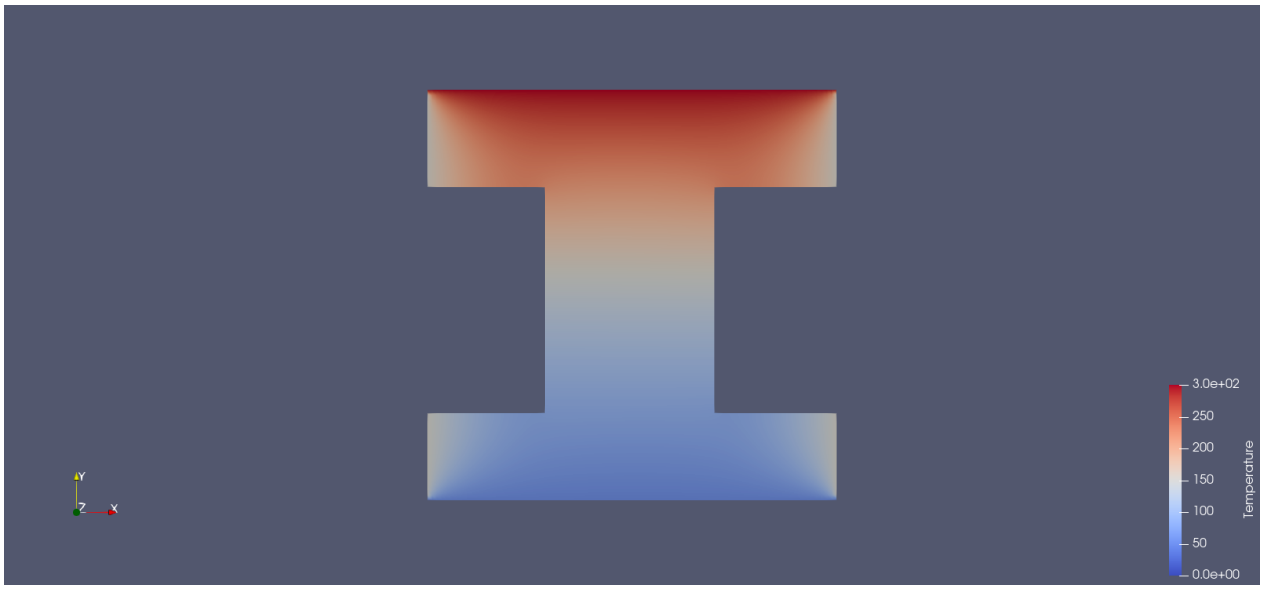
4. **Zapis tipov celic:**
    - Za vsako celico se določi njen tip, ki je `9` (kvadratna celica v VTK formatu).
5. **Zapis temperaturnih podatkov:**
    - Na koncu se za vsako vozlišče zapiše vrednost temperature.
    - Temperaturni podatki so zapisani kot skalarni podatki (`SCALARS`) in so povezani z vozlišči mreže.
6. **Zaključek:**
    - Po končanem zapisovanju se datoteka zapre.

Datoteka `rezultat_vtkcpp.vtk` tako vsebuje vse potrebne informacije za vizualizacijo mreže in temperaturnih vrednosti, kar omogoča nadaljnjo analizo v programski opremi, ki podpira VTK format, kot je ParaView.
Rezultat metode je rešen sistem enačb, kjer je vektor T rešitev problema (npr. razporeditev temperatur v mreži).

4 Rezultati

4.1 Prikaz rezultatov v programu ParaView

Na sliki 2 je prikazana izračunana porazdelitev temperature. Po razpravi v timu smo ugotovili, da je rešitev smiselna. Robni pogoji 1, 2 in 3 določajo vrednosti temperature v vozliščih, kar je jasno razvidno na sliki 2. Prav tako je razvidno, da sta na robovih 4 in 5 toplotna toka enaka nič. Zaradi robnih pogojev 4 in 5 imajo vozlišča na teh robovih enake temperature, kot vozlišča proti notranjost prereza, kar je prav tako potrjeno na sliki 2.



Slika 2: Dobljena porazdelitev temperature po 2D preseku

4.2 Optimizacija in paralelizacija

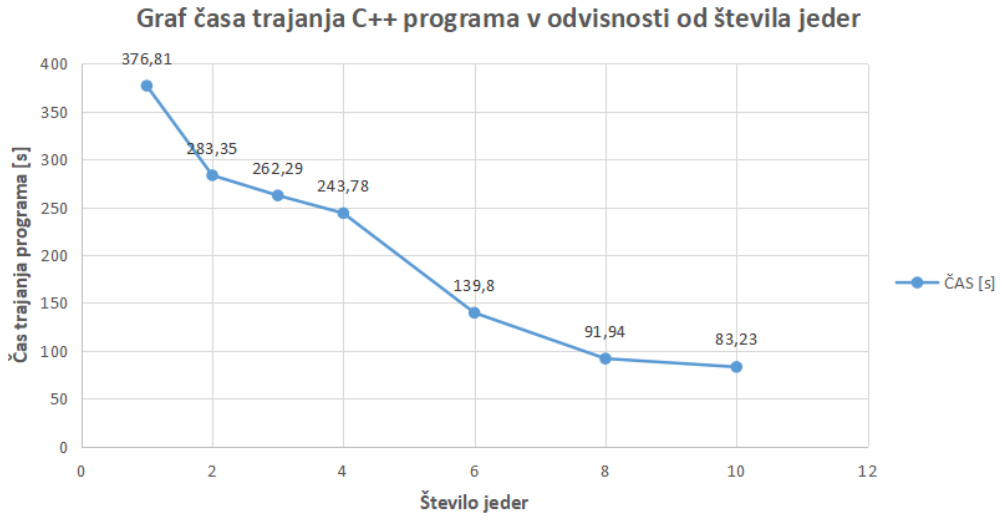
Po uspešnem izračunu rešitve, je bila izvedena optimizacija programa ter vključitev paralelnega računanja z uporabo knjižnice `omp.h` in funkcije `omp_set_num_threads(10)`, kjer definiramo število niti.

4.2.1 Optimizacija

Prvo je bila uporabljena **Gauss-Seidlovo metoda**, s katero smo dobili čas reševanja **248,488 sekund**. Po poizvedbi je bilo ugotovljeno, da osnovna Gauss-Seidlova metoda ni primerna tehnika za paralelno računanje. V ta namen se je uporabilo Red-Black metodo, ki omogoča paralelizacijo iterativnih metod, kot je Gauss-Seidlova metoda. Nov čas reševanja z uporabo paralelnega računanja je znašal **64,0185 sekund**, kar je približno štirikrat krajši čas reševanja.

4.2.2 Vpliv hitrosti izračuna glede na število niti

Testirali smo tudi vpliv števila uporabljenih niti na čas računanja. V programu smo postopno zviševali število aktivnih niti (`omp_set_num_threads(10)`) ter dobljene čase prikazali v grafu, ki je prikazan na sliki 3.



Slika 3: Graf odvisnosti časa reševanja sistema enačb od števila jeder

Pri preprostih težavah ali nalogah z majhnim številom iteracij lahko stroški upravljanja niti (kot so njihovo ustvarjanje, sinhronizacija in preklapljanje) presežejo prednosti paralelizacije, kar pomeni da se čas računanja poveča z večjim številom niti. V našem primeru tega problema ni bilo, kot je tudi razvidno iz grafa 3. Čas računanja našega problema se je zmanjševal s povečevanjem števila niti.

4.3 Reševanje problema z MATLAB

S shranjenima matriko A in vektorjem b je bil problem rešen z uporabo programa MATLAB. Uporabljena je bila vgrajena funkcija `gmres` *(Generalized Minimal Residual)*, doblenjeni čas reševanja pa je prišel **0.595437 sekunde**, kar je v primerjavi z uporabljeno metodo v C++ približno 100-krat krajši čas reševanja. Rešitev v MATLAB se je v ujemala z rešitvijo dobljeno v C++ z izjemo v rahlih odstopanjih vrednosti temperatur v sredini območja.