Средства и системы параллельного программирования

4 курс кафедры СКИ
сентябрь – декабрь 2016 г.

Лектор доцент Н.Н.Попова

Лекция 11/2

28 ноября 2016 г.

Тема

- Односторонние операции MPI-2

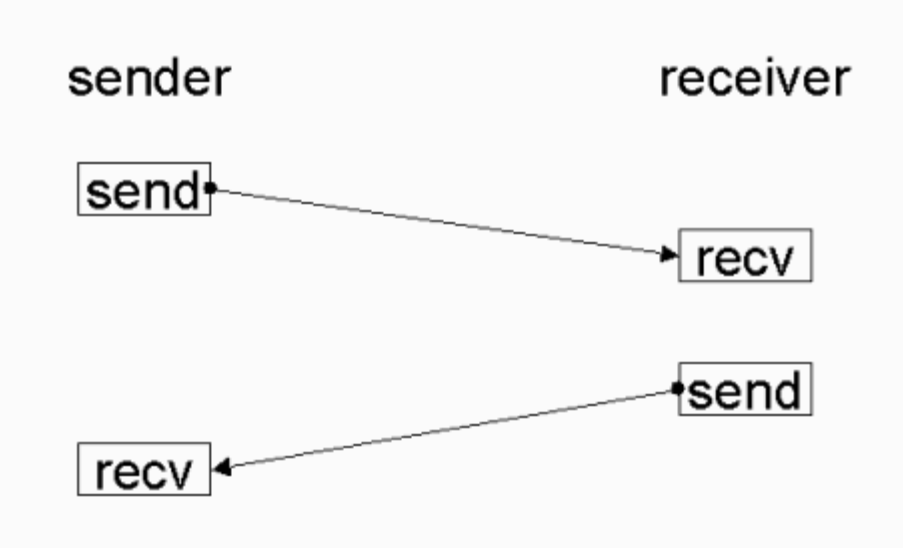
Общие концепции

- 2 основных способа организации обмена сообщениями между взаимодействующими процессами:
  - message passing
  - RMA – удаленный доступ к памяти
- В передаче данных необходимо участие лишь одного процесса
- RMA-механизм позволяет разработчикам воспользоваться преимуществом быстрых механизмов связи, обеспечиваемых различными платформами
- Примеры реализации: shmem (Cray), LAPI (IBM)
- Отличается от концепции механизмов «Общая память»

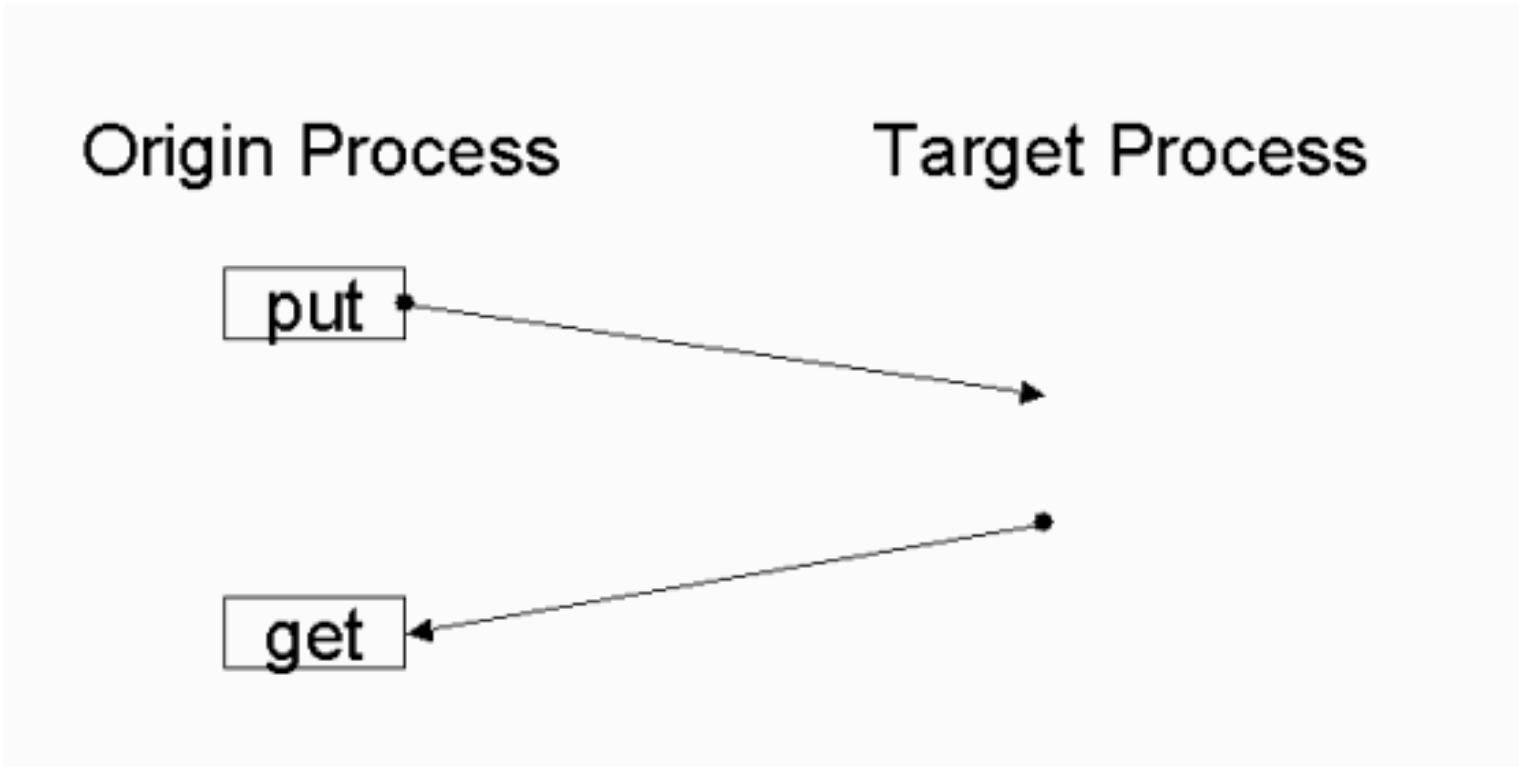
Терминология

- Инициатор (origin)– процесс, который выполняет вызов RMA-функции
- Адресат (target) – процесс, к памяти которого выполняется обращение
- 3 коммуникационных вызова
  - MPI_Put()
  - MPI_Get()
  - MPI_Accumulate()

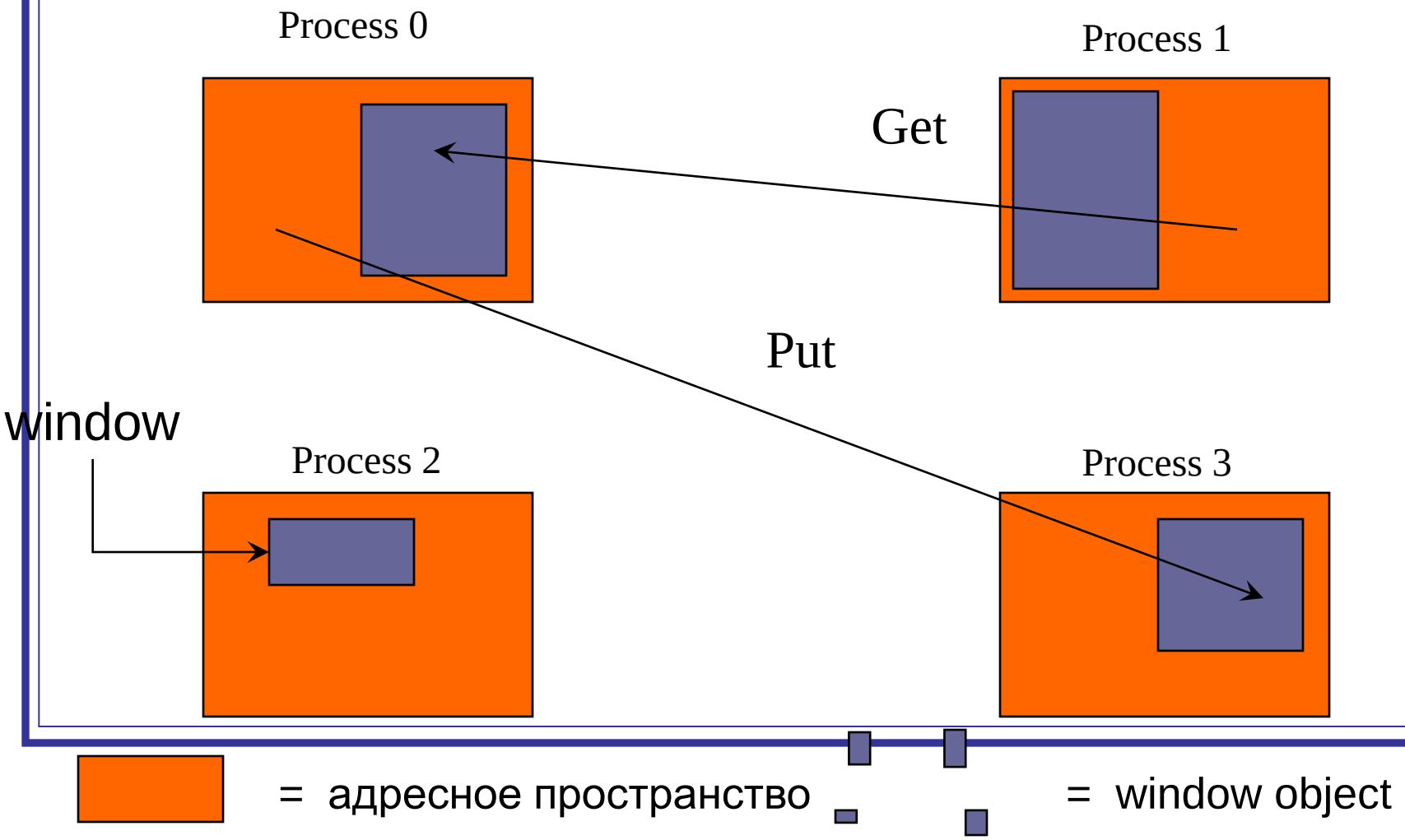
Взаимодействие: двухсторонние передачи



Взаимодействие: односторонние (RMA)



RMA (Remote Memory Access) Windows и Window Objects



Порядок работы с RMA-операциями

- Создание «окна» – определение участка памяти, который будет использован а RMA-операции
- Определение данных, участвующих в передаче
- Определение способа оповещения о готовности данных

Односторонние функции

- Initialization
  - MPI_ALLOC_MEM, MPI_FREE_MEM
  - MPI_WIN_CREATE, MPI_WIN_FREE
- Remote Memory Access (RMA, nonblocking)
  - MPI_PUT
  - MPI_GET
  - MPI_ACCUMULATE
- Synchronization
  - MPI_WIN_FENCE (like a barrier)
  - MPI_WIN_POST / MPI_WIN_START / MPI_WIN_COMPLETE / MPI_WIN_WAIT
  - MPI_WIN_LOCK / MPI_WIN_UNLOCK

Создание «окна»

- «Окно» –участок памяти, который может использоваться в RMA-операциях
- Этот участок должен быть непрерывным (contiguous) блоком
- Для создания «окна» достаточно указать начальный адрес и количество байт
- Создание «окна» – коллективная операция, должна вызываться всеми процессами внутри коммуникатора
- «Окно» - объект со скрытой структурой, который используется для всех дальнейших RMA-операций

Пример
асинхронные пересылки – двухсторонние передачи

```
/* Создание коммуникаторов */
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_split (MPI_COMM_WORLD, rank<=1, rank, &comm);

/* Только процессы 0 и 1 выполняют последующие действия */
if (rank > 1) return;

/* Процесс 0 посылает, а процесс 1 получает данные */
if (rank == 0)
     MPI_Isend (outbuf, n,MPI_INT, 1,0,comm, &request);
else if (rank ==1) {
     MPI_Irecv (inbuf, n,MPI_INT, 0,0,comm, &request);
}
/* Вычисления, другие передачи */
…
/* Завершение передачи */
MPI_Wait(&request, &status);
MPI_Comm_free(&comm);
```

Пример
асинхронные пересылки – односторонние передачи

```
/* Создание коммуникаторов */
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_split (MPI_COMM_WORLD, rank<=1, rank, &comm);
/* Только процессы 0 и 1 выполняют последующие действия */
if (rank > 1) return;
/* Процесс 0 посылает, а процесс 1 получает данные */
if (rank == 0)
   MPI_Win_create (MPI_BOTTOM, 0, sizeof(int),
   MPI_INFO_NULL,comm, &win);}
else if (rank ==1) {
     MPI_Win_create (inbuf, n* sizeof(int),
     sizeof(int),MPI_INFO_NULL,comm, &win);}
     }
```

Пример
асинхронные пересылки – односторонние передачи

```
if (rank >1) return;
/* Process 0 puts data into proc 0 */
MPI_Win_fence (0,win);
if (rank ==0)
    MPI_Put (outbuf, n, MPI_INT, 1,0,n,MPI_INT,win);
…….

/* Завершение передачи */
MPI_Win_fence (0,win);
/* Free window  */
MPI_Comm_free(&win);
```

Создание «окна»

int MPI_Win_create(void *base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, MPI_Win *win)

base - начальный адрес окна

size - размер окна в байтах (неотрицательное целое число)

disp_unit - размер локальной единицы смещения в байтах (положительное целое)

info - аргумент (дескриптор)

comm - коммуникатор (дескриптор)

win - оконный объект, вызвращаемый вызовом (дескриптор)

Атрибуты «окна»

- MPI_WIN_BASE - базовый адрес «окна»

  MPI_Win_get_attr(win, MPI_WIN_BASE, &base, &flag)
- MPI_WIN_SIZE - размер «окна», в байтах

  MPI_Win_get_attr(win, MPI_WIN_SIZE, &size, &flag)
- MPI_WIN_DISP_UNIT - единица смещения, связанная с «окном»

  MPI_Win_get_attr(win,MPI_WIN_DISP_UNIT, &disp_unit,&flag)
- Группа процессов, присоединенных к «окну»

  int MPI_Win_get_group(MPI_Win win, MPI_Group *group)

Пример MPI_WIN_CREATE

```
int main(int argc, char ** argv)
{
    int *a;    MPI_Win win;
    MPI_Init(&argc, &argv);
    /* create private memory */
    a = (void *) malloc(1000 * sizeof(int));
    /* use private memory like you normally would */
    a[0] = 1;  a[1] = 2;

    /* collectively declare memory as remotely accessible */
    MPI_Win_create(a, 1000, sizeof(int), MPI_INFO_NULL,
                   MPI_COMM_WORLD, &win);

    /* Array 'a' is now accessibly by all processes in
     * MPI_COMM_WORLD */
    MPI_Win_free(&win);

    MPI_Finalize(); return 0;
}
```

Перемещение данных

- Перемещение данных – MPI_Put, MPI_Get, MPI_Accumulate
- Все операции по перемещению данных – неблокирующие
- Синхронизация обязательна (чтобы убедиться, что операция завершена)

Перемещение данных: *Get*

MPI_Get(

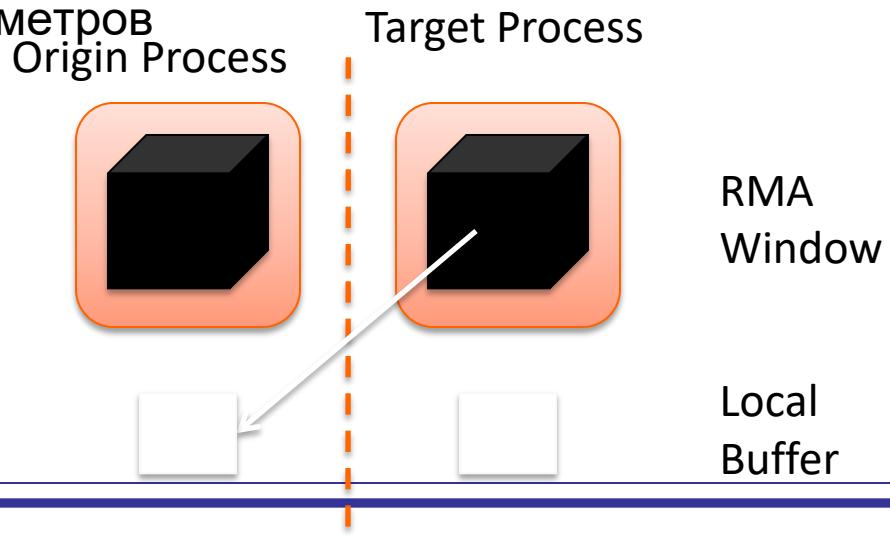
origin_addr, origin_count, origin_datatype,

target_rank,

target_disp, target_count, target_datatype,

win)

- Пересылка данных **в** origin **из** target
- Отдельное описание тройки параметров для origin и target



```
int MPI_Get(void *origin_addr, int origin_count,
    MPI_Datatype origin_datatype, int target_rank,
    MPI_Aint target_disp, int target_count,
    MPI_Datatype target_datatype, MPI_Win win)
```

origin_addr начальный адрес буфера инициатора

origin_count число записей в буфере инициатора (неотрицательное целое)

origin_datatype тип данных каждой записи в буфере инициатора

target_rank ранк получателя (неотрицательное целое)

target_disp смещение от начала окна до буфера адресата (неотрицательное целое)

target_count число записей в буфере адресата (неотрицательное целое)

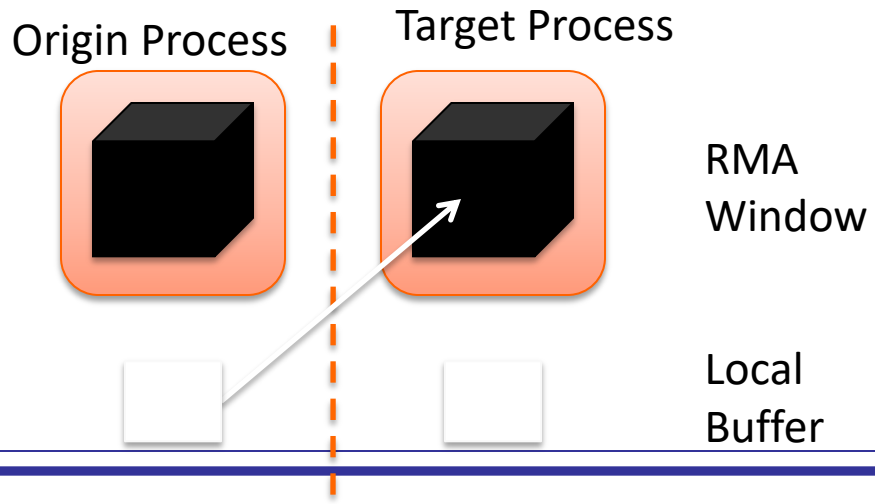
target_datatype тип данных каждой записи в буфере адресата

win оконный объект, используемый для коммуникации

Перемещение данных: *Put*

MPI_Put(
origin_addr, origin_count, origin_datatype,
target_rank,
target_disp, target_count, target_datatype,
win)

- Пересылка данных **<u>из</u>** origin **<u>в</u>** target
- Такие же аргументы как и у MPI_Get



```
int MPI_Put(void *origin_addr, int origin_count,
    MPI_Datatype origin_datatype, int target_rank,
    MPI_Aint target_disp, int target_count, MPI_Datatype
    target_datatype, MPI_Win win)
```

origin_addr начальный адрес буфера инициатора

origin_count число записей в буфере инициатора (неотрицательное целое)

origin_datatype тип данных каждой записи в буфере инициатора (дескриптор)

target_rank номер получателя (неотрицательное целое)

target_disp смещение от начала окна до буфера получателя (неотрицательное целое)

target_count число записей в буфере получателя (неотрицательное целое)

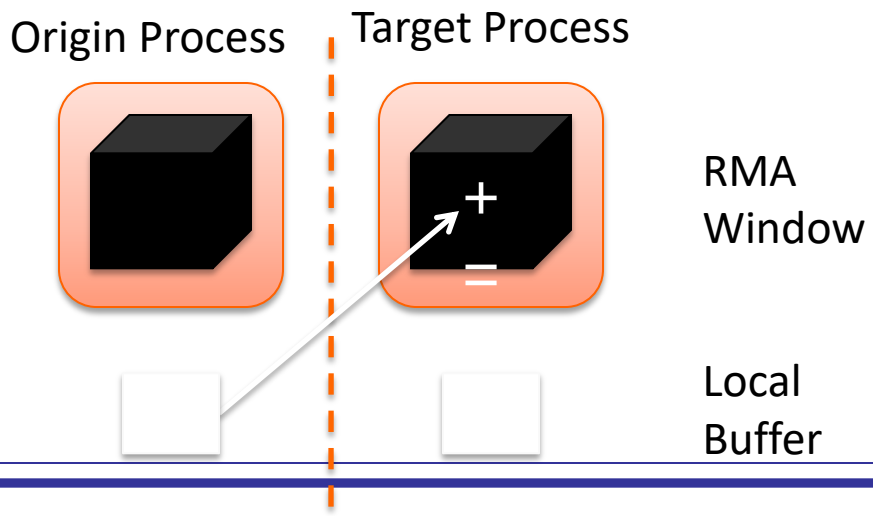
target_datatype тип данных каждой записи в буфере получателя (дескриптор)

win оконный объект, используемый для коммуникации (дескриптор)

Агрегирование данных: *Accumulate*

- Как MPI_Put, но с применением MPI_Op
  - Предопределенные операции только, нет user-defined!
- Результат в target буфере

- Put-подобное поведение с MPI_REPLACE (*f(a,b)=b*)
  - Атомарный PUT



```
int MPI_Accumulate(void *origin_addr, int origin_count,
    MPI_Datatype origin_datatype, int target_rank, MPI_Aint
    target_disp, int target_count, MPI_Datatype
    target_datatype, MPI_Op op, MPI_Win win)
```

origin_addr начальный адрес буфера (выбор)

origin_count число записей в буфере инициатора (неотрицательное целое)

origin_datatype тип данных каждой записи в буфере (дескриптор)

target_rank ранг адресата (неотрицательное целое)

target_disp смещение от начала окна до буфера адресата (неотрицательное целое)

target_count число записей в буфере адресата (неотрицательное целое)

target_datatype тип данных каждой записи в буфере адресата (дескриптор)

op операция - как в MPI_Reduce (дескриптор)

win оконный объект (дескриптор)

Барьерная синхронизация

- int MPI_Win_fence(int assert, MPI_Win win)

  ASSERT программное допущение (целое) (assert = 0)

  WIN объект окна (дескриптор)

- Вызовы MPI_WIN_FENCE должны как предшествовать, так и следовать за вызовами get, put или accumulate, которые синхронизируются с помощью вызовов fence.

Параметры синхронизации

- Задание режима синхронизации (параметр MPI_Info в операции синхронизации), может её ускорить

  Однако значение 0 всегда работает правильно.

- Значения MPI_Info:
  - **MPI_MODE_NOSTORE** – данные в локальном окне локально не изменялись,
  - **MPI_MODE_NOPUT** – данные в локальном окне не будут изменяться удаленными операциями Put / Accumulate
  - **MPI_MODE_NOPRECEDE** – не было локальных Get / Put / Accumulate
  - **MPI_MODE_NOSUCCEED** – не будет локальных Get / Put / Accumulate

  } только все процессы вместе

Барьерная синхронизация

```
MPI_Win_create (A, …., &win);
MPI_Win_fence (0, win);
if (rank == 0) {
MPI_Put (….., win);
MPI_Put (….., win);
……
MPI_Put (….., win);
}
```

```
MPI_Win_fence (0, win);
MPI_Get (…. , win);
MPI_Win_fence (0, win);
A[rank] = 4;
MPI_Win_fence (0, win);
MPI_Put ( … , win);
MPI_Win_fence (0, win);
```

Простейшие правила доступа

- Доступ к окну не должен пересекаться
- Локальные операции в окне должны отделяться от RMA-операций вызовом MPI_Win_fence

Синхронизация передач данных

- Время работы программы разделено на периоды, в которые происходят асинхронные передачи.
- В конце каждого периода происходит ожидание всех запущенных в нём команд передачи данных.
- Начало и конец периода определяется специальными командами синхронизации

Синхронизация

- **Active target** communication – оба процесса вовлечены в передачу данных
- **Passive target** communication – только origin process
- **Access epoch** – содержит RMA функции в origin. Стартует и завершает операции синхронизации.
- **Exposure epoch** – содержит RMA-вызовы в active target

Функции

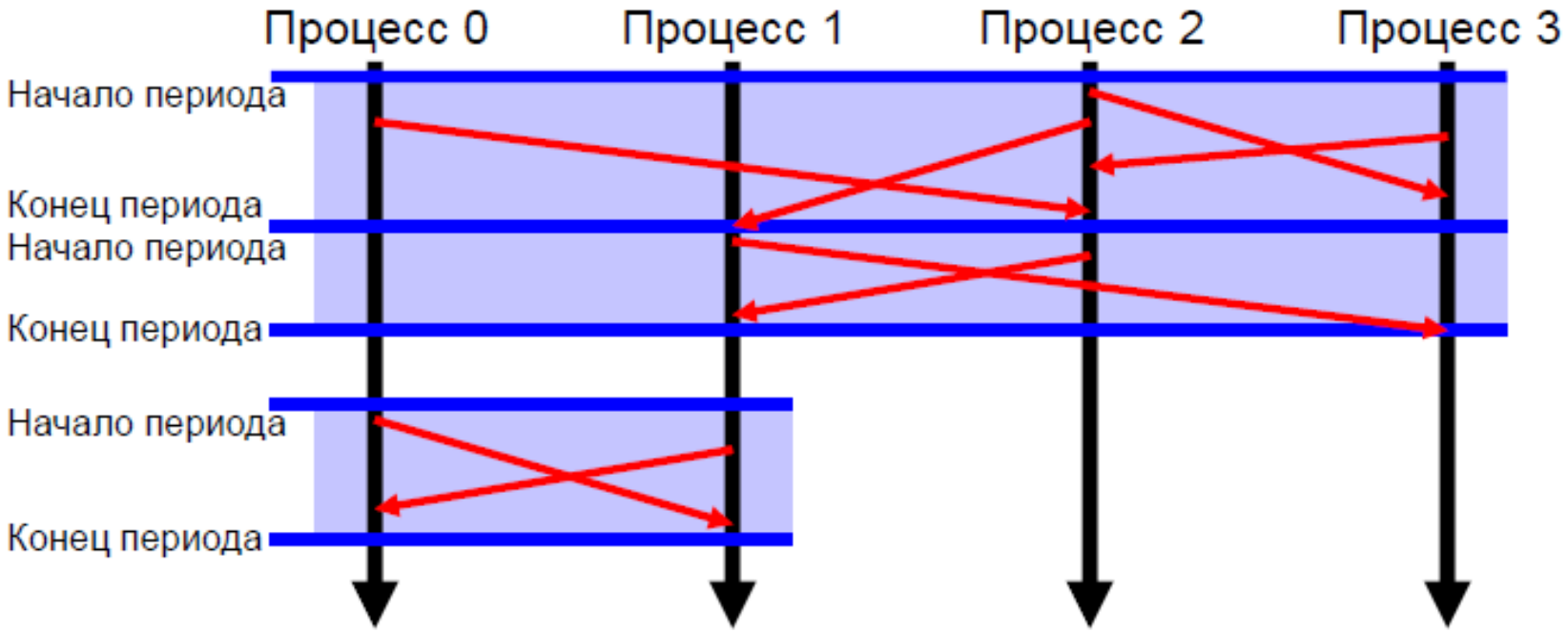
Два типа периодов обменов

- Период глобальных обменов – в обменах участвуют все процессы
- Период локальных обменов – процесс сам выбирает, с кем он обменивается

Функции синхронизации обменов:

- **MPI_Win_fence** – граница периода глобальных обменов
- **MPI_Win_start / MPI_Win_post / MPI_Win_lock** – начало периода локальных обменов
- **MPI_Win_complete / MPI_Win_wait / MPI_Win_unlock** – конец периода локальных обменов

Периоды обмена



Задание Info

```
MPI_Info info ;
int error ;
error = MPI_Info_create ( & info ) ;
error = MPI_Info_set ( info , "no_locks" , "true" ) ;
/* Use the info object */
error = MPI_Info_free ( info ) ;
```

Согласование периодов

- Период локальных обменов групп процессов

  - Процесс явно указывает группу процессов, которые будут обращаться в локальное окно.

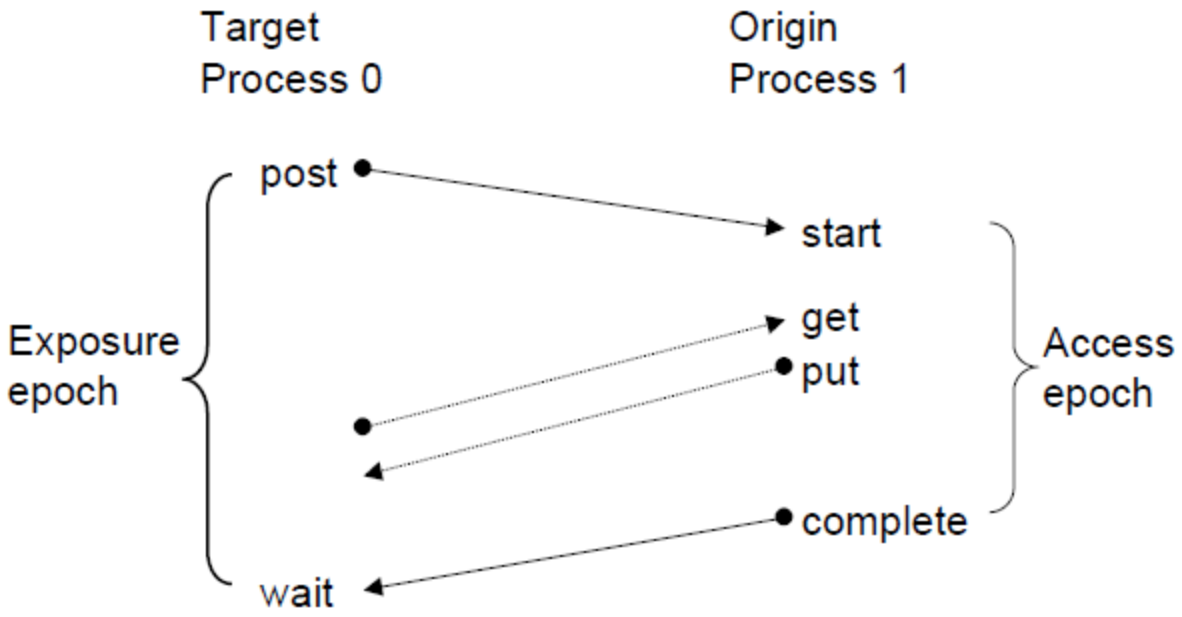
  - Процесс явно указывает группу процессов, в окно к которым он сам будет обращаться.

Начало периода, намерение обращаться в окна группе grp_to
MPI_Win_start(grp_to,0,win);
Операции доступа
MPI_Put/Get/Accumulate(…,win);
Конец периода
MPI_Win_complete(win);

Начало периода, предоставление локального окна группе grp_from
MPI_Win_post(grp_from,0,win);
Здесь в наше окно происходят обращения

Конец периода
MPI_Win_wait(win);

Start-Complete & Post-Wait



Start-Complete & Post-Wait

- RMA (put, get, accumulate) завершены:
  – локально после win_complete
  – для target после win_wait
- локальный буфер не должен использоваться до локального завершения RMA
  - взаимодействующие процессы должны быть известны
- нет атомарности для пересекающ “puts”
- assert могут улучшить эффективность

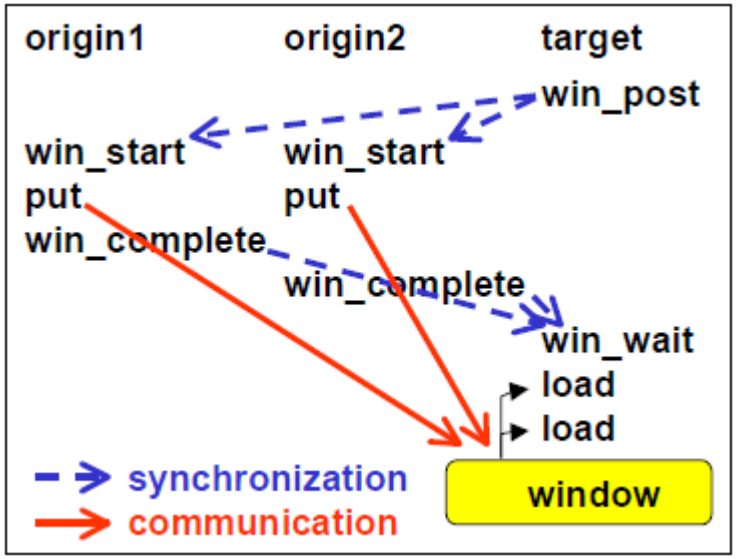
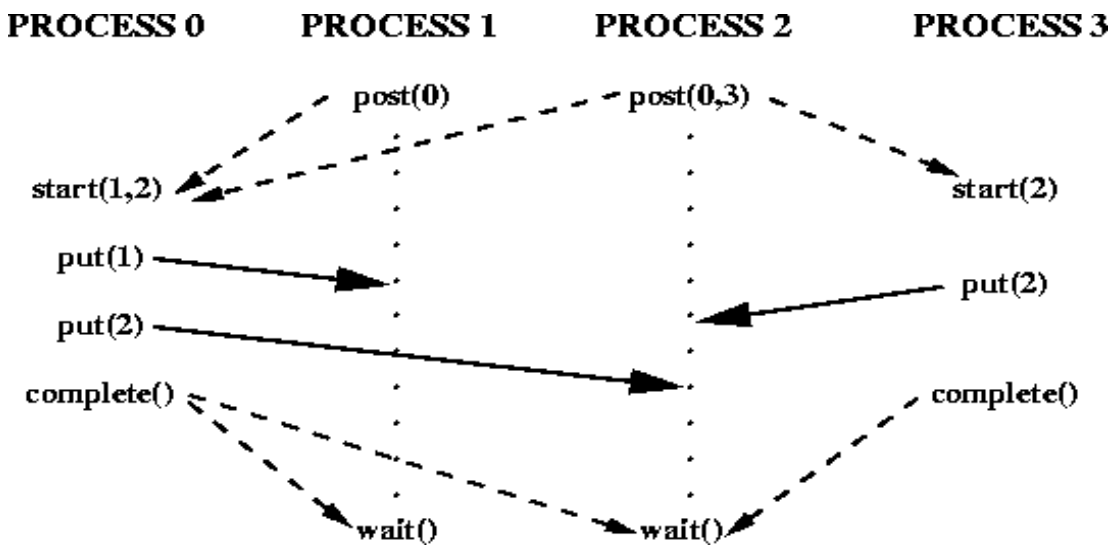


Иллюстрация – start, complete, post, wait

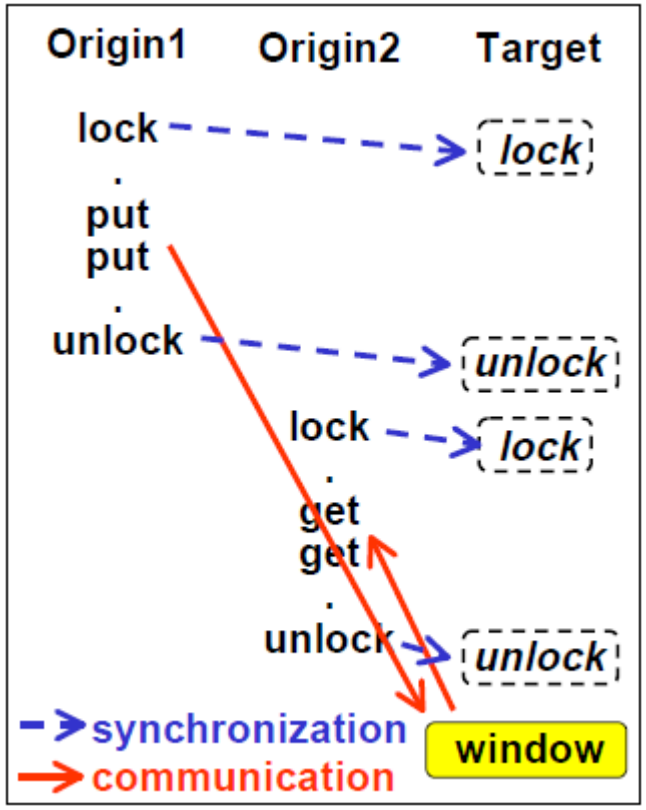


Lock-Unlock

- Можно использовать вызов

lock только для окон, созданных с использованием MPI_ALLOC_MEM

• RMA операции завершаются после **UNLOCK к**ак в origin, так и в target



MPI_Win_post

int MPI_Win_post (MPI_group group , int assert , MPI_Win window)

Окно регистрируется для RMA доступа

Только процессы указанной группы могут получить доступ к окну

Int MPI_Win_wait (MPI_Win window)

Блокировка до завершения всех RMA

Все процессы должны вызвать MPI_Win_complete

Закрывается доступ для RMA операций

MPI_Win_start

int MPI_Win_start (MPI_group group , int assert,MPI_Win window)

Старт удаленного доступа для окна процессам, принадлежащим указанной группе. Вызвавший функцию процесс может быть блокирован до вызова соответствующего MPI_Win_post

int MPI_Win_complete (MPI_Win window)

Завершение удаленного доступа

Синхронизация Lock/Unlock

- **int MPI_Win_lock(int lock_type, int rank, int assert, MPI_Win win)**

  lock_type : MPI_LOCK_EXCLUSIVE (только одна операция в окне) или MPI_LOCK_SHARED (несколько операций могут быть в окне)

  rank - ранк блокированного окна (неотрицательное целое)

  assert - программный ассерт (целое)

  win - объект окна (дескриптор)

- **int MPI_Win_unlock(int rank, MPI_Win win)**

  rank - ранк окна (неотрицательное целое)

  win - объект окна (дескриптор)

  В случае MPI_LOCK_SHARED пользователь обеспечивает непересекающийся доступ к одной переменной

Ограничения на использование

int MPI_Alloc_mem(MPI_Aint size, MPI_Info info, void *baseptr)

Пример

```
MPI_Group grp_all,grp_from,grp_to; MPI_Win win;
int x,rank,prev,next;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_group(MPI_COMM_WORLD, &grp_all);
prev = (rank+size-1)%size;
next = (rank+1)%size;
MPI_Group_incl(grp_all, 1, &prev, &grp_from); // создаём группу
MPI_Group_incl(grp_all, 1, &next, &grp_to); // создаём группу
MPI_Win_create(&x, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD,
    &win);
MPI_Win_post(grp_from, 0, win); // открываем доступ к себе
MPI_Win_start(grp_to, 0, win); // заявляем, куда будем обращаться
MPI_Put(&rank,1,MPI_INT, next,0,1,MPI_INT, win); // отправляем
MPI_Win_complete(win); // дожидаемся завершения отправки
MPI_Win_wait(win); // дожидаемся завершения приёма
printf("%d : %d \n",rank,x);
MPI_Win_free(&win);
```

«Защищённый» доступ в окно другого процесса. Пример.

```
int *x, y, esize = sizeof(int);
MPI_Alloc_mem(2*esize,MPI_INFO_NULL,&x);
MPI_Win_create(x,2*esize,esize,MPI_INFO_NULL,
 MPI_COMM_WORLD,&win);
x[0] = rank; x[1] = rank;
MPI_Barrier(MPI_COMM_WORLD);
MPI_Win_lock(MPI_LOCK_SHARED,prev, 0, win);
MPI_Get(&y,1,MPI_INT, prev,0,1, MPI_INT,win);
MPI_Win_unlock(prev, win);
MPI_Win_lock(MPI_LOCK_EXCLUSIVE,next,0,win)
MPI_Put(&y,1,MPI_INT, next,1,1, MPI_INT,win);
MPI_Win_unlock(next, win);
MPI_Barrier(MPI_COMM_WORLD);
printf("%d : %d\n", rank, x[1]);
```

Ошибки

```
int one=1;
MPI_Win_create(…,&win);
…
MPI_Win_lock(MPI_LOCK_EXCLUSIVE,0,0,win);
MPI_Get(&value,1,MPI_INT,0,0,1,MPI_INT,win);
MPI_Accumulate(&one,1,MPI_INT,0,0,1,MPI_INT,MPI_SUM,win);
MPI_Win_unlock(0,win);
```

-Чтение и запись в одну переменную
- Порядок выполнения не гарантируется