Steady-state heat distribution

Gregor Kerševan in Miha Jamšek

Junij 2020

1 Uvod

Cilj projektne naloge je bil implementirati paralelni algoritem za porazdelitev toplote v mirnem stanju. Rezultat algoritma naj bi bila mreža, prikazana na sliki 1.

0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
100,00	50,30	30,85	21,87	17,12	14,34	12,63	11,56	10,93	10,63	10,63	10,93	11,56	12,63	14,34	17,12	21,87	30,85	50,30	100,00
100,00	70,37	51,21	39,51	32,25	27,62	24,61	22,67	21,51	20,97	20,97	21,51	22,67	24,61	27,62	32,25	39,51	51,21	70,37	100,00
100,00	79,95	64,12	52,72	44,77	39,27	35,51	33,02	31,49	30,76	30,76	31,49	33,02	35,51	39,27	44,77	52,72	64,12	79,95	100,00
100,00	85,31	72,59	62,49	54,83	49,19	45,15	42,39	40,66	39,83	39,83	40,66	42,39	45,15	49,19	54,83	62,49	72,59	85,31	100,00
100,00	88,70	78,45	69,80	62,86	57,51	53,53	50,72	48,94	48,08	48,08	48,94	50,72	53,53	57,51	62,86	69,80	78,45	88,70	100,00
100,00	91,05	82,71	75,41	69,32	64,45	60,72	58,04	56,31	55,46	55,46	56,31	58,04	60,72	64,45	69,32	75,41	82,71	91,05	100,00
100,00	92,77	85,94	79,81	74,56	70,25	66,88	64,41	62,79	61,99	61,99	62,79	64,41	66,88	70,25	74,56	79,81	85,94	92,77	100,00
100,00	94,10	88,47	83,34	78,86	75,11	72,13	69,92	68,46	67,73	67,73	68,46	69,92	72,13	75,11	78,86	83,34	88,47	94,10	100,00
100,00	95,15	90,50	86,22	82,43	79,22	76,63	74,69	73,39	72,75	72,75	73,39	74,69	76,63	79,22	82,43	86,22	90,50	95,15	100,00
100,00	96,01	92,17	88,60	85,42	82,70	80,48	78,80	77,68	77,12	77,12	77,68	78,80	80,48	82,70	85,42	88,60	92,17	96,01	100,00
100,00	96,72	93,56	90,61	87,95	85,67	83,80	82,37	81,42	80,93	80,93	81,42	82,37	83,80	85,67	87,95	90,61	93,56	96,72	100,00
100,00	97,32	94,73	92,31	90,12	88,23	86,67	85,48	84,68	84,27	84,27	84,68	85,48	86,67	88,23	90,12	92,31	94,73	97,32	100,00
100,00	97,84	95,75	93,78	92,00	90,46	89,18	88,20	87,54	87,21	87,21	87,54	88,20	89,18	90,46	92,00	93,78	95,75	97,84	100,00
100,00	98,29	96,63	95,07	93,65	92,42	91,40	90,61	90,08	89,81	89,81	90,08	90,61	91,40	92,42	93,65	95,07	96,63	98,29	100,00
100,00	98,69	97,41	96,21	95,12	94,17	93,38	92,77	92,36	92,15	92,15	92,36	92,77	93,38	94,17	95,12	96,21	97,41	98,69	100,00
100,00	99,05	98,12	97,25	96,46	95,76	95,19	94,74	94,44	94,29	94,29	94,44	94,74	95,19	95,76	96,46	97,25	98,12	99,05	100,00
100,00	99,38	98,78	98,21	97,69	97,24	96,86	96,57	96,37	96,27	96,27	96,37	96,57	96,86	97,24	97,69	98,21	98,78	99,38	100,00
100,00	99,69	99,40	99,12	98,86	98,64	98,45	98,31	98,21	98,16	98,16	98,21	98,31	98,45	98,64	98,86	99,12	99,40	99,69	100,00
100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00

Figure 1: Rezultat mreže 20x20

2 Sekvenčni algoritem

Za testiranje pravilnosti algoritma smo najprej spisali sekvenčno različico algoritma in testirali izvajanje tega algoritma na mrežah velikost 100x100 in 500x500 (tabela 1). Mreža 100x100 se relativno hitro izračuna, medtem ko mreža 500x500 porabi kar 3 minute in pol za izračun.

Program za argumente sprejme višino in širino naše celotne plošče, ter višino in širino posamezne ploščice. Kot zadnji argument podamo še epsilon vrednost, ki določi kdaj se algoritem ustavi - ko se v eni iteraciji temperatura spremeni manj kot za epsilon, algoritem prekine izvajanje.

Izvorna koda sekvenčnega programa je na voljo v sequential/main.cpp

Velikost tabele	velikost ploščic	t_s
100x100	1x1	2.309 s
500x500	1x1	334.914 s

Table 1: Čas izvajanja sekvenčnega algoritma

Velikost tabele	velikost ploščic	Št. procesov	t_s
100x100	1x1	1	2.465 s
500x500	1x1	1	435.8997 s
100x100	1x1	2	1.5392 s
500x500	1x1	2	223.033 s
100x100	1x1	8	0.527 s
500x500	1x1	8	67.343 s
100x100	1x1	32	0.494 s
500x500	1x1	32	19.326 s

Table 2: Čas izvajanja algoritma z openMP

3 Paralelizem z openMP

Sekvenčni algoritem smo nato prilagodili za uporabo z openMP. Najprej smo paralelizirali nastavljanje začetnih vrednosti, čeprav pri merjenju časa tega nismo upoštevali, saj smo merili samo čas izvajanja algoritma, ne pa predpriprave podatkov.

Nato smo izvajali algoritem, precej podoben sekvenčnemu, le da smo funkcionalnosti, kot je kopiranje tabele, računanje nove vrednosti in ugotavljanje, ali je pogoj za ustavitev izpolnjen, izvajali paralelno.

Ta algoritem smo nato izvedli z dvema različnima velikosti mreže - 100x100 in 500x500, ter z uporabo različnega števila procesov - 1, 2, 8 in 32. Rezultati so vidni v tabeli 2. Opazimo, da pri manjši mreži večje število jeder ne prinaša večje izboljšave v hitrosti.

4 Paralelizem z MPI

4.1 Delitev po vrsticah

Simulacije porazdelitve po vrsticah smo poganjali do 20.000 iteracij, namesto da bi čakali da doseženo napako EPS (sprememba vrednosti polja med prejšnjo in sedanjo iteracijo). To pa zato, ker so nekateri procesi na plošči velikosti 512x512 dosegli EPS že pri 4.000-ti iteraciji, nekateri pa šele pri 17.000-ti. Rezultati so vidni v tabeli 3. Pri implementaciji smo privzeli, da je število vrstic plošče deljivo s številom procesov. S tem dosežemo, da vsak proces obdeluje enako število vrstic.

Pri testiranju z 64 procesi na dveh vozliščih smo prejeli zelo nenavadne rezultate - 160 s in 35 s. Razlog za to bi lahko bila kakšna težava v sistemu ali kaj podobnega, saj sta časa zelo različna od primerljivih simulacij.

V prihodnje bova dodala še ustavitev ob doseženi vrednosti EPS ter implementacijo z neblokirajočo komunikacijo.

4.2 Delitev po stolpcih

Ni še končano.

Velikost tabele	velikost ploščic	Št. procesov	t_s
96x96	1x1	1	2.189 s
512x512	1x1	1	71.062 s
512x512	1x1	2	48.676 s
512x512	1x1	8	10.453 s
512x512	1x1	32 - 1 vozlišče	3.500 s
512x512	1x1	32 - 2 vozlišči	3.817 s

Table 3: Čas izvajanja delitve po vrsticah MPI

4.3 Delitev s ploščicami

Ni še končano.

5 Paralelizem z openMP in MPI

Ni še končano. Pri tej implementaciji nameravava implementirati zunanje ogrodje z MPI, nato pa bi vsak proces izvedel računanje z OpenMP.

6 Zaključek

Nekaj vprašanj:

- Naj izvajanje raje ustaviva po določenem številu iteracij ali ob zelo majhnih spremembah - doseženi vrednosti EPS? V podani psevdokodi izgleda, kot da je cilj simulirati določeno število iteracij, vendar ni smiselno simulirati v nedogled.
- Se da ustaviti izvajanje procesov pri MPI implementaciji brez uporabe neblokirajočih funkcij? V zdajšnji implementaciji bi namreč morali vsi procesi izvesti enako število iteracij zaradi SendRecv. Možna rešitev bi bila upraba $MPI\_Isend, MPI\_Irecv, MPI\_Test$, s čimer bi en proces vsem ostalim sporočil koliko iteracij naj opravijo.
- Naj raje implementirava še MPI verziji po stolpcih in s ploščicami, ali se raje posvetiva analizi Infiniband/Ethernet in blocking/non-blocking komunikaciji?