Programowanie w modelu przesyłania komunikatów – specyfikacja MPI

Model przesyłania komunikatów

- Paradygmat “send-receive”
  - wysyłanie komunikatu:
    send(cel, identyfikator_komunikatu, dane)
  - odbieranie komunikatu:
    receive(źródło, identyfikator_komunikatu, dane)
- Uniwersalność modelu
- Wysoka efektywność i skalowalność obliczeń
- Programy MPMD lub SPMD
- Konkretne środowiska programowania z przesyłaniem komunikatów zawierają narzędzia uruchamiania programów (często także kompilowania i debugowania)

Środowisko przesyłania komunikatów MPI

- Interfejs programowania definiujący powiązania z językami C, C++, Fortran
- Standaryzacja (de facto) i rozszerzenie wcześniejszych rozwiązań dla programowania z przesyłaniem komunikatów (PVM, P4, Chameleon, Express, Linda) w celu uzyskania:
  - przenośności programów równoległych
  - kompletności interfejsu
  - wysokiej wydajności obliczeń
  - łatwości programowania równoległego
- Opracowany w latach: 1992-95 MPI-1 i 1995-97 MPI-2

Środowisko przesyłania komunikatów MPI

- Podstawowe pojęcia:
  - komunikator (predefiniowany komunikator MPI_COMM_WORLD)
  - ranga procesu
- Podstawowe procedury:
  - int MPI_Init(int *pargc, char ***pargv)
  - int MPI_Comm_size(MPI_Comm comm, int *psize)
  - int MPI_Comm_rank(MPI_Comm comm, int *prank)

    (0 <= *prank < *psize)
  - int MPI_Finalize(void)

Środowisko przesyłania komunikatów MPI

```
#include “mpi.h”
int main( int argc, char** argv ){
    int rank, size, source, dest, tag, i;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf(“Mój numer %d, w grupie %d procesów\n”, rank, size);
    MPI_Finalize();
    return(0);
}
```

Środowisko przesyłania komunikatów MPI

- Procedury dwupunktowego (*point-to-point*) przesyłania komunikatów:
  - MPI gwarantuje postęp przy realizacji przesłania (po prawidłowym zainicjowaniu pary send-receive co najmniej jedna z nich zostaje ukończona)
  - MPI gwarantuje zachowanie porządku przyjmowania (w kolejności wysyłania) dla komunikatów z tego samego źródła, o tym samym identyfikatorze i w ramach tego samego komunikatora
  - MPI nie gwarantuje uczciwości przy odbieraniu komunikatów z różnych źródeł
  - W trakcie realizacji procedur przesyłania może wystąpić błąd związany z przekroczeniem limitów dostępnych zasobów systemowych

Środowisko przesyłania komunikatów MPI

- Procedury dwupunktowego przesyłania komunikatów:
  - Przesyłanie blokujące – procedura nie przekazuje sterowania dalej dopóki operacje komunikacji nie zostaną ukończone i nie będzie można bezpiecznie korzystać z buforów (zmiennych) będących argumentami operacji
  - int MPI_Send(void* buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm)
  - int MPI_Recv(void *buf, int count, MPI_Datatype dtype, int src, int tag, MPI_Comm comm, MPI_Status *stat)
  - Wykorzystanie zmiennej *stat do uzyskania parametrów odebranego komunikatu (m.in. MPI_Get_count)

Środowisko przesyłania komunikatów MPI

```
#include “mpi.h”
int main( int argc, char** argv ){
int rank, ranksent, size, source, dest, tag, i; MPI_Status status;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if( rank != 0 ){ dest=0; tag=0;
    MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
} else {
    for( i=1; i<size; i++ ) { MPI_Recv( &ranksent, 1, MPI_INT, MPI_ANY_SOURCE,
                              MPI_ANY_TAG, MPI_COMM_WORLD, &status );
    printf(“Dane od procesu o randze: %d (%d)\n”, ranksent, status.MPI_SOURCE );}
}
MPI_Finalize(); return(0); }
```

MPI – typ danych spakowanych

- int MPI_Pack(void* buf_dane, int count, MPI_Datatype typ, void* buf_send, int buf_send_size, int* pozycja, MPI_Comm comm) - pakowanie count zmiennych o typie typ i o początku w pamięci buf_dane do bufora buf_send o rozmiarze buf_send_size; pozycja jest pozycją końca danych w buforze wysyłania, i jednocześnie rozmiarem spakowanej paczki
- int MPI_Unpack(void* buf_recv, int buf_recv_size, int* pozycja, void* buf_dane, int count, MPI_Datatype typ, MPI_Comm comm) - rozpakowanie paczki
- typ MPI_PACKED stosuje się tak jak predefiniowane typy elementarne i nowo tworzone typy, z tym że liczba zmiennych jest teraz rozmiarem paczki w bajtach (rozmiar można uzyskać używając MPI_Pack_size)

Typ MPI_PACKED - przykład

```
struct rekord{   double skalar;  char znak;  float wektor[3];   };
struct rekord   baza[20000000];
int rozm, rozm_pakietu, pozycja; void* bufor; // lub char bufor[10000000000];
MPI_Pack_size(1, MPI_DOUBLE, MPI_COMM_WORLD, &rozmiar);
rozm_pakietu = rozm;
MPI_Pack_size(1, MPI_CHAR, MPI_COMM_WORLD, &rozmiar);
rozm_pakietu += rozm;
MPI_Pack_size(3, MPI_FLOAT, MPI_COMM_WORLD, &rozmiar);
rozm_pakietu += rozm;
bufor = (void *)malloc(3*rozm_pakietu);  pozycja = 0;
for( i=0; i<3; i++ ) {
      MPI_Pack(&baza[i].skalar,1,MPI_DOUBLE,bufor,3*rozm_pakietu,&pozycja,MCW);
      MPI_Pack(&baza[i].znak, 1, MPI_CHAR, bufor, 3*rozm_pakietu, &pozycja, MCW);
     MPI_Pack(&baza[i].wektor[0],3,MPI_FLOAT,bufor,3*rozm_pakietu,&pozycja,MCW);
}
MPI_Send( bufor, pozycja, MPI_PACKED, 1, 0, MPI_COMM_WORLD );
```