Antoni Pawlak

401480

Laboratorium 11: MPI - komunikaty

Cel ćwiczenia

- Opanowanie podstaw programowania z przesyłaniem komunikatów MPI.
- Poznanie składni MPI
- Kompilacja programu MPI

Wykonanie

Przygotowania

1. Utworzenie katalogu i pobranie plików ze strony
2. Instalacja MPI

   # paczka instalująca mpicc

   sudo apt-get install libopenmpi-dev

3. Modyfikacja Makefile

 # kompilator c

CCOMP = mpicc

 # konsolidator

LOADER = mpicc

 # program uruchomieniowy MPI

MPI_run = mpirun -oversubscribe

4. Przesłanie hostnamame

- W programie przesyłającym

 # pobranie nazwy hosta

MPI_Get_processor_name(processor_name, &res_len);

 # wysłanie długości nazwy

MPI_Send(res_len, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);

 # wysłanie nazwy

MPI_Send(processor_name, res_len, MPI_CHAR, dest, tag2, MPI_COMM_WORLD);

- W programie odbierającym

 # odebranie długości nazwy

MPI_Recv(res_len_sent, 1, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, &status);

 # odebranie nazwy

MPI_Recv(processor_name_sent, res_len_sent, MPI_CHAR, MPI_ANY_SOURCE, 2, MPI_COMM_WORLD, &status);

5. Sztafeta

```
# pętla po ilości rund
for(int i = 0; i<ROUNDS; i++) {

    # proces rozpoczynający sztafetę
  if (rank == 0) {
    # wysłanie danych do kolejnego, rozpoczęcie pierścienia
    MPI_Send( &to_send, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    printf("Proces: %d || Wyslal dane: %d || Do procesu: %d\n\n",rank, to_send, 1);
    # odebranie danych od ostatniego
    MPI_Recv(&received, 1, MPI_INT, size-1, 0, MPI_COMM_WORLD, &status);
    printf("Proces: %d || Otrzymane dane: %d || Od procesu: %d\n",rank, received,
status.MPI_SOURCE);
    # inkrementacja danych
    to_send = received+1;
    # weryfikacja czy to koniec sztafety, zakończenie pierścienia
    if(i == ROUNDS-1) printf("\nMETA: Koniec sztafety! || Konczy proces: %d || WARTOSC:
%d\n", status.MPI_SOURCE, received);
  }

    # pozostałe procesy
  else {
    # odebranie danych od poprzednika
    MPI_Recv(&received, 1, MPI_INT, rank-1, 0, MPI_COMM_WORLD, &status);
    # inrementacja danych
    to_send = received+1;
    printf("Proces: %d || Otrzymane dane: %d || Od procesu: %d\n",rank, received,
status.MPI_SOURCE);
    # sytaucja w której proces jest ostatnim, kończącym sztafetę
    if(rank == size-1) {
      MPI_Send(&to_send,1,MPI_INT,0,0,MPI_COMM_WORLD);
      printf("Proces: %d || Wyslal dane: %d || Do procesu: %d\n\n",rank, to_send, 0);
    }
```

 # sytuacja w której proces NIE jest kończącym sztafetę

```
    else {
      MPI_Send(&to_send,1,MPI_INT,rank+1,0,MPI_COMM_WORLD);
      printf("Proces: %d || Wyslal dane: %d || Do procesu: %d\n\n",rank, to_send, rank+1);
    }
  }
}
```

6. Struktura
   - **Ustalenie wielkości bufora**
     - **zliczenie rozmiaru poszczególnych pól**

     MPI_Pack_size(1,MPI_DOUBLE,MPI_COMM_WORLD,&rozmiar);

     rozm_pakietu = rozmiar;

     MPI_Pack_size(1,MPI_INT,MPI_COMM_WORLD,&rozmiar);

     rozm_pakietu += rozmiar;

     MPI_Pack_size(NAME_LENGTH,MPI_CHAR,MPI_COMM_WORLD,&rozmiar);

     rozm_pakietu += rozmiar;

     - **zarezerwowanie pamięci dla bufora**

     void* bufor = (void*)malloc(rozm_pakietu);

     void* in_bufor = (void*)malloc(rozm_pakietu);

   - **zapakowanie struktury**
     - **wyzerowanie pozycji**

     pozycja = 0;

     - **zapakowanie kolejnych pól**

     MPI_Pack(&to_send.name,NAME_LENGTH,MPI_CHAR,bufor,rozm_pakietu,&pozycja, MPI_COMM_WORLD);

     MPI_Pack(&to_send.step,1,MPI_INT,bufor,rozm_pakietu,&pozycja, MPI_COMM_WORLD);

     MPI_Pack(&to_send.time,1,MPI_DOUBLE,bufor,rozm_pakietu,&pozycja, MPI_COMM_WORLD);

   - **Wysłanie zapakowanej struktury**
     - **wysłanie jako MPI_PACKED**

     MPI_Send(bufor, rozm_pakietu, MPI_PACKED,1,0,MPI_COMM_WORLD);

- **Odebranie zapakowanej struktruy**
  - **odebranie jako MPI_PACKED**

MPI_Recv(in_bufor,rozm_pakietu,MPI_PACKED,size-1,0,MPI_COMM_WORLD,&status);

- **Odpakowanie struktury**
  - **wyzerowanie pozycji**

pozycja =0;

- **odpakowanie kolejnych pól**

MPI_Unpack(in_bufor,rozm_pakietu,&pozycja,&received.name, NAME_LENGTH ,MPI_CHAR, MPI_COMM_WORLD);

MPI_Unpack(in_bufor,rozm_pakietu,&pozycja,&received.step, 1 ,MPI_INT, MPI_COMM_WORLD);

MPI_Unpack(in_bufor,rozm_pakietu,&pozycja,&received.time, 1 ,MPI_DOUBLE, MPI_COMM_WORLD);

Wnioski

Przesyłanie kilku komunikatów

- Przesyłając pomiedzy procesami kilka komunikatów stajemy przed problemem synchronizacji
- MPI rozwiązuje ten problem poprzez zastosowanie tagów
- tag dodany przy wysyłaniu wiadomości sprawia, że możliwa jest odbioru jedynie wiadomości z określonego źródła I tagu a tym samym niwelacja konfliktu synchronizacji
- Wniosek: chcąc mieć wpływ na procesowanie kilku wiadomości należy je w jakiś sposób rozróżniać, system tagów jest jednym z rozwiązań

Komunikaty w konwencji pierścienia

- Tworząc strukturę pierścienia pomiędzy procesami, deklarujemy krąg po którym będzie krążyć wiadomość
- Kluczowymi w implementacji są tutaj pierwszy I ostatni węzeł
- Pierwszy musi rozpocząć I bardzo często również kończyć
- Ostatni musi wiedzieć że jest ostatni I wysłać z powrotem na pierwszy
- Wysłanie komunikatów w kolejności jest możliwe dzięki blokującej charakterytyce polecenia MPI_Recv()
- Wniosek: Implementacja pierścienia opiera się na zaplanowaniu chronologii komunikacji, oraz użycia odpowiednich blokujących odbiorników.

Pakowanie struktur

- Implemntacja przesyłu własnego typu danych leży po stronie programisty
- Nie jest możliwa trywialna forma polecenia, ze względy na specyfikę środowiska MPI
- Naszą paczkę danych należy spakowac I rozpakować:
  - Zapisujemy rozmiar paczki
  - Pakujemy
  - Wysyłamy
  - Rozpakowujemy
- Wniosek: Przesył elementu opiera się na podzieleniu go na prymitywne pola znanych dla MPI typów.

Przebieg programu recv czeka

- Polecenie MPI_Recv() wstrzymuje wykonanie aż do odebrania wiadomości
- Pozwala to na
  - obsługę synchornizacji
  - utworzenie struktury pierścienia
  - sterowanie przebiegiem programu

Wniosek

- MPI posiada wbiudowany mechanizm komunikacji pomiedzy procesami
- Znajomość odpowiednich funkcji znacząco zaoszczędza czasu programiście
- Programy MPI są przenośne, niezależne od środowiska
- Poprawne stosowanie tego środowiska jest możliwe jedynie dzięki wcześniejszemu pozananiu paradygmatu Programowania Równoległego I typowych jego problemów.