Przegląd funkcji MPI

Wojciech Wieczorek

Optymalizacja z użyciem klastrów komputerowych

Plan

- Zasady programowania z przesyłaniem wiadomości
- Podstawowe operacje: Send i Receive
- Topologie
- Nakładanie komunikacji i obliczeń
- Komunikacja kolektywna
- Grupy i komunikatory

Podstawowe zasady programowania zgodnego z MPI

- System komputerowy, który jest w stanie wykonywać programy MPI składa się z p procesów, a każdy z nich ma własną (niewspółdzieloną) przestrzeń adresową.
- Dane wejściowe muszą być rozdzielone przez programistę (nie ma żadnego automatycznego rozdzielania danych pomiędzy procesy).
- Komunikacja wymaga kooperacji między dwoma procesami – procesem, który ma dane i procesem, który chce je mieć.

Uwagi dodatkowe

- Procesy MPI działają asynchronicznie lub częściowo synchronicznie.
- W modelu częściowej synchronizacji procesy lub ich podzbiór uzgadniają interakcje. Pomiędzy tymi okresami interakcji procesy działają całkowicie asynchronicznie.
- W większości programów MPI mamy do czynienia z tzw. modelem single program multiple data (SPMD).

Podstawowy podział komunikacji

Blokująca:

Łatwa w implementacji, ale może dochodzić do marnotrawienia czasu procesora (do bezczynnego czekania).

Nieblokująca:

Trudniejsza w implementacji, ale dopuszcza się w niej współbieżne wykonywanie: przesyłania wiadomości oraz innych operacji.

Cztery tryby przesyłania blokującego

- Wszystkie cztery tryby przesyłania (Send) działają z funkcją MPI_Recv.
- Różnica sposobu przesyłania wiadomości w tych trybach dotyczy takich elementów jak: występowanie lub niewystępowanie buforowania oraz kolejność rozpoczęcia/zakończenia operacji przesyłania/odbierania.
- We wszystkich trybach parametry są takie same; różnica jest oczywiście w działaniu i nazwach funkcji:
 MPI Ssend, MPI Bsend, MPI Rsend, MPI Send

MPI_Ssend (synchronicznie)

- Bardzo bezpieczna metoda, bo:
- Nieistotna jest kolejność wywołania funkcji wysyłającej i odbierającej oraz:
- Komunikacja nie jest uzależniona od żadnego bufora.
- Musimy jednak ponieść oczywiste ryzyko synchronizacji, czyli potencjalne przestoje.
- (Standard MPI dopuszcza, aby standardowy MPI_Send był zaimplementowany za pomocą MPI_Ssend.)

MPI_Bsend (z buforowaniem)

- Przesyłanie może się zakończyć, nawet zanim odbieranie się rozpocznie.
- Utworzenie bufora odpowiedniej wielkości spoczywa na programiście. Robi się to za pomocą funkcji MPI_Buffer_attach. Taki bufer może być tylko jeden na proces. Do jego dezaktywacji służy funkcja MPI_Buffer_detach.
- Może wystąpić błąd przepełnienia bufora, ale:
- Eliminujemy przestoje związane z synchronizacją.

MPI_Rsend (bez bufora, bez synchronizacji)

- Odbieranie musi być zainicjowane przed rozpoczęciem przesyłania.
- Brak przestojów po stronie procesu przesyłającego kosztem możliwych przestojów po stronie procesu odbierającego.

MPI_Send (standardowe przesyłanie)

- Może występować bufor (systemowy) po stronie procesu przesyłającego, odbierającego lub obydwu.
- Może być synchronizowane, ale zazwyczaj jest buforowane.
- W przypadku wyczerpania bufora następuje synchronizacja.

Nieblokujące przesyłanie i odbiór

Schemat postępowania jest następujący:

```
MPI_Isend lub MPI_Irecv
... inne operacje (ale bez odwołań do bufora p/o) ...
MPI Wait MPI Waitany MPI Test MPI Testany
```

- Wait(any) jest blokujący
- Test (any) jest nieblokujący

Alternatywny nieblokujący odbiór

Schemat postępowania jest następujący:

```
MPI_Send
... inne operacje (ale bez odwołań do bufora p/o) ...
MPI_Iprobe
gdy flaga ustawiona można wykonać MPI_Recv
```

 Różnica między tym schematem, a poprzednim polega na tym, że Iprobe nie jest skojarzone z żadną operacją wysyłania danych (można odebrać dowolną wiadomość, a nie tylko taką, która ma taki sam parametr request).

Przykład: produkcja - konsumpcja

```
typedef struct {
  char data[MAXSIZE];
  int datasize;
} Buffer;
std::vector<Buffer> buffer;
std::vector<MPI Request> req;
MPI Status status;
int rank; // nr procesu
int p; // liczba procesów
```

Przykład: produkcja - konsumpcja

```
if(rank != p-1) { /* kod procesu produkującego */
  buffer.reserve(1);
 while(true) { /* zasadnicza pętla */
    produce(buffer[0].data, buffer[0].datasize);
   MPI_Send(buffer[0].data, buffer[0].datasize, MPI_CHAR, p-1, 0, MPI_COMM_WORLD);
else { /* rank == p-1; kod procesu konsumującego */
  buffer.reserve(p-1);
  req.reserve(p-1);
  int i;
  for(i=0; i < p-1; i++)
   MPI Irecv(buffer[i].data, MAXSIZE, MPI CHAR, i, 0, MPI COMM WORLD, &req[i]);
 while(true) { /* zasadnicza petla */
   MPI_Waitany(p-1, &req[0], &i, &status);
   MPI_Get_count(&status, MPI_CHAR, &buffer[i].datasize);
    consume(i);
   MPI Irecv(buffer[i].data, MAXSIZE, MPI CHAR, i, 0, MPI COMM WORLD, &req[i]);
```

MPI: ciekawostka

Minimalny zbiór funkcji MPI.

```
MPI_Init
MPI_Finalize
MPI_Comm_size
MPI_Comm_rank
MPI_Send
```

MPI Recv

Komunikatory

- Służą do definiowania grupy komunikujących się ze sobą procesów.
- Informacje związane z komunikatorami odczytujemy za pomocą funkcji typu MPI Comm.
- Komunikatory używane są jako argumenty we wszystkich funkcjach związanych z przesyłaniem wiadomości.
- Proces może należeć do wielu grup (komunikatorów).
- Zdefiniowano domyślny komunikator o nazwie
 MPI_COMM_WORLD, który zawiera wszystkie procesy.

Uzyskiwanie podstawowych informacji

- Funkcje MPI_Comm_size oraz MPI_Comm_rank służą do określenie liczby procesów oraz numeru procesu wywołującego.
- Parametry wywołania są następujące:

```
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

 Zmienna rank czyli numer procesu przyjmie wartość od 0 do size-1, czyli do liczby procesów w danej grupie.

Typy MPI

MPI MPI_CHAR MPI_SHORT MPI_INT MPI_LONG MPI_UNSIGNED_CHAR unsigned char MPI UNSIGNED SHORT unsigned short int MPI UNSIGNED MPI UNSIGNED LONG MPI FLOAT MPI DOUBLE

MPI_BYTE

MPI PACKED

C/C++ signed char signed short int signed int signed long int unsigned int unsigned long int float double MPI_LONG_DOUBLE long double

Adresowanie w komunikacji

- W MPI stosuje się maski zarówno w określaniu źródła jaki i taga.
- Jeśli źródło przyjmuje postać MPI_ANY_SOURCE, to dowolny proces w grupie może pełnić rolę nadawcy.
- Jeśli tag przyjmuje postać MPI_ANY_TAG, to akceptowane są wiadomości etykietowane dowolnym tagiem.
- Po stronie odbiorcy należy zadeklarować odbiór danych dostatecznie dużego rozmiaru (faktycznie przesłana wiadomość może być krótsza lub równa temu co zadeklarowano).

Unikaj zakleszczeń!

Rozważmy fragment programu:

```
int a[10], b[10], myrank;
MPI Status status;
MPI Comm rank(MPI COMM WORLD, &myrank);
if (myrank == 0) {
    MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
    MPI Send(b, 10, MPI INT, 1, 2, MPI_COMM_WORLD);
else if (myrank == 1) {
    MPI Recv(b, 10, MPI INT, 0, 2, MPI COMM WORLD);
    MPI Recv(a, 10, MPI INT, 0, 1, MPI COMM WORLD);
```

Jeśli MPI_Send jest blokujące, to mamy problem.

Inny przykład zakleszczenia

Rozważmy fragment programu, w którym proces i wysyła wiadomość do procesu i + 1 (modulo p), a odbiera od procesu i - 1 (modulo p).

```
int a[10], b[10], p, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Send(a, 10, MPI_INT, (myrank+1)%p, 1, MPI_COMM_WORLD);
MPI_Recv(b, 10, MPI_INT, (myrank-1+p)%p, 1, MPI_COMM_WORLD);
...
```

Raz jeszcze mamy problem, jeśli MPI_Send jest blokujące.

Możliwe rozwiązanie

Jednoczesne wysyłanie i odbiór

Aby wymienić się wiadomościami, można zastosować funkcję:

```
int MPI_Sendrecv(void *sendbuf, int sendcount,
    MPI_Datatype senddatatype, int dest, int
    sendtag, void *recvbuf, int recvcount,
    MPI_Datatype recvdatatype, int source, int recvtag,
    MPI_Comm comm, MPI_Status *status)
```

Możliwe jest skorzystanie tylko z jednego bufora, z którego dane są odczytywane i do którego dane będą zapisywane:

```
int MPI_Sendrecv_replace(void *buf, int count,
    MPI_Datatype datatype, int dest, int sendtag,
    int source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

Topologie

- W MPI jednostki obliczeniowe można organizować w k-wymiarowe siatki.
- Numery procesów w grupie MPI_COMM_W0RLD można odwzorowywać w inne komunikatory (odpowiadające wielowymiarowej siatce) na kilka sposobów.
- O skuteczności i użyteczności przyjętej topologii decyduje charakter problemu, ale także budowa systemu komputerowego.
- Programista nie ma żadnej kontroli nad szczegółami tego odwzorowania (np. (1,1) = proces_nr 1, czy nr 7).

Tworzenie siatki w układzie kartezjańskim

Służy do tego funkcja:

Każdy proces w starej grupie comm_old otrzymuje miejsce w utworzonej siatce (grupie comm_cart).

 Do procesu można się od tego momentu odwoływać za pomocą wektora współrzędnych.

Używanie siatek

 Standardowe wysyłanie i odbiór nadal wymagają (jednowymiarowego) numeru procesu, dlatego też w MPI mamy funkcję konwerującą nr do współrzędnych kartezjańskich i funkcję odwrotną:

 Jedną z najczęściej wykonywanych operacji na siatkach jest MPI_Sendrecv, gdzie parametry źródła i celu łatwo uzyskać za pomocą funkcji:

Komunikacja kolektywna

- W MPI zdefiniowano duży zbiór funkcji realizujących komunikację kolektywną.
- W tego rodzaju komunikacji biorą udział wszystkie procesy z grupy.
- Wszystkie procesy z grupy muszą daną funkcję wywoływać.

Przykłady operacji kolektywnych

 Blokowanie do momentu kiedy wszystkie procesy osiągną wyznaczony punkt:

```
int MPI_Barrier(MPI_Comm comm)
```

Rozgłaszanie typu jeden-do-wszystkich:

Redukcja typu wszyscy-do-jednego:

Predefiniowane operatory redukcji

Operacja	Znaczenie	Typ danych
MPI_MAX	Maksimum	Całkowite i zmiennopozycyjne
MPI_MIN	Minimum	Całkowite i zmiennopozycyjne
MPI_SUM	Suma	Całkowite i zmiennopozycyjne
MPI_PROD	lloczyn	Całkowite i zmiennopozycyjne
MPI_LAND	Logiczne AND	Całkowite
MPI_BAND	Bitowe AND	Całkowite
MPI_LOR	Logiczne OR	Całkowite
MPI_BOR	Bitowe OR	Całkowite
MPI_LXOR	Logiczne XOR	Całkowite
MPI_BXOR	Bitowe XOR	Całkowite
MPI_MAXLOC	Lokalizacja max	Pary danych
MPI_MINLOC	Lokalizacja min	Pary danych

Dodatkowe typy danych

Zdefiniowane w celu łatwiejszego odczytywania danych z operacji redukcji za pomocą MPI_MAXLOC oraz MPI_MINLOC.

. , , , , , , , , , , , , , , , , , , ,	Typ	danych MPI	
---	-----	------------	--

MPI_2INT

MPI_SHORT_INT

MPI_LONG_INT

MPI_LONG_DOUBLE_INT

MPI_FLOAT_INT

MPI_DOUBLE_INT

Typy danych w C/C++

para int-ów

short i int

long i int

long double i int

float i int

double i int

Inne operacje komunikacji kolektywnej

 Aby wynik redukcji został przekazany wszystkim procesom, należy zastosować funkcję:

Aby wyznaczyć sumy skumulowane stosujemy:

Inne operacje komunikacji kolektywnej

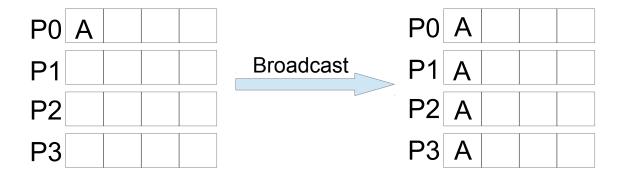
Gromadzenie danych:

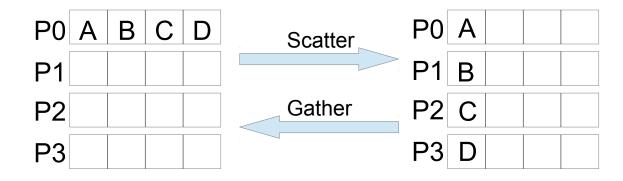
Rozdzielanie danych:

Inne operacje komunikacji kolektywnej

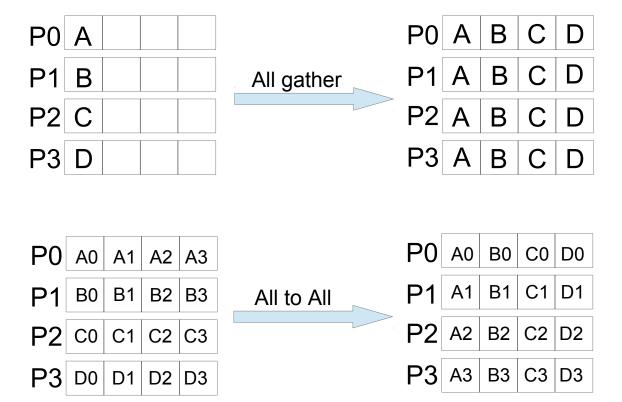
Wszyscy-do-wszystkich:

Ilustracja graficzna wybranych operacji





Ilustracja graficzna wybranych operacji

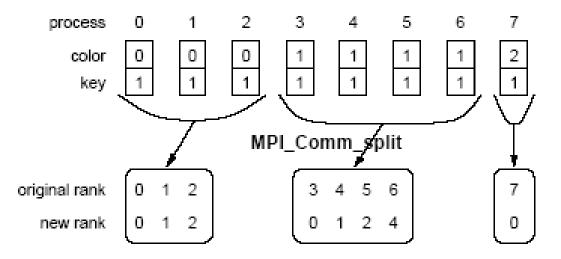


Podział na grupy

- W niektórych algorytmach równoległych wymaga się, aby komunikacja przebiegała w ramach podzbioru procesów.
- W MPI można dokonać takiego podziału, uzyskując zbiór nowych komunikatorów.
- Najprostszą funkcją do tego służącą jest:

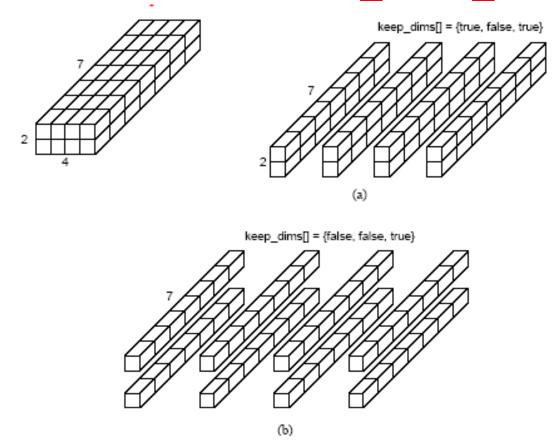
 Podział na grupy następuje na podstawie kolorów, a w ramach grupy procesy są numerowane na podstawie klucza.

Grupy i komunikatory



Przed i po wywołaniu funkcji MPI_Comm_split.

Podział siatki: MPI_Cart_sub



Podział siatki wymiaru 2 x 4 x 7 na (a) cztery grupy wymiaru 2 x 1 x 7 lub (b) osiem grup wymiaru 1 x 1 x 7.