Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 12.

Cel zajęć:

Doskonalenie podstaw programowania z przesyłaniem komunikatów MPI.

W ramach zajęć zrealizowałem następujące kroki:

1. Utworzyłem katalog roboczy lab_12 i podkatalog MPI_pi.
2. Wykorzystując szkielet programu oblicz_PI.c napisałem rócœnoległą wersję aplikacji obliczającej wartość liczby PI:

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int max_liczba_wyrazow = 0;
if(rank == 0) {
   printf("Podaj maksymalną liczbę wyrazów do obliczenia przybliżenia
           PI\n");
   scanf("%d", &max_liczba_wyrazow);
}

MPI_Bcast( &max_liczba_wyrazow, 1, MPI_INT, root, MPI_COMM_WORLD );
int wyrazow_na_proces = max_liczba_wyrazow / size;
int a = wyrazow_na_proces * (rank);
int b = wyrazow_na_proces * (rank + 1);

if(rank == size - 1)
   b = max_liczba_wyrazow;
printf("jestem procesem %d i moj przedzial to: %d -> %d\n", rank, a, b);

SCALAR suma_plus = 0.0;
SCALAR suma_minus = 0.0;
int i = 0;
for(i=a ; i<b ; i++) {
   int j = 1 + 4 * i;

   suma_plus += 1.0 / j;
   suma_minus += 1.0 / (j + 2.0);
}
SCALAR suma = suma_plus - suma_minus;
SCALAR suma_global = 0;
MPI_Reduce( &suma, &suma_global, 1, MPI_DOUBLE, MPI_SUM, root,
            MPI_COMM_WORLD);
```

```
if(rank == 0) {
   int i = 0;
      //TUTAJ KOD SEKWENCYJNY DLA PORÓWNANIA WYNIKU
   suma_plus = 0;
   suma_minus = 0;
   for (i = 0; i < max_liczba_wyrazow; i++) {

       int j = 1 + 4 * i;
       suma_plus += 1.0 / j;
       suma_minus += 1.0 / (j + 2.0);
   }
   printf("PI obliczone: \t\t\t%20.15lf\n", 4 * (suma_plus - suma_minus));
   printf("PI rownolegle: \t%20.15lf\n", 4 * (suma_global));
   printf("PI z biblioteki matematycznej: \t%20.15lf\n", M_PI);
}

MPI_Finalize();
```

Po zainicjalizowaniu zmiennych rank oraz size pobieramy liczbę wyrazów, którą później proces root wysyła do wszystkich procesów. Na jej podstawie obliczamy zakres jaki przypada na każdy z procesów. Ostatni z procesów otrzymuje nadwyżkę w przypadku gdy liczba wyrazów nie jest podzielna przez liczbę procesów. Następnie realizuję pętlę analogiczną do tej w programie sekwencyjnym z jedną różnicą w zakresie. Po policzeniu suma_plus i suma_minus sumuję je tak, aby wynikowej zmiennej użyć w funkcji MPI_Reduce, która zbierze wyniki procesów i zsumuje je w procesie root.

Ostatni element kodu, to sekwencyjne policzenie tego samego ciągu i porównanie wyników poprzez ich wypisanie na ekran.

3. Utworzyłęm podkatalog MPI_mat_vec, w której wypakowałem pobraną ze strony przedmiotu paczkę.
4. Uruchomiłem oraz sprawdziłem poprawność kodu. Następnym etapem było przeanalizowanie kodu i wyróżnienie różnych jego fragmentów.

```
[ligeza_tomasz@ESTERA MPI_mat_vec]$ make all
rm -f *.o ./mat_vec_row_MPI
/usr/lib64/openmpi/bin/mpicc -c -O2 -fopenmp mat_vec_row_MPI.c
/usr/lib64/openmpi/bin/mpicc -O2 -fopenmp mat_vec_row_MPI.o -o mat_vec_row_MPI -lm
/usr/lib64/openmpi/bin/mpiexec -np 4  ./mat_vec_row_MPI
poczatek (wykonanie sekwencyjne)
        czas wykonania (zaburzony przez MPI?): 0.087347, Gflop/s: 2.326507, GB/s> 9.306953
Starting MPI matrix-vector product with block row decomposition!
Wersja rownolegla MPI z dekompozycją wierszową blokową
        czas wykonania: 0.033099, Gflop/s: 6.139604, GB/s> 24.560852
Starting MPI matrix-vector product with block column decomposition!
Werja rownolegla MPI z dekompozycją kolumnową blokową
        czas wykonania: 0.033279, Gflop/s: 6.106304, GB/s> 24.427638
Blad! i=0, y[i]=170698750320.000000, z[i]=0.000000 - complete the code for column decomposition
```

5. Następnie zmodyfikowałem kod tak, żeby zamiast MPI_Send i MPI_Recv używać procedur komunikacji grupowej.

Najpierw wykomentowałem odpowiednio procedury rozprowadzania i odbierania danych przy pomocy MPI_Send oraz MPI_Recv (w kodzie rozpoczynające się komentarzem "`point-to-point not optimal communication`"). Po czym zastąpiłem je poniższymi wywołaniami funkcji MPI_Scatter oraz MPI_Gather.

Dane początkowe rozsyłam za pomocą MPI_Scatter.

```
MPI_Scatter(a, n_wier*WYMIAR, MPI_DOUBLE, a_local, n_wier*WYMIAR,
            MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Scatter(x, n_wier, MPI_DOUBLE, &x[rank*n_wier], n_wier, MPI_DOUBLE, 0,
            MPI_COMM_WORLD);
```

Po czym odbieram je przy pomocy MPI_Gather:

```
MPI_Gather(z, n_wier, MPI_DOUBLE, z, n_wier, MPI_DOUBLE, 0,
           MPI_COMM_WORLD);
```

6. Po modyfikacji uruchomiłem program. Jak widać dekompozycja wierszowo blokowa działa bez zarzutu, jako, że nie ukazują się żadne komunikaty błędów.

```
[ligeza_tomasz@ESTERA MPI_mat_vec]$ make all
rm -f *.o ./mat_vec_row_MPI
/usr/lib64/openmpi/bin/mpicc -c -O2 -fopenmp mat_vec_row_MPI.c
/usr/lib64/openmpi/bin/mpicc -O2 -fopenmp mat_vec_row_MPI.o -o mat_vec_row_MPI -lm
/usr/lib64/openmpi/bin/mpiexec -np 4  ./mat_vec_row_MPI
poczatek (wykonanie sekwencyjne)
        czas wykonania (zaburzony przez MPI?): 0.069318, Gflop/s: 2.931612, GB/s> 11.727613
Starting MPI matrix-vector product with block row decomposition!
Wersja rownolegla MPI z dekompozycją wierszową blokową
        czas wykonania: 0.032589, Gflop/s: 6.235701, GB/s> 24.945278
Starting MPI matrix-vector product with block column decomposition!
Werja rownolegla MPI z dekompozycją kolumnową blokową
        czas wykonania: 0.032412, Gflop/s: 6.269769, GB/s> 25.081566
Blad! i=0, y[i]=170698750320.000000, z[i]=0.000000 - complete the code for column decomposition
```

Wnioski:

Używanie procedur komunikacji grupowej znacznie upraszcza pisany przez nas kod. Pozwala na zmniejszenie ilości linijek kodu kilku- albo nawet kilkunastokrotnie, co oczywiście jest zaletą, bo jak każdy wie im więcej linijek kodu tym więcej okazji do popełnienia błędu.

Poniżej przedstawiam jeszcze porównanie czasu wykonania wersji z użyciem procedur komunikacji grupowej, oraz bez jej użycia. Wyniki nie mają znacznych różnic.

```
poczatek (wykonanie sekwencyjne)
        czas wykonania (zaburzony przez MPI?): 1.308181, Gflop/s: 2.485439, GB/s> 9.942003
Starting MPI matrix-vector product with block row decomposition!
Wersja rownolegla MPI z dekompozycją wierszową blokową
        czas wykonania: 0.506234, Gflop/s: 6.422730, GB/s> 25.691559
```

```
poczatek (wykonanie sekwencyjne)
        czas wykonania (zaburzony przez MPI?): 1.264623, Gflop/s: 2.571046, GB/s> 10.284439
Starting MPI matrix-vector product with block row decomposition!
Wersja rownolegla MPI z dekompozycją wierszową blokową
        czas wykonania: 0.497228, Gflop/s: 6.539062, GB/s> 26.156895
```