Paralelny výpočet konvolenčného filtra

Paralelne Programovanie

FIIT STU 2014/15

Martin Černák

Zadanie

Zvoľte si výpočtovo náročnú úlohu a preverte možnosti aplikovania paralelného spracovania na jej riešenie. Využite paralelné programátorské modely OpenMP, MPI a CUDA a dosiahnuté zrýchlenia porovnajte a komentujte. Diskutujte témy s cvičiacimi.

Možné témy:

- Počítačová grafika
  - Generovanie fraktálových obrazcov
  - N-body simulácia
- Strojové učenie
  - Dopredné šírenie v neurónových sieťach
  - Zhluková analýza rozsiahlych dát pomocou algoritmu k-means
- Vlastná téma po dohode s cvičiacimi

Implementácia

Paralelizmus v rámci konvolenčného filtra som sa rozhodol aplikovať pri prechádzaní jednotlivých pixelov zdrojového obrazu.

```
134 void convolution(float *kernel, int kernel_size, float bias) {
135 »···int x, y;
136 »···int id, np;
137 »···int i;
138 »···int kernel_half_size = floor(kernel_size/2);
139 »···const int tag = 42;
140 »···MPI_Status status;
141 »···MPI_Comm_rank(MPI_COMM_WORLD, &id);
142 »···MPI_Comm_size(MPI_COMM_WORLD, &np);
143 
144 »···int size = TEX_SIZE/np;
145 »···int size_pix = size*TEX_SIZE;
146 
147 #ifdef PRAGMA
148 #pragma omp parallel private(x) num_threads(PRAGMA_THREADS) shared(kernel, kernel_size, bias)
149 #pragma omp for
150 #endif
151 »···for(x = id*size; x < size*(id+1); x++) {
152 #ifdef PRAGMA
153 #pragma omp parallel private(y) num_threads(PRAGMA_THREADS) shared(kernel, kernel_size, bias, x)
154 #pragma omp for
155 #endif
156 »···»···for(y = 0; y < TEX_SIZE; y++) {
157 »···»···»···convolution_transform(x, y, kernel, kernel_half_size, kernel_size, bias, size_pix);
158 »···»···}
159 »···}
160 
161 »···if(id == 0) {
162 »···»···for(i=1; i<np; i++) {
163 »···»···»···MPI_Recv(&result[i*size][0], 3*size_pix, MPI_CHAR, i, tag, MPI_COMM_WORLD, &status);
164 »···»···}
165 »···} else {
166 »···»···MPI_Send(&result[id*size][0], 3*size_pix, MPI_CHAR, 0, tag, MPI_COMM_WORLD);
167 »···}
168 }
```

OpenMP

OpenMP som implementoval pomocov paralelneho for - ciklu, ktorý pracuje z obidvoma vnorenými ciklamy na prechádzanie obrázku.

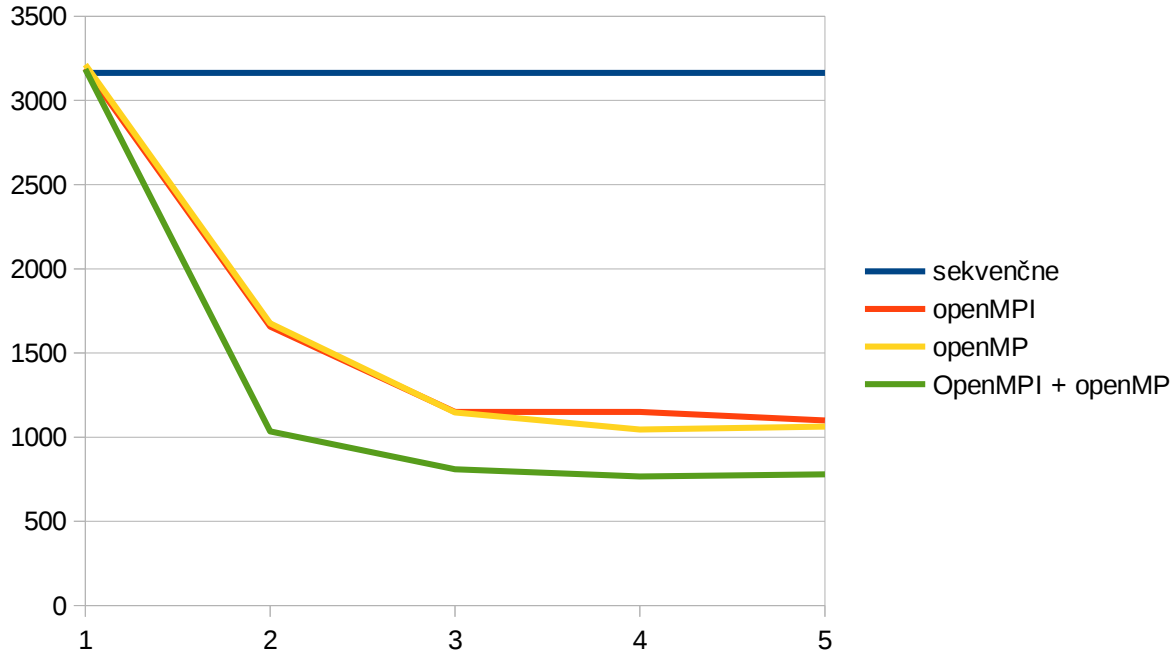
OpenMPI

OpenMPI spúšťa rovnaký program pre každý procesor, programy pracujú úplne rovnako až na dané cikly, kde si na základe id vypočítajú ktorú časť obrázka majú spracovať, následne na konci si master(process 0) zozbiera výsledky práce ostatných procesorov.

Výkonostné testy

Testy sa robili pre kombinácie openMP a openMPI z rôznym počtom procesorov, každé číslo je priemer 3 meraní.

	1	2	3	4	5
sekvenčne	3164,2	3164,2	3164,2	3164,2	3164,2
openMPI	3190,6666666667	1657	1150,6666667	1149,6666667	1100
openMP	3212,6666666667	1676	1146,6666667	1046	1062
OpenMPI + openMP	3186,6666666667	1033,6666667	808,66666667	767	779,66666667



Merania som robil do 5 procesorov pretože pri vyššom počte sa už nedostavilo zlepšienie u žiadneho riešenia.

Na grafe môžeme vydieť že openMP aj openMPI dosiahli omnoho lepšie výsledky ako neparalelne spracovanie (sekvenčné) a tak isto že kombinácie openMP a openMPI dosahuje lepšie výsledky ako každá osobitne.

Záver

OpenMP aj openMPI boli efektívnejšie ako neparalelne riešenie a ich kombinácia dosahovala v priemere najlepšie výsledky.