ELEKTROTEHNIŠKI VESTNIK 79(1-2): 19–24, 2012
EXISTING SEPARATE ENGLISH EDITION

Implementacija modula r.cuda.los v odprtokodnem paketu GRASS GIS z vzporednim računanjem na grafičnih karticah NVIDIA CUDA

Andrej Osterman

Telekom Slovenije d.d., Cigaletova 15, 1000 Ljubljana, Slovenija
E-pošta: andrej.osterman@telekom.si

Povzetek. Pri uporabi geografskih informacijskih sistemov (GIS) se čedalje več uporablja vzporedno (paralelno) izračunavanje. Privlačna rešitev za vzporedno računanje so grafične enote z arhitekturo CUDA (*Compute Unified Device Architecture*). Osnova za članek je modul $r.los$ za izračun optične vidljivosti (*LOS - Line of Sight*), ki je že implementiran v okolju GRASS GIS (Geographic Resources Analysis Support System). Predstavljen je popolnoma nov modul $r.cuda.los$ z isto funkcionalnostjo kot modul $r.los$. Zaradi morebitne uporabe modula $r.cuda.los$ v radijsko planerske namene je dodana še možnost omejitve računanja po vertikalnem in horizontalnem kotu. Vidnost računamo po posameznih rezinah. Za izračun vsake rezine je določena svoja nit (*thread*) vzporednega procesorja. Pri velikosti zemljevida $28161\,x\,17921$ točk z ločljivostjo $12,5m\,x\,12,5m$, je čas računanja 18s. Pri vzporedni obdelavi podatkov GIS dobimo za en, dva ali celo tri velikostne razrede hitrejšo izvedbo kot pri sekvenčni obdelavi.

Ključne besede: CUDA, vzporedno računanje, GRASS GIS, GPU, vidnost, LOS, kartografija

Implementation of module r.cuda.los into GRASS GIS

Parallel computing is in expanding phase in GIS applications. A very attractive solution for parallel computing are the NVIDIA graphic cards, with a parallel computing platform and the CUDA (*Compute Unified Device Architecture*) programming model. The basis for this paper is the $r.los$ module used to calculate optical visibility (*LOS - Line of Sight*), which is already implemented in the GRASS GIS environment. A completely new $r.cuda.los$ module with the same functionality as the $r.los$ module is presented. By using the $r.cuda.los$ module for radio planning purposes of limiting the computation along the vertical and horizontal angle is also make possible. Visibility is calculated for each slice. The responsibility for the calculation of each slice is with its own thread from the parallel processor. At the size of the map of $28161\,x\,17921$ points with the resolution $12,5m\,x\,12,5m$, the computation time is 18 s. In parallel computing the GIS data, the performance can be one, two or even three size classes faster than in the sequential computing.

1 UVOD

Uporaba geografskih informacijskih sistemov (GIS) je v vzponu. Kot pri drugih področjih se tudi tukaj čedalje bolj uveljavljajo odprtokodni paketi. V članku je za osnovo uporabljen odprtokodni paket GRASS GIS (Geographic Resources Analysis Support System) [1]. GRASS GIS je odprtokodno okolje, ki na enoten način omogoča urejanje geografskih podatkov in njihovo analiziranje, vsebuje pa tudi orodja za izdelavo želenih storitev. Slaba lastnost posameznih orodij v GRASS je njihova relativna počasnost izračunavanja. Počasnost izračunavanja izvira iz dejstva, da se v okoljih GIS uporabljajo relativno obsežne rastrske mape, na katerih je potreben izračun ter sekvenčni način računanja. V sekvenčnem načinu izračunavanja je čas, potreben za izračun neke funkcije, sorazmeren produktu izračuna same funkcije nad točko in števila točk (velikostjo zemljevida).

Boljša možnost je vzporedno (paralelno) izračunavanje. Pri vzporednem računanju lahko čas računanja v dobršni meri skrajšamo na račun delov programa, ki jih lahko prevedemo v vzporedno računanje. Pri vzporednem računanju moramo odpraviti še nekaj drugih težav, na katere pri sekvenčnem računanju ne naletimo, kot na primer sinhronizacija vzporednih opravil. Ena najbolj popularna rešitev v sedanjem času za vzporedno računanje so grafične enote z arhitekturo CUDA (*Compute Unified Device Architecture*). Nova arhitektura omogoča vzporedno procesiranje (lahko tudi negrafičnih matematičnih problemov) na sami grafični kartici ([2]).

V članku se osredinimo na modul $r.los$ za izračun optične vidljivosti (*LOS - Line of Sight*), ki je že implementiran v odprtokodnem okolju GRASS GIS. Sekvenčni modul $r.los$ izračuna polje vidnosti opazovalca. Glavni vhodni podatek je digitalni zemljevid višine terena (*DEM - digital elevation model*). Modulu moramo podati tudi točko interesa oz. opazovanja (koordinato x,y), višino opazovanja nad terenom in maksimalno razdaljo opazovanja oz. računanja (*max_dist*). Sekvenčni modul $r.los$ je sorazmerno počasen za računanje. Zaradi

Prejet 12. december, 2011
Odobren 12. marec, 2012

dolgega računanja postane tako rekoč neuporaben nad maksimalno razdaljo opazovanja več kot 40 km pri digitalnem zemljevidu ločljivosti 100 x 100 m, saj je čas računanja več kot 30 s. V tem članku predstavimo nov modul za računanje vidnosti (*LOS - Line of Sight*). Modul pridobi vstavek $cuda$ in se imenuje $r.cuda.los$. Uporabljeno je vzporedno računanje na grafični kartici NVIDIA. Izkaže se, da je vzporedni modul $r.cuda.los$ sposoben izračunati vidnosti za faktor od 10 do 1000 hitreje od sekvenčnega modula $r.los$. Glavni časovni faktor pri uporabi modula ni več računanje, temveč branje in zapisovanje digitalnega zemljevida iz trdega diska v sistemski spomin gostitelja ter naprej na globalni spomin GPE. Velja tudi, da moramo izračunane podatke zopet prenesti iz globalnega spomina GPE (grafične procesne enote) v sistemski spomin, iz sistemskega spomina pa jih je treba zapisati na trdi disk.

Zaradi morebitne uporabe modula $r.cuda.los$ v radijsko planerske namene (hitra ocenitev, kaj antena 'vidi') so dodani še parametri za azimut, obseg azimuta (horizontalni kot sevanja 'antene'), nagib (*angl. tilt*) in obseg nagiba (vertikalni kot sevanja 'antene'). Modul se uspešno uporablja pri začetnem planiranju radijskega sektorja v mobilni tehnologiji.

Članek je organiziran takole: v poglavju 2 je najprej na kratko opisana priprava digitalnega zemljevida višine terena. Poglavje 3 govori o pripravi globalnega spomina. Poglavje 4 razdela geometrijo posamezne rezine (niti) pri računanju vidnosti. Srce programa, ščepec je opisan v poglavju 5. Tu je podana tudi okrnjena koda ščepca. V poglavju 6 so podani rezultati izračuna in primerjava med sekvenčnim modulom $r.los$ ter modulom $r.cuda.los$. V poglavju 7 podam razpravo, prednosti in slabosti nove implementacije. Opisane so tudi mogoče izboljšave novega modula.

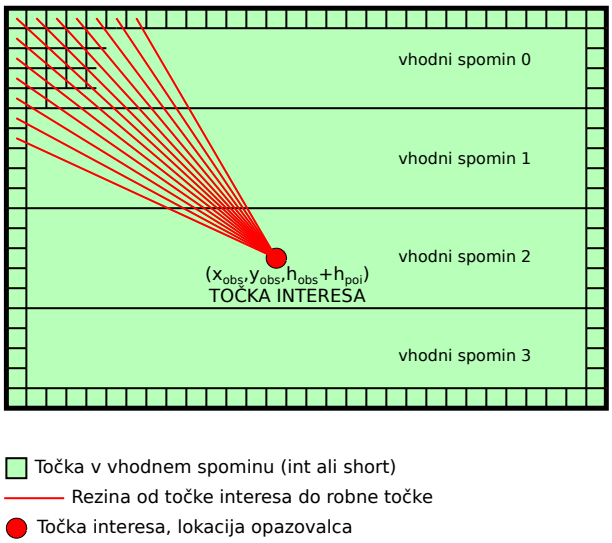
2 Priprava digitalnega zemljevida

Digitalni zemljevid višine terena (*angl. digital elevation map - DEM*) je shranjen na disku v rastrski obliki. Vsebino digitalne mape je treba prebrati v spomin gostitelja ter nato prenesti na napravo GPE. Na napravi GPE se izvede vzporedno računanje nad podatki. Po končanem računanju je treba rezultat prenesti iz naprave GPE v spomin gostitelja ter nato zapisati rezultat na trdi disk. Branje in pisanje na trdi disk je izvedeno s klasičnima C-funkcijama `fread()` in `fwrite()`. Prenos podatkov z gostitelja na napravo GPE in nazaj je izvedeno s funkcijama `cudaMemcpyHostToDevice()` in `cudaMemcpyDeviceToHost()`.

Izkaže se, da večino računalniškega časa (več kot 90 %) porabijo zgoraj omenjene funkcije. Te funkcije so sekvenčne in opravljajo prenašanja podatkov z enega medija na drugega. Vzporedno računanje na podatkih pa se izvede na napravi GPE.

3 Priprava spomina

Vhodni digitalni zemljevid ima dimenzijo $R * C$, pri čemer je R število vrstic, C pa je število stolpcev. Vsaka točka zemljevida vsebuje celoštevilčno vrednost (v dveh ali štirih zlogih), ki pomeni nadmorsko višino. Videz vhodnega spomina je prikazana na sliki 1.



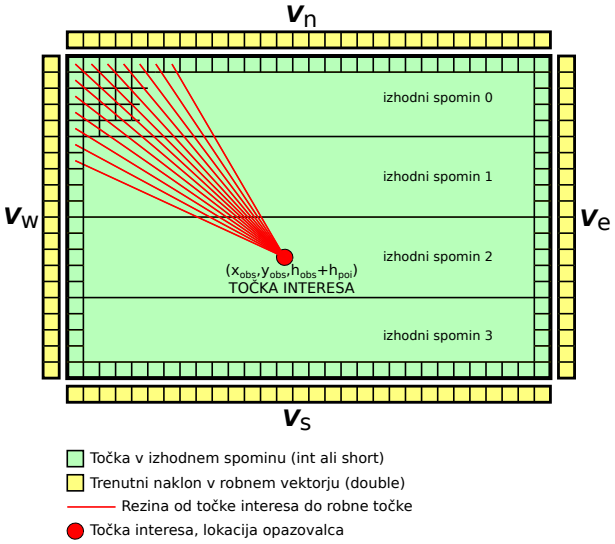
Slika 1: Vhodni spomin

Če je digitalni zemljevid večji od približno polovice razpoložljivega spomina na grafični kartici, moramo digitalni zemljevid razdeliti na več pasov. Računanje na grafični kartici izvedemo najprej s tistim pasom, kjer je točka, iz katere nas zanima izračun vidnosti. Vrstni red računanja je v našem primeru: *vhodni spomin 2; vhodni spomin 1; vhodni spomin 0; vhodni spomin 3*. Omenjeni spomin je na kartici tako imenovani globalni spomin. To pomeni, da lahko do tega spomina katerakoli nit iz kateregakoli bloka enakovredno dostopa. Prav zato ni pomembno, kako so niti organizirane v bloke.

Za izhodni digitalni zemljevid (kjer bo shranjen rezultat) predvidimo enako velikost $R * C$. Pred tem ga moramo zapolniti z vrednostmi *null*, kar se najhitreje lahko naredi z vzporednim računanjem. Poleg tega pa moramo rezervirati še štiri *double* vektorje $\{V_n, V_s, V_w, V_e\}$. V_w, V_e imata R elementov, V_n, V_s imata C elementov. V te štiri vektorje shranjujemo začasne vertikalne kote, enote pa so v radianih. Pred začetkom računanja vse elemente vseh štirih vektorjev napolnimo z vrednostjo $-\pi/2$. Ta vrednost pomeni pogled v tla s strani opazovalca. Pogled naravnost proti horizontu ima vrednost 0, pogled navpično navzgor pa vrednost $\pi/2$.

4 Računanje vidnosti po posameznih rezinah

Vidnost računamo po posameznih rezinah, in sicer vedno začnemo pri točki, ki nas zanima (*Point of Interest*), in se po vnaprej izračunanem koraku premikamo do točke

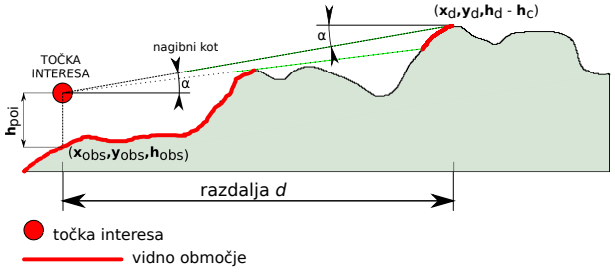


Slika 2: Izhodni spomin in naklonski vektorji

na robu digitalnega zemljevida (glej sliko 2, rdeče črte od POI proti robu zemljevida). Za izračun vsake rezine je določena svoja nit (*thread*) vzporednega procesorja. V posamezni rezini za vsako točko izračunamo razdaljo d od začetne točke (enačba (1)). (x_{obs}, y_{obs}) je koordinata opazovalca, (x_d, y_d) je koordinata trenutne točke, ki jo računamo. d je razdalja med tema dvema točkama.

$$d = \sqrt{(x_{obs} - x_d)^2 + (y_{obs} - y_d)^2} \tag{1}$$

Za vsako točko rezine izračunamo tudi vertikalni kot α (enačba (2)), pod katerim bi bila (je) vidna točka (slika 3).



Slika 3: Rezina

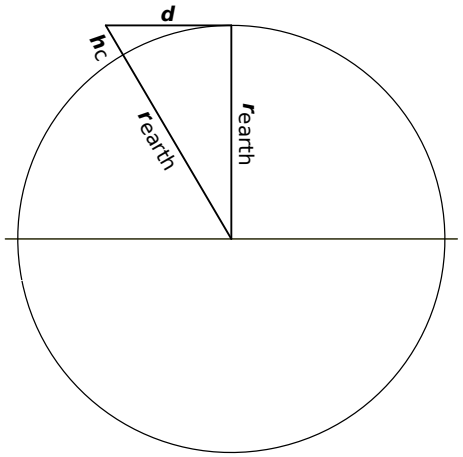
α je kot (*angl. tilt*), pod katerim opazovalec vidi točko (x_d, y_d). h_d je nadmorska višina točke (x_d, y_d). Nadmorska višina, kjer stoji opazovalec, je označena s h_{obs}. Višina opazovalca nad tlemi pa je h_{poi}.

$$\alpha = \arctan\left(\frac{(h_d - h_c) - (h_{obs} + h_{poi})}{d}\right) \tag{2}$$

Enačba (2) ima korekcijski faktor višine h_c. Ta nastopa zaradi ukrivljenosti Zemlje (slika 4), izračunamo ga s pomočjo enačbe (3). V tej enačbi vzamemo za premer Zemlje povprečno vrednost $6370.997\,km$ in ne upoštevamo nadmorske višine opazovalca (če vnesemo povprečno oddaljenost od središče Zemlje za želeno območje, nastane napaka na četrti decimalki).

$$\begin{aligned}(h_c + r_{earth})^2 &= d^2 + r_{earth}^2 \\ h_c &= \sqrt{d^2 + r_{earth}^2} - r_{earth}\end{aligned} \tag{3}$$

V element vektorja $V[tid]$, ki pripada končni točki rezine, vpišemo novo vrednost vertikalnega kota α takrat, ko je nova vrednost večja od že zapisane vrednosti. Hkrati pa dotično točko v izhodnem digitalnem zemljevidu označimo kot vidno (vpišemo vrednost vertikalnega kota, pod katerim jo vidimo). Če pa je vertikalni kot α manjši od vpisane vrednosti v vektorju, ne vpišemo v dotično točko nič (pred tem je že vpisana vrednost *null*). Tako imamo v vidnih točkah vpisane vrednosti kota. V točkah, ki niso vidne, je vpisan *null* element. Če izhodni digitalni zemljevid zapisujemo v celoštevilčnem formatu, izračunu prištejemo vrednost $\pi/2$ in pomnožimo z 10^5.



Slika 4: Popravek višine zaradi ukrivljenosti Zemlje

5 Realizacija programa, ščepec

Poenostavljena izvorna koda ščepca (*angl.kernel*, osnovni gradnik vzporednega programa, ki se izvaja na arhitekturi CUDA [2]) za izračun vidnosti, je izpisana spodaj.

Na začetku vsaka nit dobi svoj indeks (*tid*). Organizacija niti po blokih v tem primeru ni pomembna, ker niti med seboj neposredno ne sodelujejo. Hkrati pa se za shranjevanje digitalnega zemljevida uporablja globalni pomnilnik, do katerega imajo vse niti enakovreden skupen dostop. Slabost globalnega pomnilnika je relativna počasnost dostopa z latenco nekaj 10 urnih period [2]. Vendar tu ni drugih možnosti zaradi velikosti digitalnih zemljevidov.

Vse vhodne podatke, ki se med delovanjem ščepca ne spreminjajo (maksimalna razdalja, omejitev azimuta,

omejitev naklona ...), vpišemo v tako imenovani konstantni spomin (__constant__).

Izkoriščen je indeks *threadIdx.y*, ki pomeni takoimenovane štiri podniti. Namenjen je za nastavljanje in izvrševanje računanja režnjev na severno, južno, zahodno in vzhodno stran digitalnega zemljevida. Na začetku izračunamo korak pomika od točke POI do robne točke zemljevida (x_{step} in y_{step}).

V kodi ni podan izračun predznaka koraka pomika, odvisen pa je od tega, v katero smer od točke POI se pri izračunu gibljemo.

Glavna zanka dejansko pelje spremenljivki x in y od točke POI proti robni točki v smeri x s korakom x_{step} in v smeri y s korakom y_{step}. Znotraj glavne zanke se izračuna razdalja d, kot α in popravek višine h_c za vsako točko v režnju. V originalni kodi se izračuna še azimut (kot proti severu), na podlagi katerega lahko omejimo računanje (če je tako nastavljeno z vhodnimi parametri). Kot α nazadnje primerjamo z vrednostjo vpisano v elementu vektorja $V[tid]$. Če je kot α večji od kota vpisanega v elementu vektorja $V[tid]$, je trenutna točka računanja (x, y) vidna in zato vpišemo kot α v izhodno datoteko.

Število niti v ščepcu je odvisno od velikosti digitalnega zemljevida. Število niti je $N = 2 * C + 2 * R$, pri čemer je C število stolpcev in R število vrstic v vhodnem digitalnem zemljevidu. Kako so niti organizirane v bloke, v tem konkretnem primeru ni pomembno, ker vse niti uporabljajo globalni spomin na GPE.

```
tid=threadIdx.x + blockIdx.x * blockDim.x;
x_step=1.0;
y_step=1.0;

switch( threadIdx.y )
{
case 0:
  if(tid>cols) return;
  x_end=tid;
  y_end=0;
  V=V_n;
  x_step=fabs((x_end - x_obs)/(y_end - y_obs));
  break;
case 1:
  if(tid>cols) return;
  x_end=tid;
  y_end=rows;
  V=V_s;
  x_step=fabs((x_end - x_obs)/(y_end - y_obs));
  break;
case 2:
  if(tid>rows) return;
  x_end=0;
  y_end=tid;
  V=V_w;
  y_step=fabs((y_end - y_obs)/(x_end - x_obs));
  break;
case 3:
  if(tid>rows) return;
  x_end=cols;
  y_end=tid;
  V=V_e;
  y_step=fabs((y_end - y_obs)/(x_end - x_obs));
  break;
}
tilt=-PI/2.0;

// loop from POI to edge point
for(x=x_obs,y=y_obs; (x<=cols)and(y<=rows);
  x+=x_step,y+=y_step)
{
  d=sqrt((x-x_obs)*(x-x_obs)
        +(y-y_obs)*(y-y_obs));
  h_c=sqrt(d*resol*d*resol+6370997.0*6370997.0)
      -6370997.0;
  h=read_input_map(x,y);
  h=h-h_c;
  alpha=atan((h-h_obs)/d);
  if(alpha > (V[tid]-0.00005) )
  {
    write_output_map(x,y,alpha);
    if( alpha > V[tid] ) V[tid]=alpha;
  }
  // POI red color
  if(d<2.5)
  {
    write_output_map(x,y,PI/2);
  }
}
__syncthreads();
```

6 PRIMERJAVA R.LOS IN R.CUDA.LOS

Modul $r.cuda.los$ poženemo iz terminalne vrstice podobno kot modul $r.los$. Oba modula sta testirana na osebnem PC računalniku, na katerem teče Linux. Grafična kartica je NVIDIA GeForce GTX 560Ti, CPU je Intel Core(TM) Duo CPU E8200 @ 2.66GHZ, velikost spomina je 2.0 GiB.

Grafična kartica NVIDIA GeForce GTX 560Ti ima naslednje za nas pomembne lastnosti:

- globalni spomin: 1024 MB
- število jeder: 336
- največje število niti na blok: 1024
- verzija: 2.1

Podrobnejši podatki o omenjeni grafični kartici se nahajajo na spletni strani [8].

Na sliki št. 5 je prikazan primer izračuna z modulom $r.los$ na digitalnem zemljevidu z ločljivostjo $25m\,x\,25m$. Maksimalna razdalja izračuna je 10 km, višina opazovalca pa 20 m. Isti izračun je narejen z novim modulom $r.cuda.los$ (slika št. 6). Obe sliki se po obliki ne razlikujeta. Namerno sta obarvani z različnimi barvami, kar je narejeno z modulom $r.color$.

Na sliki št. 6 tople barve (rumena, oranžna, rdeča) pomenijo pogled opazovalca nad črto navideznega horizonta (negativni naklon, $angl.tilt$), hladne barve (svetlo modra, temno modra) pa pogled opazovalca pod črto navideznega horizonta (pozitivni naklon, $angl.tilt$).

V tabeli 1 je prikazana primerjava časa izvrševanja obeh modulov. Prikazana je kombinacija s tremi različnimi vhodnimi zemljevidi s tremi različnimi ločljivostmi ($100m\,x\,100m$, $25m\,x\,25m$ in $12,5m\,x\,12,5m$). Za vsako ločljivost je vzeta kombinacija štirih razdalj, do katere računamo (parameter max_dist), in sicer 5 km, 10 km, 20 km

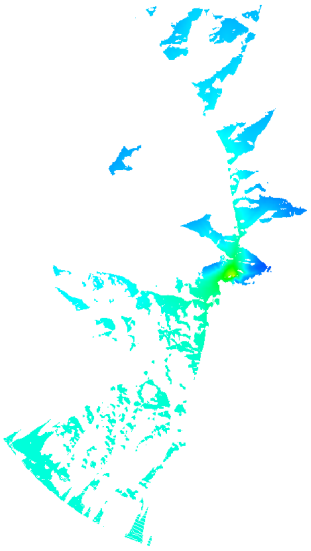
in 50 km. Čas računanja za posamezne kombinacije je podan v sekundah.

Modul $r.los$ za računanje na veliko točkah (daljše razdalje oz. srednje razdalje z bolj podrobno ločljivostjo) ni več primeren, ker je čas računanja praktično neuporaben (na tabeli označeno s pomišljajem).

Z modulom $r.cuda.los$ lahko računamo vidnost čez območje celotne Slovenije na zemljevidu velikosti $28161\,x\,17921$ točk z ločljivostjo $12,5m\,x\,12,5m$, čas računanja je okoli 18 s.

Tabela 1: Primerjava časa računanja modulov r.los in r.cuda.los

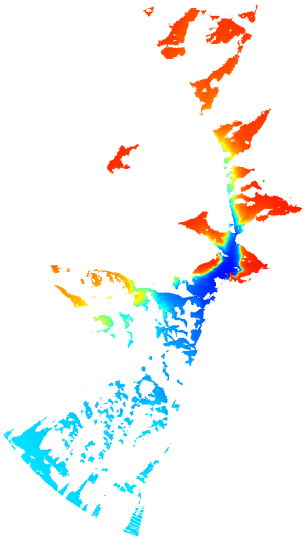
Map [m x m]	max_dist	r.los	r.cuda.los
100m x 100m	5 km	0.1 s	0.05 s
	10 km	0.2 s	0.06 s
	20 km	2.2 s	0.09 s
	50 km	44 s	0.15 s
25m x 25m	5 km	2.4 s	0.3 s
	10 km	30 s	0.3 s
	20 km	511 s	0.6 s
	50 km	-	1.3 s
12.5m x 12.5m	5 km	32 s	0.7 s
	10 km	516 s	1.2 s
	20 km	-	3.1 s
	50 km	-	6.8 s



Slika 5: Izračun vidnosti z modulom r.los

7 Razprava in sklep

Kdorkoli se je že kdaj srečal z okolji GIS, ve, da je priprava geografskih podatkov natančno, vestno, predvsem pa časovno zamudno delo. Vsaka premišljena poteza pri pripravi podatkov se pozneje obrestuje s hitrejšo in lažjo obdelavo podatkov. Vendar pa je zelo pomemben dejavnik tudi samo računanje nad podatki.



Slika 6: Izračun vidnosti z modulom r.cuda.los

Ključno vlogo igra pri tem ustrezna strojna oprema, ki pa mora biti podprta z optimalno programsko opremo.

Obdelava geografskih podatkov kar kliče po vzporednem računanju. Če primerjamo sekvenčno in vzporedno obdelavo podatkov GIS, lahko dobimo pri vzporedni obdelavi za en, dva ali celo tri velikostne razrede hitrejšo izvedbo!

Članek [2] govori na splošno o arhitekturi CUDA. Poleg strokovne vrednosti ima ta članek tudi velik pomen zaradi slovenskega izrazoslovja v tej novi veji tehnike.

V okolju GRASS-GIS je bilo že nekaj raziskav o vzporednem računanju. Ena izmed njih je [3], kjer so vzporedno računanje reševali z grozdom strežnikov, kjer vsak strežnik naredi eno nalogo.

V članku [4] je opisan splošni način programiranja GIS v CUDA s primerom na filtru za povprečenje ($angl.meanfilter$).

V članku [5] je opisana implementacija transformacij projekcij digitalnih zemljevidov s pomočjo CUDA.

Članek [6] sicer ne opisuje vzporednega računanja, vendar se skupina že ukvarja z implementacijo svojih modulov na vzporedni način računanja.

Pričujoči članek ne obravnava splošnega problema migracije iz sekvenčnega na vzporedni način računanja, temveč se omeji na konkretno nalogo: izboljšanje obstoječega modula $r.los$.

Verjetno bo splošen preskok na vzporedni način računanja nastal s časom, ko bo več konkretnih modulov 'migriralo' na vzporedni način računanja.

Za modul $r.los$ je na spletu prosto dostopna izvorna koda. Kljub temu je modul $r.cuda.los$ popolnoma na novo narejen. Oba modula izračunata enak rezultat ter oba uporabljata datotečni način zapisovanja podatkov GRASS. Klic programa je za oba modula precej podoben. Velika večina argumentov je enakih, pri $r.cuda.los$

je nekaj argumentov dodanih za nastavitev dodatne funkcionalnosti.

Novi modul $r.cuda.los$ je pisan za CPU in GPE. Prevajalnik za to je $nvcc$ (namesto gcc za GRASS).

GRASS ima izredno učinkovito narejeno ogrodje za izdelavo modulov. To ogrodje vsebuje tudi GUI za zagon samega modula s pripadajočim ogrodjem za pomoč pri uporabi modula. Za zdaj novi modul $r.cuda.los$ ni združen s tem ogrodjem, zato ga lahko poženemo samo prek komandne vrstice.

Pri novem modulu se omejimo na dva načina zapisa vhodnega in izhodnega zemljevida. Modul lahko prebere nestisnjeni način zapisa v obliki int ali $short$. Rezultat (izhodni zemljevid) dobimo v enaki obliki, kot so vhodni podatki (vhodni zemljevid). Tu se pokaže slabost trenutne implementacije, saj v izhodno datoteko vpisujemo samo celoštevilčne vrednosti. Če imamo majhne vrednosti (kot v našem primeru, kjer zapisujemo kota α z vrednostmi $-\pi/2$ do $\pi/2$), mora biti vpisana vrednost skalirana (pomnožena npr. z 10^5). Zapis v obliki $double$ še čaka na implementacijo.

Če je vhodni zemljevid zapisan v stisnjeni obliki, ga moramo pred tem razširiti z modulom $r.compress$.

Iz tabele 1 vidimo, kakšna prednost nam prinese vzporedno računanje. Tabela prikazuje čas izvajanja modula $r.los$ in modula $r.cuda.los$ glede na vhodne podatke. Modul $r.los$ postane zelo počasen pri velikih vrednostih max_dist. Vidimo, da postane zelo počasen pri vrednostih večjih od 50 km za digitalne zemljevide z ločljivostjo $100m\,x\,100m$, ter pri vrednostih max_dist večjih od 5 km za zemljevide z ločljivostjo $12.5m\,x\,12.5m$.

Modul $r.cuda.los$ ima še kar nekaj možnosti za izboljšave. Predvsem je treba izboljšati časovno potratno branje digitalne mape z diska in nazaj. Ena izmed možnosti je zapis digitalne mape v stisnjeni obliki. Za enoto GPE bi bilo treba v tem primeru napisati ščepec za hitro iskanje vrednosti točk iz stisnjene oblike zapisa.

Vsekakor je vzporedno računanje prihodnost za sisteme GIS. Izvajanje operacij bo pohitril vsaj za velikostni razred.

LITERATURA

[1] *GRASS GIS,* http://grass.fbk.eu

[2] Tomaž Dobravec, Patricio Bulić, *Strojni in programski vidiki arhitekture CUDA*, Elektrotehniški vestnik 77(5): 267–272, 2010

[3] Fang Huang, Dingsheng Liu, Peng Liu, Shaogang Wang, Yi Zeng, Guoqing Li, Wenyang Yu, Jian Wang, Lingjun Zhao, and Lv Pang, Chinese Academy of Sciences, Beijing, China, Northeastern University, Shenyang, China
Research On Cluster-Based Parallel GIS with the Example of Parallelization on GRASS GIS , Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on

[4] Yong Zhao, Zhou Huang, Bin Chen, Yu Fang, Menglong Yan, Zhenzhen Yang, Institute of Remote Sensing and Geographic Information System, Peking University
Local Acceleration in Distributed Geographic Information Processing with CUDA, Geoinformatics, 2010 18th International Conference on.

[5] Yanwei Zhao, Zhenlin Cheng, Hui Dong, Jinyun Fang, Liang Li, Institute of Computing Technology, Chinese Academy of Sciences, Graduate University of Chinese Academy of Sciences
FAST MAP PROJECTION ON CUDA, Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International.

[6] Andrej Hrovat, Igor Ozimek, Andrej Vilhar, Tine Celcer, Iztok Saje, Tomaž Javornik, *An Open-Source Radio Coverage Prediction Tool*, Department of Communication Systems, Jozef Stefan Institute, Mobitel, d.d. ISSN: 1792-4243, ISBN: 978-960-474-200-4

[7] Jasons Sanders, Edward Kandrot, *CUDA by Example*, Addison-Wesley, 2011

[8] *NVIDIA GeForce GTX 560 Ti,* http://uk.geforce.com/hardware/desktop-gpus/geforce-gtx-560ti

Andrej Osterman je diplomiral leta 1991 na Fakulteti za elektrotehniko Univerze v Ljubljani z naslovom diplomske naloge: Grafični prikaz antenskih smernih diagramov v programskem jeziku C++. Zaposlen je v podjetju Telekom Slovenije, d.d., Sektor za radijsko omrežje. Ukvarja se s statistikami na mobilnem omrežju, orodji GIS ter programiranjem v odprtokodnih sistemih.