Grupa 1	Kamil Sacha, Konrad Szwedo	13/12/2016

Zadanie 5 - Rozmycie Gaussa w OpenMPI

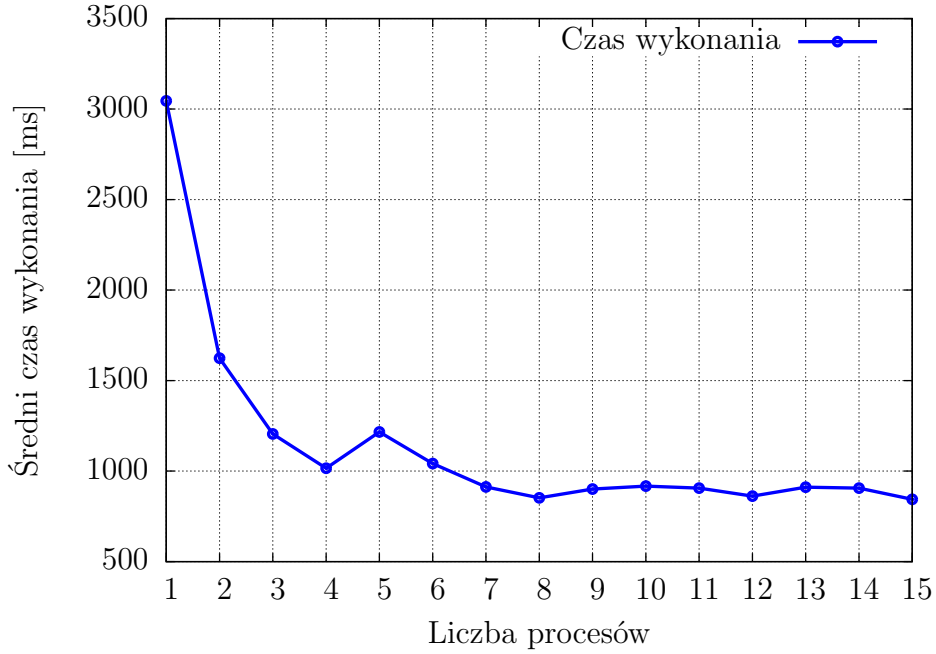
Naszym zadaniem laboratoryjnym było napisanie programu wykonującego rozmycie Gaussa, w oparciu o środowisko do zrównoleglenia kodu: OpenMPI. Algorytm Gaussa polega na zastąpieniu pikseli obrazu wejściowego, średnią z obszaru otaczającego ten piksel. W naszym przypadku rozpatrywaliśmy maskę wielkości 5×5, czyli do obliczenia musieliśmy znać wszystkie 25 wartości trzech kanałów rozpatrywanego piksela (kanały zapisane odpowiednio w kolejności B - Niebieski, G - Zielony, R - Czerwony). Jako wartość poszczególnego kanału (B, G, R) ustawiamy średnią z całości rozpatrywanego obszaru dla konkretnego kanału.

Algorytm obliczeń

Głównym problemem w wykonaniu tego zadania był odpowiedni podział obrazu wejściowego tak aby każdy proces miał odpowiednią porcję danych potrzebną do obliczeń. Proces MASTER wykonujący główne operacje w programie jest odpowiedzialny za sprawdzenie poprawności danych wejściowych, wczytanie obrazu, zapis rozmytego obrazu. Jest również odpowiedzialny za przydzielenie pozostałym procesom zadań i odbiór wyników. Proces główny MASTER, dzieli ilość wierszy na wszystkie obecne procesy (również dla samego siebie) i wysyła je do czekających na dane procesów. Korzystamy z funkcji która zwraca podział obrazu w zależności od całkowitej ilości wierszy, całkowitej ilości procesów oraz numeru procesu dla którego pytamy o wynik. Przykładowo gdy mamy 100 wierszy i uruchamiamy program dla 3 procesów wtedy procesy P_0= 33, P_1=33, P_2=34 otrzymują podaną ilość wierszy. Ponadto zerowy proces musi otrzymać dodatkowo 2 wiersze (wiersze od 33 do 35) ostatni proces także dostaje dwa wiersze poniżej (wiersze od 64 do 100), natomiast procesy środkowe w naszym przykładzie P_1 dostają dodatkowe 4 wiersze (wiersze od 31 do 68). Nadmiarowe wiersze potrzebne są do obliczenia prawidłowej ilości wierszy wynikowych (zawsze 4 mniej) związane jest to oczywiście z zastosowaną przez nas maską 5x5.

```
1  int getRowsCountForProcess(int totalRows, int totalProcesses, int
       currentProcessCount) {
2
3      int rows = 0;
4
5      if (totalProcesses == 1) {
6          return totalRows;
7      }
8
9      rows = totalRows / totalProcesses;
10
11     if (currentProcessCount + 1 == totalProcesses) {
12         rows += totalRows % totalProcesses;
13     }
14
15     return rows;
16 }
```

Listing kodu 1: Funkcja dzieląca procesy.

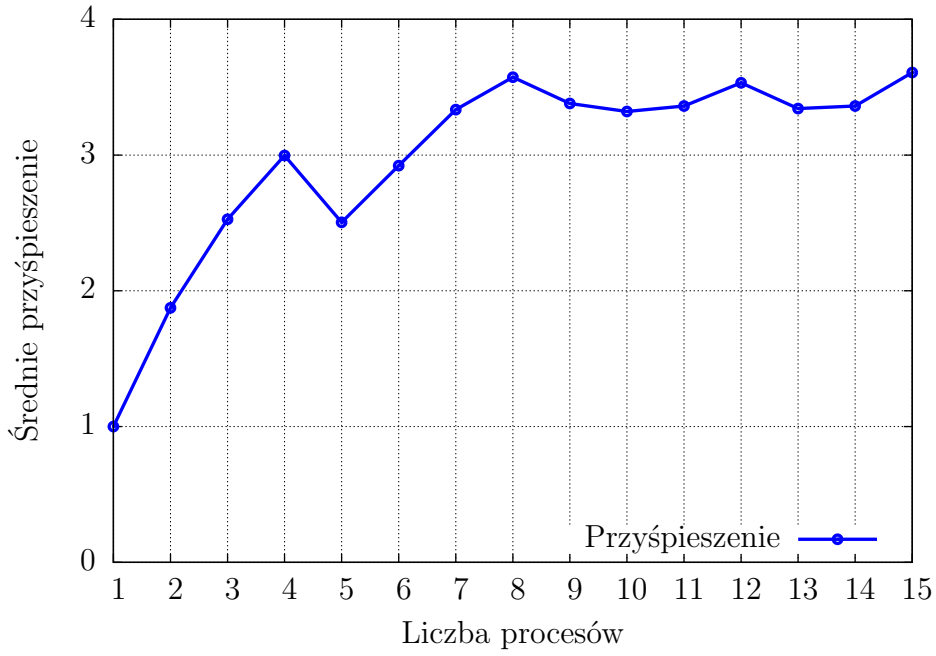


Rysunek 1: Wykres zależności czasu wykonania od ilości procesów. Dla obrazu wejściowego '1.jpg' o wymiarach $24107 \times 4491 \approx 10^9$ pikseli.

Powyższy listing kodu pokazuje początkowy algorytm podziału ilości wierszy dla procesów. Wszystkie procesy otrzymują po równo ilość wierszy, poza ostatnim, który otrzymuje to co zostało w puli. Pierwszy proces dostaje: getRowsCountForProcess + 2 wierszy. Środkowe procesy dostają: getRowsCountForProcess + 4 wierszy z offsetem wierszy na -2 od bieżącego. Ostatni proces dostaje: getRowsCountForProcess + 2 wierszy z offsetem wierszy na -2 od bieżącego.

Wyniki i wnioski

Jak możemy się spodziewać, czas wykonania programu zmniejsza się dla większej niż 1 liczbie procesów. Niestety zaobserwowany wzrost przyśpieszenia nie jest liniowy. Związane jest to przede wszystkim z dużym narzutem na kopiowanie danych. Dla obrazu "1.jpg"posiadającego 3 kanały każdy po 1 bajt wielkość transmisji równa będzie około 309 MiB (minus część nieprzesyłana, którą oblicza proces MASTER). Jest to ilość potrzebna do wysłania w jedną stronę, a dane początkowe musimy jednak najpierw wysłać a potem odebrać wyniki. Wartość czasu wykonania dla 5 procesów większa niż dla 4, najprawdopodobniej jest związana z obecnością 4 rdzeni fizycznych reszta rdzeni jest dublowana w technologii HyperThreading, która to nie odzwierciedla wydajności jak fizyczne rdzenie. Innym problemem wskazującym na dane wyniki, jest problem prawidłowego sprawdzania czasu wykonania. Nie jesteśmy w stanie zmierzyć dokładnego czasu wykonań funkcji realizujących rozmycie Gauss'a, do tego czasu również wliczamy jest okres poświęcony na odbiór wyników z procesów.



Rysunek 2: Wykres zależności przyśpieszenia od ilości procesów. Dla obrazu wejściowego '1.jpg' o wymiarach $24107 \times 4491 \approx 10^9$ pikseli.