Semestrální projekt NI-PDP

Paralelní algoritmus pro řešení problému: *Úloha SAJ — střelec a jezdec na šachovnici*

Jan Bittner

Magisterské studium, FIT ČVUT, Thárukova 9, 160 00 Praha 6

10. května 2021

Definice problému a popis sekvenčního algoritmu

Úkolem semestrálního projektu je najít řešení pro problém úlohy střelec a jezdec na šachovnici. Tento problém spočívá v odstranění všech figurek pešce pomocí minimálního počtu tahů figurek střelce a jezdce na šahovnici, zatímco se střelec a jezdec střídají a pešci zůstávájí na místě.

Algoritmus na vstupu obdrží přirozené číslo k, $20 > k > 5$, které reprezentuje délku strany šachovnice, přirozené číslo q, které reprezentuje maximální hloubku zanoření a pole polí šachovnice, kde jsou vyznačeni pěšci, střelec a jezdec.

Výstupem algoritmu je nejkratší posloupnost střídavých tahů střelce a jezdce vedoucí do cílové konfigurace (počet pěsců je roven 0).

Popis sekvenčního algoritmu

Sekvenční algoritmus je typu BnB DFS a je implementovaný pomocí rekurze, kde je vstupem aktuální hloubka a šachovnice. Funkce nejprve ověří podmínky, zda-li nebylo nalezeno vyhovující řešení (a zda-li je lepší, než současně nejlepší) a nebo naopak, zda-li je nevyhovující. Následně je sestavený seznam kroků, které figurka na tahu (určuje se podle aktuální hloubky) může provést. Pomocí těchto kroků se vytvoří nové konfigurace, které se předají do rekurzivní funkce. Jako cenu minimalizujeme počet tahů, které vedou do koncové konfigurace. Algoritmus končí, když se cena rovná dolní mezi (počet pěšců na šachovnici) nebo prohledáním celého stavového prostoru. Horní mez na cenu je `2*k^2`.

Popis paralelního algoritmu a jeho implementace v OpenMP — taskový paralelismus

Algoritmus taskového paralelismu pomocí OpenMP funguje totožně s sekvenčním algoritmem, avšak rekurzivní funkce pro hloubku menší než 35 % q (maximální hloubka zanoření) rozděluje práci disjunktních úkolů mezi více vláken. Jakmile je funkce zavolána s vyšší hodnotou aktuální hloubky, zavolá se sekvenční rekurzivní funkce, která řeší zbývající průchod stavového prostoru z dané konfigurace.

Kritickou sekcí algoritmu je porovnání a zápis nového nejlepšího řešení, které je uloženo ve sdílené proměnné.

Popis paralelního algoritmu a jeho implementace v OpenMP — datový paralelismus

Algoritmus datového paralelismu pomocí OpenMP spočívá ve předgenerování počátečních stavů pomocí algoritmu BFS. Generuje se až `std::thread::hardware_concurrency() * 1000` unikátních stavů s podmínkou maximální aktuální hloubky rovné 6. Vygenerované stavy jsou následně postupně rozděleny ke zpracování mezi několik vláken. Kritická sekce je stejná jako u taskového paralelismu.

Popis paralelního algoritmu a jeho implementace v MPI

Paralelní algoritmus pomocí MPI spočívá v rozšíření paralelního řešení tak, aby fungovalo i na klastry s distribuovanou pamětí. Byl využit datový paralelismus pomocí OpenMP.

Algoritmus využívá několik procesů. Jeden master proces a několik slave procesů. Master proces předgeneruje pomocí algoritmu BFS `5 * početProcesů` stavů, které postupně posílá slave procesům. Slave proces po obdržení stavu vygeneruje další stavy pomocí datového paralelismu OpenMP a po ukončení algoritmu pošle nejlepší cenu a k tomu náležité tahy zpět master procesu. Master proces postupně předává stavy, aktualizuje nejlepší hodnoty, a následně vypíše nejlepší řešení.

Naměřené výsledky a vyhodnocení

Pro implementaci byl použit jazyk C++. Na zachycení aktuálního času bylo využito `std::chrono::high_resolution_clock::now()`. Měření probíhalo na výpočetním klastru Star FIT ČVUT.

Tabulky ukazují naměřený čas řešení a zrychlení oproti sekvenčnímu času, podle vzorce `sekvencni_cas / (paralelni_cas * pocet_vlaken * pocet_procesu)`.

Specifikace výpočetního klastru Star:

- CPU Model: Intel® Xeon® CPU E5-2630 v4 @ 2.20GHz
- #CPU jader: 20 (možno až 40 se zapnutým hyperthreadingem)
- #sockets, #cores per socket, #threads per socket: (2, 10, 1 (2))
- CPU Cache L1 - L2 - L3: 32KB - 256KB - 25600KB

Instance

Pro účely měření na výpočetním klastru Star, aby byly časy sekvenčního řešení mezi 1 a 10 minutami, byly zvoleny či modifikovány instance dodané pro úlohy NI-PDP.

Níže jsou zobrazeny používané konfigurační soubory instancí, které byly využity při měření na výpočetním klastru Star.

sajA

```
11
20
-----------
-------P---
P-----P--P-
---PJS--P--
----P------
---P--P----
-----------
-----------
------P----
-----P-----
-----------
```

sajB

```
11
18
-----------
-------P---
P-----P--P-
---PJS--P--
----P------
---P--P----
------P----
------P----
-----------
-----------
-----------
```

sajC

```
13
21
-JPP---------
-PPP---------
-PPP---------
-PPP---------
----P--------
-----P-------
------S------
-------P-----
--------P----
---------P---
-------------
-------------
-------------
```

Měření sekvenčního řešení

Nejprve byly naměřeny časy sekvenčního řešení.

Měření zobrazuje tabulka 1.

Table 1. Sekvenční řešení

file	time [s]
sajA	333
sajB	273
sajC	200

Měření paralelního řešení

Následně bylo provedeno měření OpenMP taskového a datového paralelismu. Čas byl testován na různém počtu vláken. Jak již bylo zmíněno, vzorec pro zrychlení je `sekvencni_cas / (paralelni_cas * pocet_vlaken)`.

Průměrné zrychlení u taskového paralelismu bylo 176.258, u datového paralelismu 77.753.

Měření zobrazují tabulky 2 a 3.

Table 2. Paralelní řešení pomocí taskového paralelismu s OpenMP

file	# thread	time [s]	speed-up
sajA	2	2.861	58.196
	4	0.176	473.011
	8	0.061	682.377
	10	0.026	1280.769
	15	0.162	137.037
	20	0.041	406.098
sajB	2	12.000	11.375
	4	1.673	40.795
	8	1.332	25.619
	10	1.279	21.345
	15	1.339	13.592
	20	1.131	12.069
sajC	2	22.000	4.545
	4	20.000	2.500
	8	20.000	1.250
	10	20.000	1.000
	15	22.000	0.606
	20	22.000	0.455

Table 3. Paralelní řešení pomocí datového paralelismu s OpenMP

file	# thread	time [s]	speed-up
sajA	2	0.430	387.209
	4	0.256	325.195
	8	0.271	153.598
	10	0.349	95.415
	15	0.163	136.196
	20	0.156	106.731
sajB	2	1.728	78.993
	4	1.475	46.271
	8	1.331	25.639
	10	1.336	20.434
	15	1.723	10.563
	20	1.766	7.729
sajC	2	46.000	2.174
	4	37.000	1.351
	8	34.000	0.735
	10	34.000	0.588
	15	32.000	0.417
	20	31.000	0.323

Měření MPI

Na závěr bylo provedeno měření MPI. Měření bylo provedeno za využití 4 procesů. Vzorec pro zrychlení je znovu `sekvencni_cas / (paralelni_cas * pocet_vlaken * pocet_procesu)` .

Průměrné zrychlení u MPI bylo 11.726.

Měření zobrazuje tabulka 4.

Table 4. Paralelní řešení pomocí MPI

file	# thread	time [s]	speed-up
sajA	2	40.000	1.041
	4	0.462	45.049
	8	0.446	23.332
	10	0.464	17.942
	15	0.477	11.635
	20	0.444	9.375
sajB	2	0.857	39.819
	4	0.719	23.731
	8	0.695	12.275
	10	0.678	10.066
	15	0.669	6.801
	20	0.687	4.967
sajC	2	12.000	2.083
	4	10.000	1.250
	8	10.000	0.625
	10	11.000	0.455
	15	9.687	0.344
	20	9.116	0.274

Závěr

Byly implementovány 4 řešení úlohy. Sekvenční řešení, které naivně prochází stavový prostor a hledá řešení. Taskový paralelismus pomocí OpenMP, který rozděluje práci disjunktních úkolů mezi více vláken. Datový paralelismus pomocí OpenMP, který předgenerovává stavy, které jsou následně rozděleny ke zpracování mezi několik vláken. A paralelismus pomocí MPI, které rozšiřuje paralelní řešení tak, aby fungovalo i na klastry s distribuovanou pamětí.

Z hlediska zrychlení (podle vzorce `sekvencni_cas / (paralelni_cas * pocet_vlaken * pocet_procesu)`) se ukázal taskový paralelismus jako nejefektivnější, s průměrným zrychlením 176.258. Následuje datový paralelismus s průměrným zrychlením 77.753 a na závěr MPI řešení s průměrným zrychlením 11.726.

Z hlediska nejmenšího času se ukázalo nejlepší řešení využávající MPI. MPI mělo na úlohách sajA, sajB a sajC průměrný čas 6.022 s, průměrný medián úloh 3.718 s. Dále se umístilo řešení taskového paralelisku s průměrným časem 8.227 s, průměrný medián úloh 7.482 s. Řešení datovým paralelismem má průměrný čas 12.499 s, průměrný medián úloh 11.954 s.

Last updated 2021-05-11 19:28:51 +0200