Средства и системы параллельного программирования

сентябрь – декабрь 2021 г.

Лектор доцент Н.Н.Попова

Лекции 14, 15
13, 20 декабря 2021 г.

Тема

- Организация односторонних передач данных в MPI

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
- Отличается от концепции механизмов «Общая память»

Организация доступа в память другого процесса

- Односторонняя передача в MPI ограничивается доступом только к специально объявленной области памяти другого процесса :

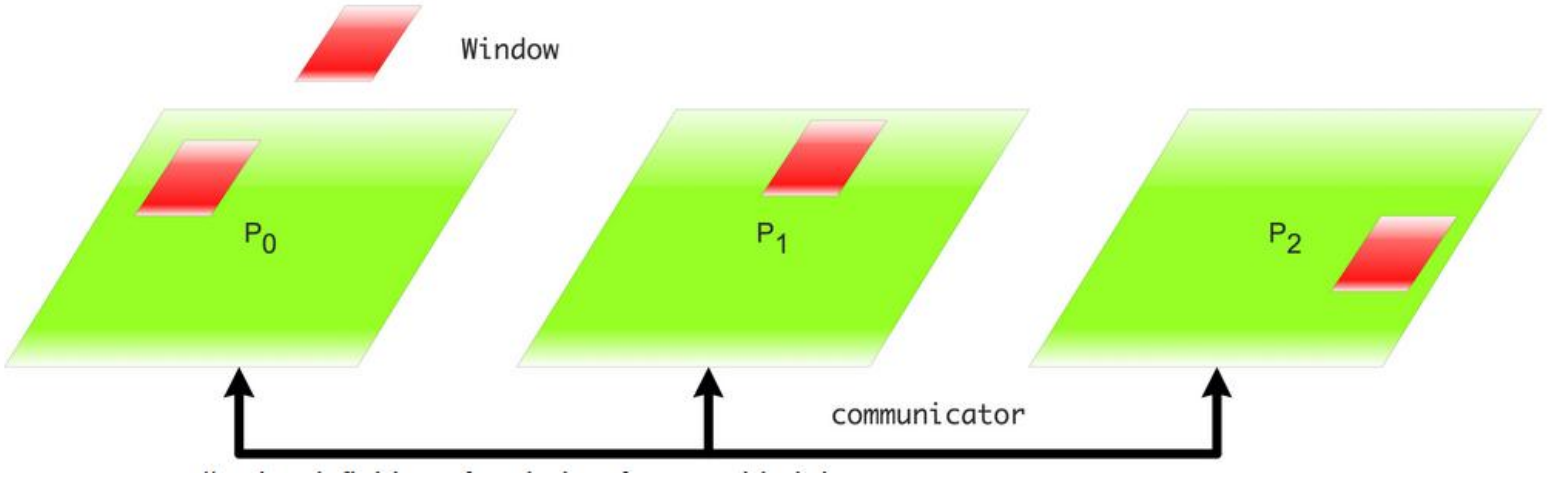
  - процесс объявляет область памяти пользовательского пространства, доступную для других процессов. Это называется **окном**.

  - Окно ограничивает доступ процессов-источников к памяти процесса: можно только "получать" данные из окна или "помещать" их в окно; вся остальная память недоступна для других процессов.

Понятие «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
- Размер окна устанавливается индивидуально в каждом процессе и может быть равным 0.

Графическая иллюстрация понятия окна



Режимы односторонних передач

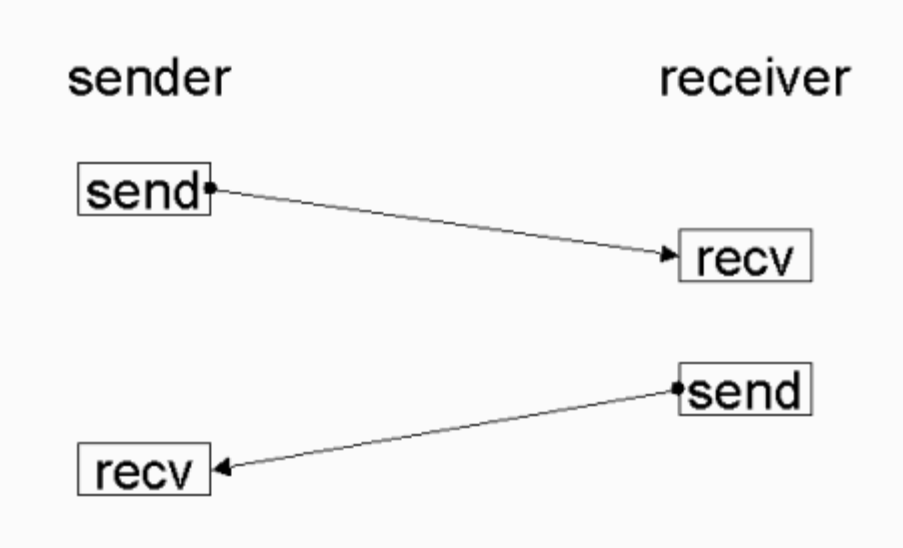
2 режима:

- **Active** - процесс устанавливает некоторый период (называется **эпохой**), в течение которого возможен доступ к его окну со стороны других процессов.
- **Passive** – нет ограничений на доступ к окну. Сложности с синхронизацией, трудно отлаживается,

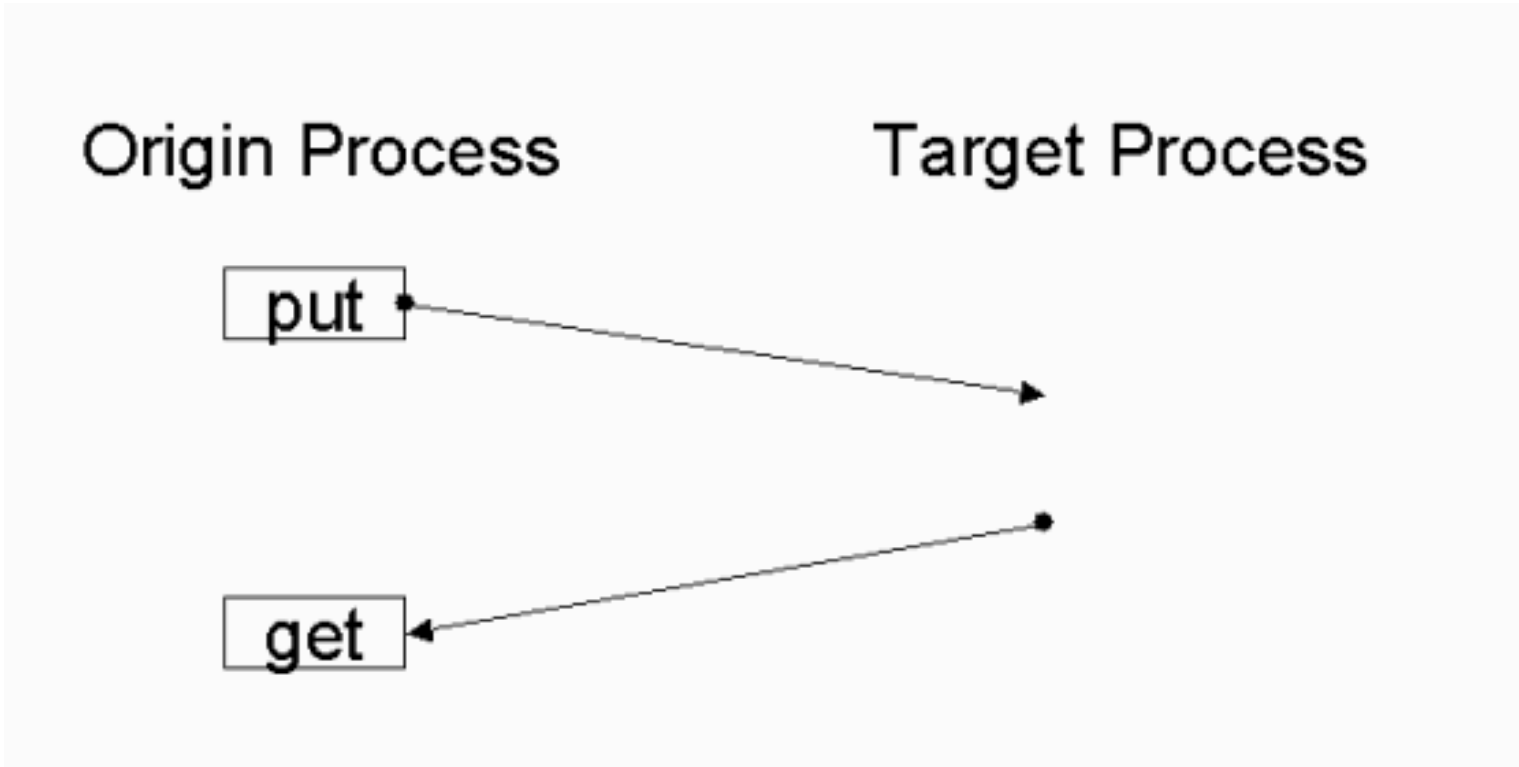
Терминология

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

Взаимодействие: двухсторонние передачи



Взаимодействие: односторонние (RMA)



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

Пример шаблона работы с окном

```
MPI_Info info;
MPI_Win window;
 MPI_Win_allocate( /* size info */, info, comm,
    &memory, &window );
 // do put and get calls
MPI_Win_free( &window );
```

Задание области памяти для окна

Способы задания области памяти для окна

- Обычный буфер в памяти

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

- Буфер в динамической памяти

- int MPI_Win_allocate (MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, void *baseptr, MPI_Win *win)

- *MPI_Win_allocate_shared*

- *MPI_Win_create_dynamic*

Специальная возможность использования динамической памяти

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

- В некоторых системах передачи сообщений и RMA операции выполняются быстрее при обращении к специально выделенной памяти (например, память, которая совместно используется другими процессами в системе обмена данными группа по SMP). MPI обеспечивает механизм распределения и освобождения такой особенной памяти. Использование такой памяти для передачи сообщений или RMA не является обязательным. Эта память может использоваться без ограничений как любая другая динамически выделяемая память. Тем не менее, реализации могут ограничить использование некоторых функций RMA только для так определенной памяти

Ограничения на использование

- RMA **может** быть быстрее при использовании памяти, возвращаемой MPI_Alloc_mem() по сравнению с malloc() . Параметр MPI_Info можно использовать для оптимизации расположения памяти (значение этого параметра является специфичным для реализации и не подпадает под стандарт, MPI_INFO_NULL всегда будет работать).
- Некоторые реализации MPI могут выполнять выравнивание возвращаемой памяти MPI_Alloc_mem() по строке кэша, что потенциально может привести к повышению производительности.
- код ошибки класса MPI_ERR_NO_MEM не хватает памяти

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;
}
```

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

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

```
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);
```

Перемещение данных

- Функции:
  - MPI_PUT
  - MPI_GET
  - MPI_ACCUMULATE (atomic)
  - MPI_GET_ACCUMULATE (atomic)
  - MPI_COMPARE_AND_SWAP (atomic)
  - MPI_FETCH_AND_OP (atomic)
- Все операции по перемещению данных – неблокирующие
- Синхронизация обязательна (чтобы убедиться, что операция завершена)

Active target synchronization: эпоха

Один из механизмов синхронизации

int MPI_Win_fence(int assert, MPI_Win win)

Промежуток между двумя последовательными вызовами MPI_Win_fence называется ЭПОХОЙ.

- Коллективная операция
- Посредством задания параметра assert можно передать системе дополнительную информацию

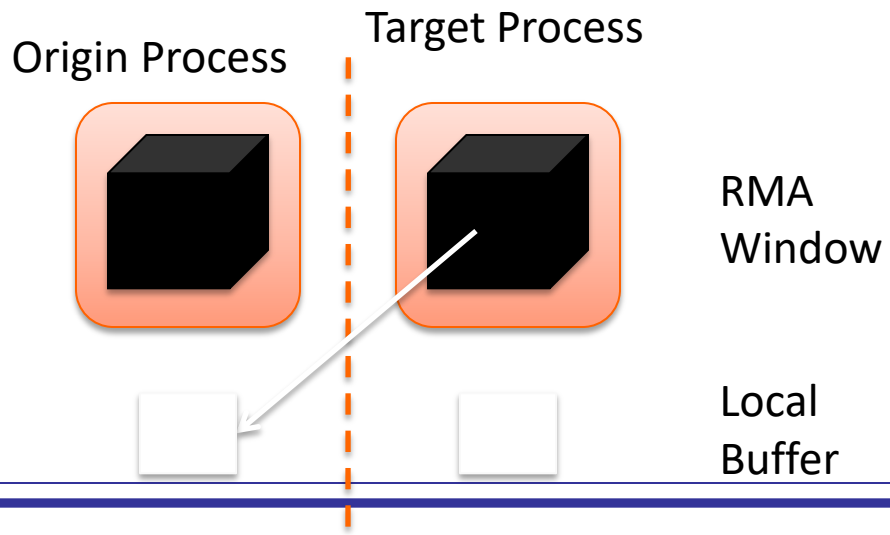
- *MPI_Win_fence((MPI_MODE_NOPUT | MPI_MODE_NOPRECEDE), win);*
  MPI_Get(/ operands */, win);*
- *MPI_Win_fence(MPI_MODE_NOSUCCEED, win);*

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

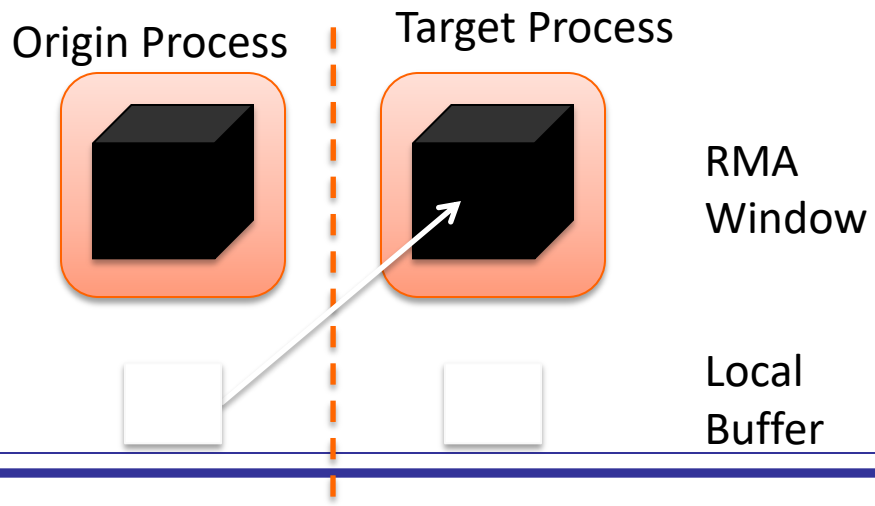
win оконный объект, используемый для коммуникации

Перемещение данных: *Put*

MPI_Put(

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

- Пересылка данных **из** origin **в** 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)

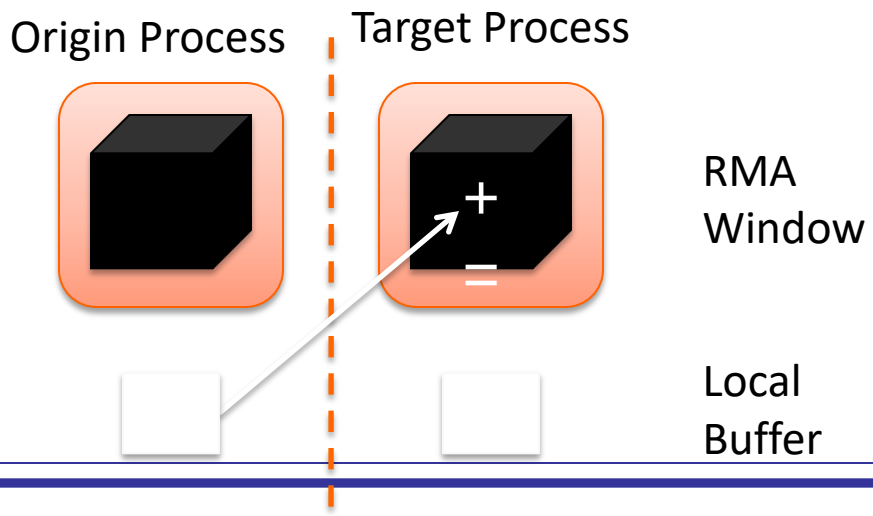
target_datatype тип данных каждой записи в буфере получателя (дескриптор)

win оконный объект, используемый для коммуникации (дескриптор)

Агрегирование данных: *Accumulate*

- Как MPI_Put, но **атомарная** 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)

MPI_Get_accumulate

```
int MPI_Get_accumulate(const void *origin_addr,
    int origin_count, MPI_Datatype origin_dtype,
    void *result_addr,int result_count,
    MPI_Datatype result_dtype, int target_rank,
    MPI_Aint target_disp,int target_count,
    MPI_Datatype target_dype, MPI_Op op, MPI_Win win)
```

Atomic read-modify-write

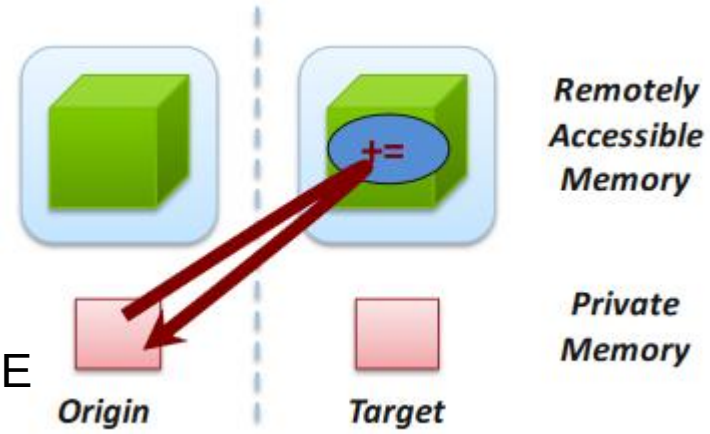
Операции – как в MPI_Reduce

Результат – в target buffer

Original data – в result buf

Атомарный get с MPI_NO_OP

Атомарный swap при MPI_REPLACE



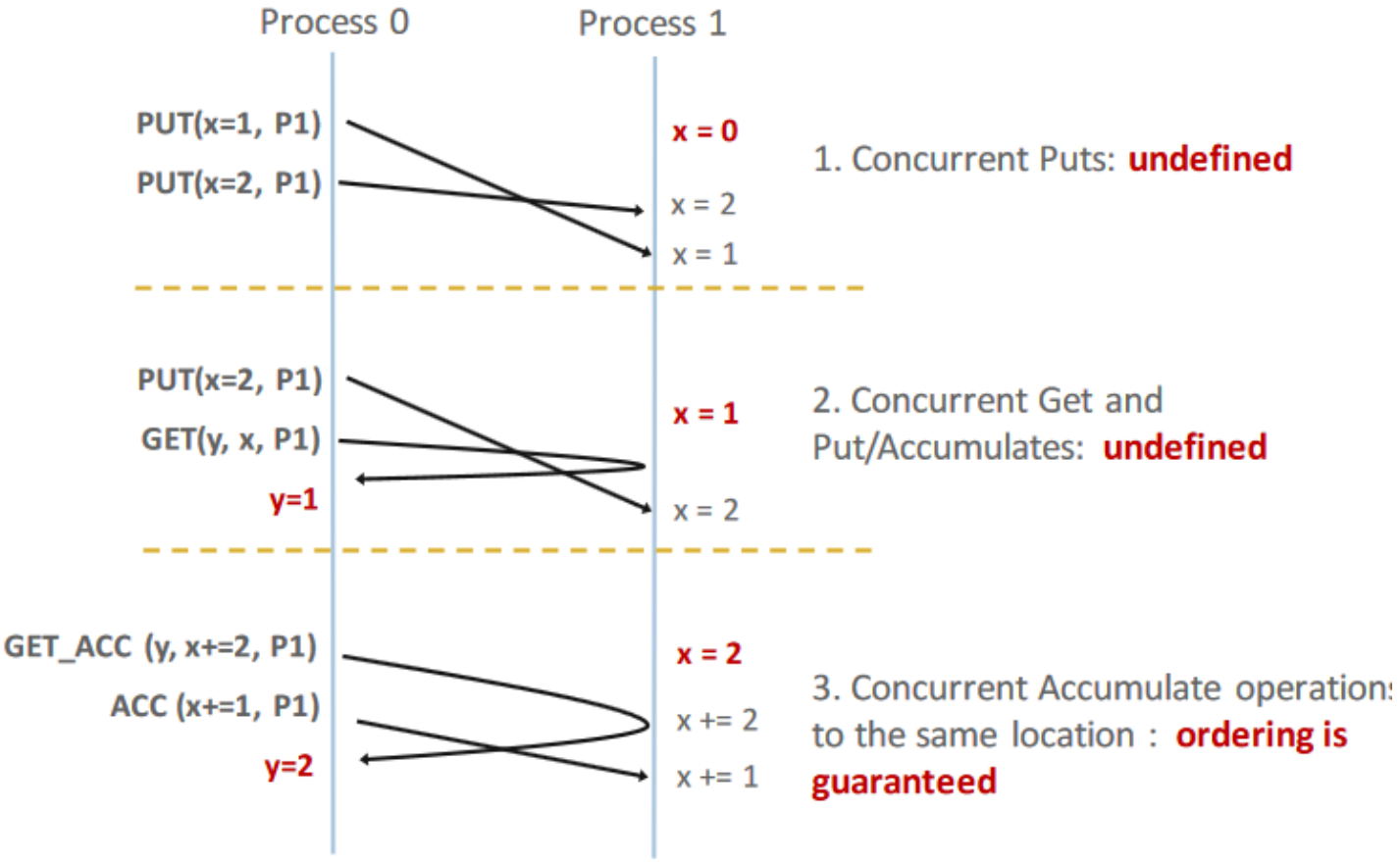
Упорядочивание выполнения односторонних операций

- Нет упорядочивания в течение эпохи по выполнению операций Get и Put/Accumulate
- Нет упорядочивания выполнения нескольких операций Put

Гарантируется

- Упорядочивание нескольких операций Accumulate
- Для упорядочивания нескольких Put можно выполнить Accumulate с операцией *MPI_REPLACE*

Пример с упорядочиванием операций



RMA модель синхронизации

- RMA data access model
  – When is a process allowed to read/write remotely accessible memory?
  – When is data written by process X is available for process Y to read?
  – RMA synchronization models define these semantics
- Three synchronization models provided by MPI:
  – Fence (active target)
  – Post-start-complete-wait (generalized active target)
  – Lock/Unlock (passive target)
- Data accesses occur within “epochs”
  – Access epochs: contain a set of operations issued by an origin process
  – Exposure epochs: enable remote processes to update a target’s window
  – Epochs define ordering and completion semantics
  – Synchronization models provide mechanisms for establishing epochs
  • E.g., starting, ending, and synchronizing epoch

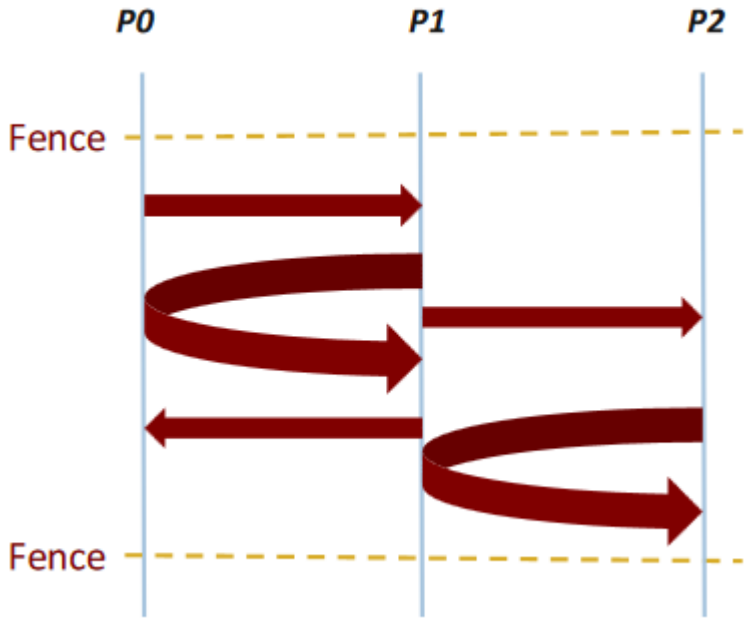
Active target барьерная синхронизация

- 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 fence



Барьерная синхронизация

```
MPI_Win_create (A, …., &win);
MPI_Win_fence (0, win);
if (rank == 0) {
/* Process 0 puts data into many local windows */
MPI_Put (….., win);
MPI_Put (….., win);
……
MPI_Put (….., win);
}
/* This fence completes the MPI_Put operations initiated by process 0 */
MPI_Win_fence (0, win);
/* All processes initiate access to some window to extract data */
MPI_Get (…. , win);
/* The following fence completes the MPI_Get operations */
MPI_Win_fence (0, win);
```

Барьерная синхронизация

```
/* After the fence, processes can load and store into A, the local window */
A[rank] = 4;
/* We need a fence between stores and RMA operations */
MPI_Win_fence (0, win);
MPI_Put ( … , win);
/* The following fence completes the preceding Put */
MPI_Win_fence (0, win);
```

Ошибочный код

```
/* This code has undefined behavior */
double b[10]; for (i=0; i<10; i++)
 b[i] = rank * 10.0 + i;
MPI_Win_create (b, 10*sizeof (double} , sizeof (double},
    MPI_INFO_NULL, MPI_COMM_WORLD, &win} ;
MPI_Win_fence (0, win} ;
 if (rank == 0) {   b[2] = 1./3.;
 else if (rank == 1) { /* Store my value of b into process 0's window,
    which is process 0's array b */
MPI_Put (b, 10, MPI_DOUBLE, 0, 0 , 1 0 , MPI_DOUBLE, win} ;
}
MPI_Fence (0,win);
```

Простейшие правила доступа

- Не допускать доступ к пересекающимся областям в окне
- 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

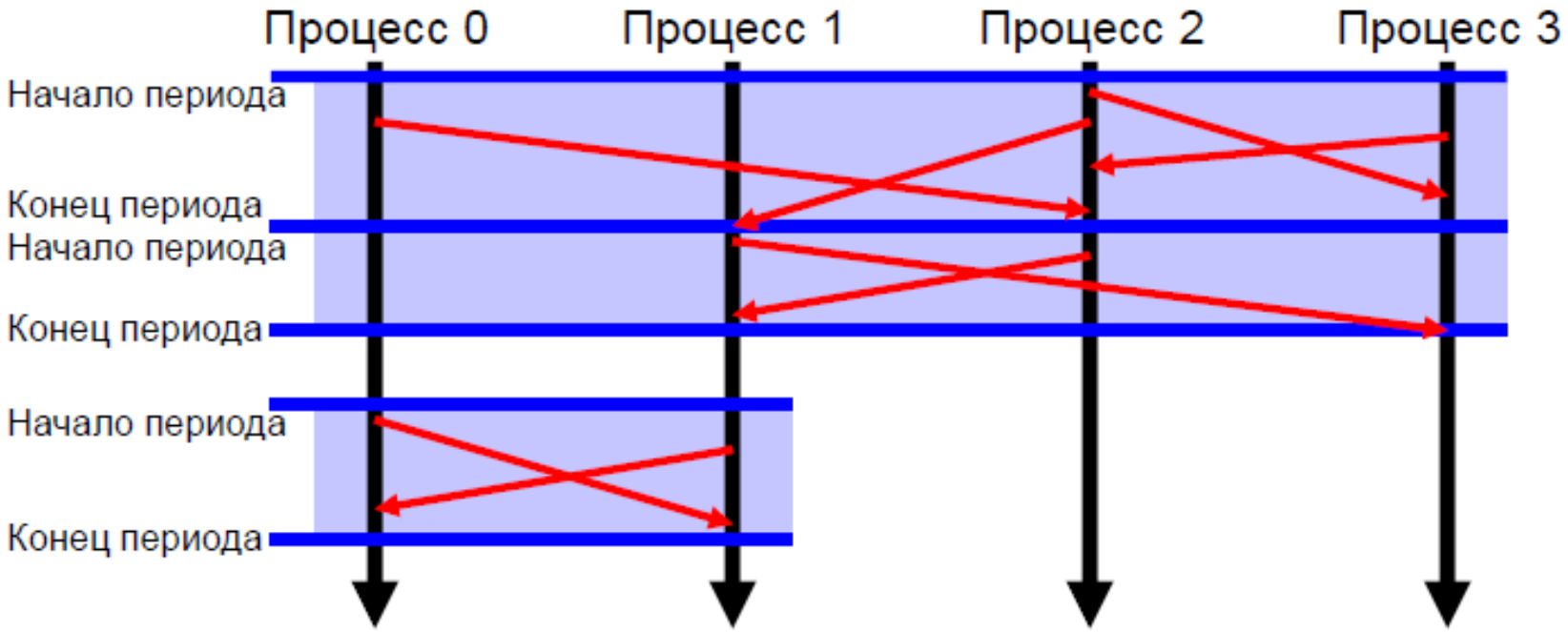
Два типа периодов обменов

- 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



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.

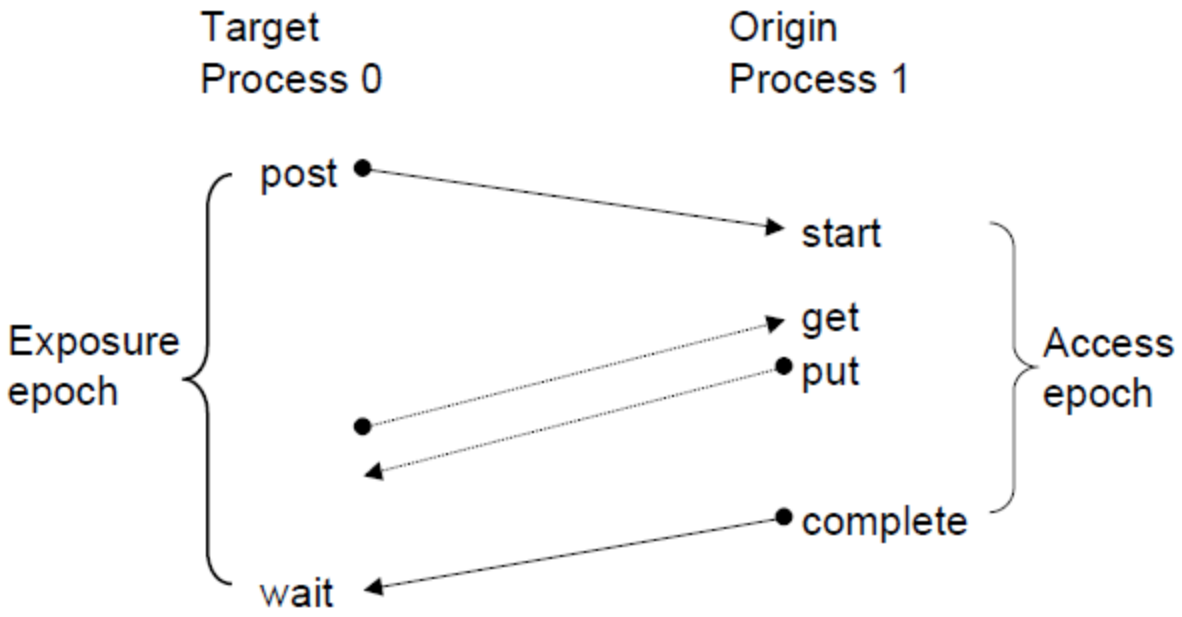
Начало периода, намерение обращаться в окна группе grp_to
MPI_Win_start(grp_to,0,win);
Операции доступа
MPI_Put/Get/Accumulate(…,win);
Конец периода
MPI_Win_complete(win);

→

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

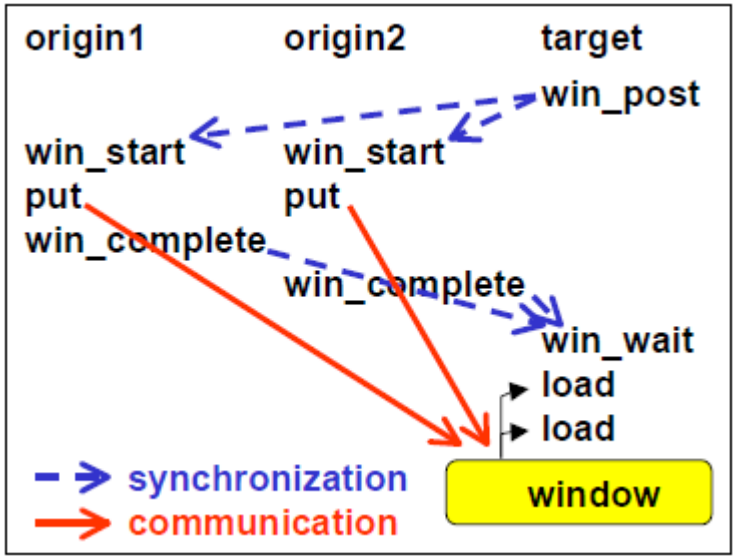
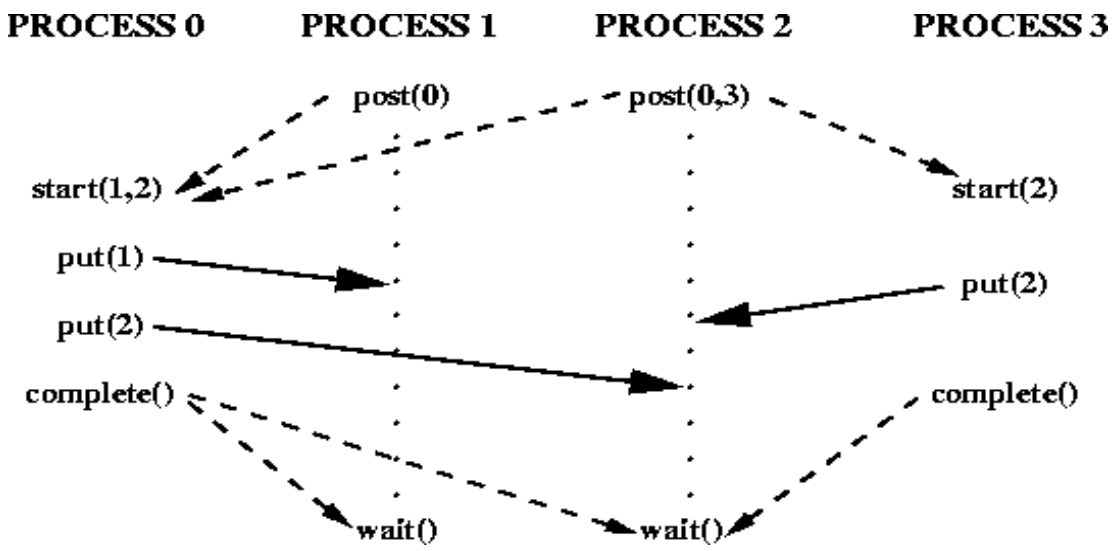
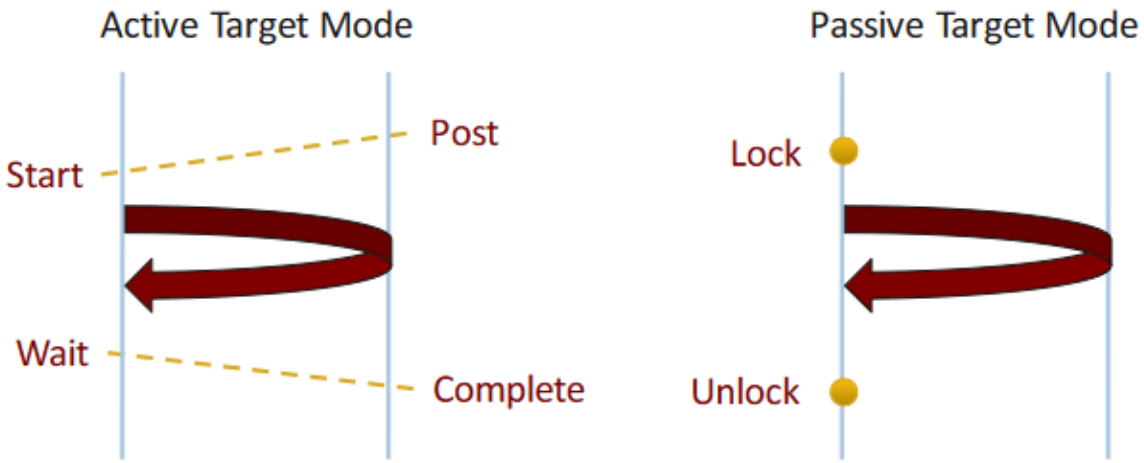


Иллюстрация – start, complete, post, wait



Lock-Unlock: Passive Target Synchronization



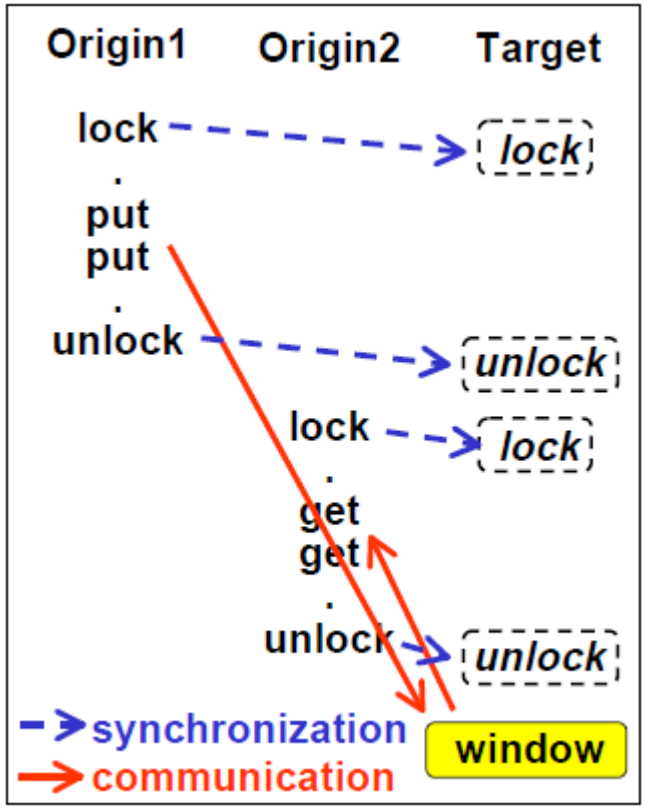
Passive mode: One-sided, asynchronous communication
– Target не участвует в операциях передачи данных
- Аналог режима рабоы с разделяемой памятью

Lock-Unlock: Passive Target Synchronization

- 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_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

Синхронизация Lock/Unlock

- **int MPI_Win_lock_all (int assert, MPI_Win win)**

  assert - программный ассерт (целое)

  win - объект окна (дескриптор)

- **int MPI_Win_unlock_all (MPI_Win win)**

  win - объект окна (дескриптор)

  В случае MPI_LOCK_SHARED пользователь обеспечивает непересекающийся доступ к одной переменной

  Lock_all: Shared lock, passive target epoch to all other processes
  – Expected usage is long-lived: lock_all, put/get, flush, ..., unlock_al

MPI_Win_flush

- int MPI_Win_flush (int rank, MPI_Win win)
- int MPI_Win_flush_local (int rank, MPI_Win win)

rank – номер процесса

win - объект окна (дескриптор)

Flush: удаленно завершает RMA operations в target процессе – После завершения данные могут быть process

Flush_local: локально завершает RMA операции для target процесса

Пример 1 (1)

```
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);
```

Пример 1 (2)

```
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