Технология разработки параллельных программ для МВС с распределенной памятью (MPI)

- Особенность - независимая работа CPU
- Для распараллеливания необходимо:
  - Распределение вычислительной нагрузки.
  - Информационное взаимодействие (передача данных) между CPU.
- Библиотека функций MPI решает эти задачи.
- MPI – Message Passing Interface, *интерфейс передачи сообщений.*

Модель программирования – SPMD
(single program multiple data)

- Одна программа запускается для **разных** данных на нескольких (>1) CPU.
- В основе MPI – явный **двусторонний** обмен сообщениями:
  - Согласование действий на обеих сторонах (передатчик-приемщик) для любой передачи данных.
  - Автоматическая принудительная синхронизация с ожиданием опоздавшего (т.наз. *рандеву*).
- Основное внимание при использовании MPI– обеспечение взаимодействия.

Понятие MPI

- **Стандарт**, которому должны удовлетворять средства организации передачи сообщений (MPI 1.0 был принят в 1994 г.).
- **Программные средства**, обеспечивающие передачу сообщений и соответствующие стандарту MPI:
  - Организованы в библиотеку.
  - Доступны для ЯП C (C++) и Fortran.

Преимущества MPI

- Реализованы **основные** коммуникационные операции.
- Обеспечены **разные** способы пересылки данных.
- Упрощена **переносимость** параллельных программ между разными компьютерными системами (и платформами!).
- В библиотеках MPI учтены **особенности** МВС на уровне оборудования.
- Упрощена **разработка** параллельных программ:
  - Основные операции передачи данных предусмотрены стандартом MPI.
  - Есть библиотеки параллельных методов, созданных на базе MPI.

Понятие процесса

- **Процесс** – ключевое понятие для технологии MPI.
- **Процесс** – отдельная программа с ее данными на процессоре:
  - Исполняемый модуль.
  - Адресное пространство.
  - Доступ к информационным ресурсам (файлам, портам).

Состояния процесса

- **Пассивное** – процесс известен системе, но в конкуренции за ресурсы не участвует.
  При этом ему предоставляется оперативная и/или внешняя память.
- **Активное -** во время своего существования процесс может участвовать в конкуренции за использование ресурсов ОС:
  - **Выполнение** (*running*) – все затребованные процессом ресурсы выделены. В МВС количество активных процессов <=числа процессоров.
  - **Готовность к выполнению** (*ready*) – если предоставить ресурсы, то процесс перейдет в состояние выполнения.
  - **Блокирование или ожидание** (*blocked)* – выполнение процесса может быть продолжено только после наступления некоторого ожидаемого им события.

Процессы и потоки (треды).

- В любой момент времени **выполняющимся** процессом (т.е. использующим процессор) может быть **только один процесс**.
- Процесс рассматривается ОС как заявка на потребление всех видов ресурсов, **кроме процессорного времени.**
- Процессорное время распределяется **на уровне тредов.**
- У каждого процесса есть по меньшей мере один тред.
- Если число CPU>1, то треды могут выполняться одновременно.
- Тред – **легковесный процесс**.

Основные понятия MPI

- Параллельная программа (ПП) – **множество одновременно выполняемых процессов**.
- Каждый процесс ПП порождается на основе копии **одного и того же** программного кода.
- Процессы могут выполняться на разных CPU.
- На одном CPU могут располагаться несколько процессов (*выполнение в режиме разделения времени*).
- Количество процессов и число используемых CPU определяется (как правило) **статически** – в момент запуска ПП средствами среды исполнения MPI-программ.
- Общих переменных нет
  взаимодействие – через **сообщения**.
- Все процессы программы перенумерованы с 0.
- Номер процесса называют **рангом процесса**.

Основные «параметры» MPI

- Тип операции передачи сообщения
- Коммуникаторы
- Тип данных, пересылаемых в сообщении
- Виртуальная топология

Операции передачи сообщений

- Операции, поддерживаемые MPI функциями:
  - **Парные** (*point-to-point) – для взаимодействия между двумя процессами.*
  - **Коллективные** *(collective) – для одновременного взаимодействия нескольких процессов.*

Коммуникаторы

- **Коммуникатор** –специально создаваемый **служебный объект, объединяющий группу процессов** и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи сообщений.
- **Парные** операции выполняются для процессов, принадлежащих **одному и тому же коммуникатору.**
- **Коллективные** операции применяются одновременно **для всех процессов коммуникатора.**
- Для операций передачи сообщений **обязательно указывается используемый коммуникатор**.

Коммуникаторы

- В ходе вычислений можно **создавать новые и удалять существующие коммуникаторы**.
- **Один** и тот же процесс может принадлежать **разным** коммуникаторам.
- **Все** процессы ПП входят в состав создаваемого по умолчанию коммуникатора с идентификатором **MPI_COMM_WORLD**.
- Для передачи данных между процессами из разных групп необходимо создавать **глобальный коммуникатор** (*intercommunicator*).

Типы данных

- При выполнении операций передачи сообщений для указания передаваемых или получаемых данных **в функциях MPI необходимо указывать тип пересылаемых данных**.
- MPI содержит набор **базовых типов данных**,

  во многом совпадающих с типами данных C и Fortran.
- В MPI имеются возможности для создания **новых производных типов данных**

  для уточнения описания содержимого пересылаемых сообщений.

Виртуальная топология

- **Логическая топология** линий связи между процессами имеет структуру **полного графа** (**независимо от наличия реальных физических каналов** связи между процессорами).
- В MPI имеются средства для формирования логических (виртуальных) топологий **любого** требуемого типа.

Основы разработки MPI-программ

- **Функция инициализации**
- **Первой** вызываемой функцией MPI должна быть **функция инициализации среды выполнения** кода MPI программы*:*

```
int MPI_Init(int* count, char** text)
```

- ***Параметры*** *функции:*
  - *count - количество аргументов в командной строке,*
  - *text – массив символов (текст) командной строки.*

Пример

```
void main (int argc, char *argv[])
{
  …
  MPI_Init(&argc, &argv);//адреса!
//далее использование MPI
  …
  MPI_Finalize();
// код без использования MPI
```

Ссылки – тип C++

- Ссылка – другое имя объекта данных.
- Ссылка - указатель, который:
  - жестко привязан к области памяти
  - (на которую он указывает),
  - автоматически разыменовывается, при обращении по имени ссылки
- Ссылка **обязательно** инициализируется при объявлении

Правила для ссылок

- **Ссылка д.б. инициализирована (не null)**

```
int& n; //ошибка
double& x=1; // правильно
……
double* x;
double temp;
temp=(double) 1;
x=&temp;// адрес
```

- **Операции не действуют на ссылки**

```
int n=0;
int &k=n;
k++;//n=1
```

```
int a; //переменная с именем "a"
  типа int размещена по адресу A
int &ra = a; //задано альтернативное
  имя (ra) для переменной по адресу
  A
cout << &a <<'\n'<<&ra << '\n';
// &a означает взятие адреса
  переменной
```

Основы разработки MPI-программ

- **Функция завершения**
- **Последней** вызываемой функцией MPI должна быть функция завершения кода MPI*:*

```
void MPI_Finalize(void)
```

- Обе функции **обязательны** и выполняются **один раз каждым процессом.**

Основы разработки MPI-программ

- **Функция определения количества процессов**

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

`comm` – коммуникатор, для которого определяется размер,

`size` – определяемое количество процессов.

- **Функция определения ранга процесса**

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

`comm` – коммуникатор, в котором определяется ранг,

`rank` – определяемый ранг процесса в коммуникаторе.

Пример

```
int main( int argc, char *argv[] )
{ int ProcNum, ProcRank;
//код без MPI функций
…
  MPI_Init( &agrc, &argv);
  MPI_Comm_size( MPI_COMM_WORLD, &ProcNum);
//нашли общее число процессов
  MPI_Comm_rank( MPI_COMM_WORLD, &ProcRank);
//нашли ранг текущего процесса
//код с использованием MPI функций
…
  MPI_Finalize();
//код без использования MPI
…
  return 0;
}
```

Комментарии

- Коммуникатор MPI_COMM_WORLD создается по умолчанию и представляет **все процессы** выполняемой параллельной программы.
- Ранг, получаемый при помощи функции MPI_Comm_rank, является **рангом процесса, выполнившего вызов** этой функции.
- Поэтому переменная ProcRank будет принимать **различные** значения в разных процессах.

Пример

```
static long num_steps = 100000;
void main (int argc, char *argv[])
{
  int i, my_id, numprocs;
  double x, pi, step, sum = 0.0 ;
  step = 1.0/(double) num_steps ;
  MPI_Init(&argc, &argv) ;
  MPI_Comm_Rank(MPI_COMM_WORLD, &my_id) ;
  MPI_Comm_Size(MPI_COMM_WORLD, &numprocs) ;
  my_steps = num_steps/numprocs ;
  for (i=my_id; i<num_steps; i+=numprocs)
  {
   x = (i+0.5)*step;
   sum += 4.0/(1.0+x*x);
  }
  sum *= step ;
  MPI_Reduce(&sum,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD)
  ;MPI_Finalize();
}
```