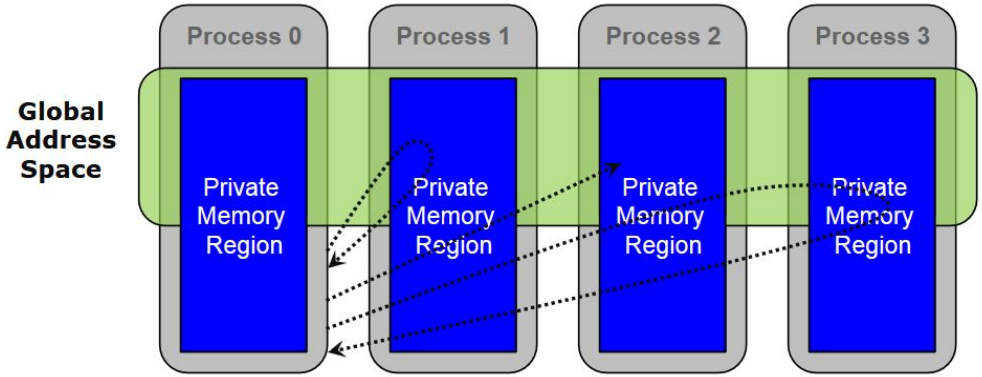
Средства и системы параллельного программирования

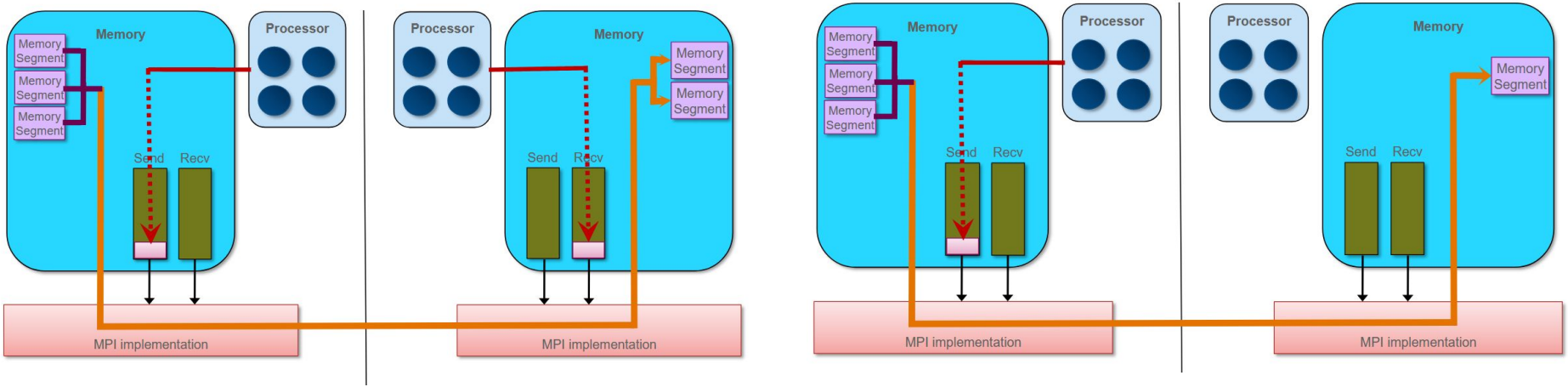
#11. MPI RMA

Что такое RMA (Remote Memory Access)

- Для получения сообщения процесс-получатель должен дождаться, пока процесс-отправитель *как минимум* дойдёт до точки отправки сообщения
- RMA призван убрать из данного сценария процесс-отправитель (то есть явного Recv не будет) и функционал отправки сообщения реализовать с помощью механизма окон



Two-sided vs one-sided модель



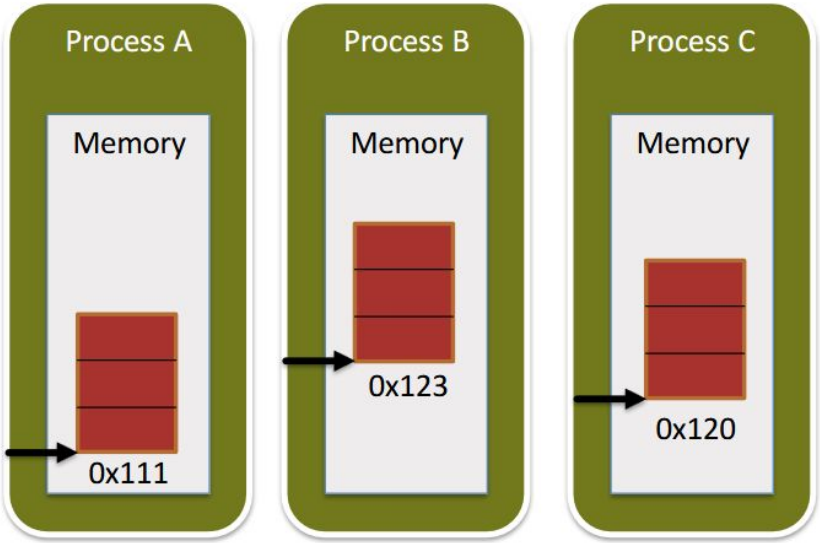
https://htor.inf.ethz.ch/publications/img/MPI_RMA_and_advanced_MPI.pdf

RMA vs RDMA

RDMA - (Remote **Direct** Memory Access) - это механизм, при котором сетевой адаптер может делать чтение и запись в пользовательские буферы памяти удалённого процесса без копий через буферы ядра и без участия удалённого CPU в этом сценарии.

Если на системе есть RDMA, то MPI старается использовать его для реализации операций RMA. Если нет, то функции, описанные дальше, по сути своей (внутри) ничем не отличаются от Send/Recv (если не брать в расчёт API).

Понятие окна



Каждый процесс может часть своей локально доступной памяти объявить общедоступной. Эта часть и будет называться окном.

Создание окна - коллективная операция

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

base - адрес начала предполагаемого окна на каждом процессе. Участок под окно должен быть непрерывным, память должна быть уже выделена

size - размер окна в байтах

disp_unit - единица адресации (когда будем использовать адресную арифметику в RMA-операциях, disp_unit будет шагом)

info - дополнительные параметры для окон (для оптимизации)

comm - коммуникатор, в который включены все процессы, которые будут работать с окном

win - инициализированный объект окна

Выделение памяти и создание окна

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

Почти те же самые параметры, что и у MPI_Win_create

И здесь, и в MPI_Win_create допускается иметь окна различных размеров на различных процессах. Окно размера 0 на каком-либо процессе также допустимо.

Что можем делать с такой памятью?

- MPI_GET
- MPI_PUT
- MPI_ACCUMULATE
- MPI_GET_ACCUMULATE
- MPI_COMPARE_AND_SWAP
- MPI_FETCH_AND_OP

Origin - процесс, инициировавший одностороннюю операцию

Target - процесс, в отношении которого была инициирована операция (в чьё окно стучится origin)

Получение данных из окна

```
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 - память, куда будем класть получаемые данные

origin_count, origin_datatype - сколько объектов какого типа получаем

target_rank - от какого процесса получаем

target_disp - сдвиг от начала окна на target процессе

target_count, target_datatype - сколько объектов какого типа "отправляем" от target

win - в рамках какого окна выполняем RMA-операции

Запись данных в окно

```
int MPI_Put(const 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)
```

Описание параметров как и у MPI_Get, только origin_addr - источник для “отправки” данных в target-процесс

Атомарная запись (с операцией) в память target-процесса

int **MPI_Accumulate**(const 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 - адрес начала "отправляемых" данных

origin_count, origin_datatype - сколько у нас данных и какого типа

target_rank - таргет процесса, где будем аккумулировать данные

target_disp - отступ от начала окна в таргете

target_count, target_datatype - сколько данных и какого типа

op - операция над данными (MPI_SUM, MPI_PROD, **MPI_REPLACE**, **MPI_NO_OP** и.т.д)

win - в в рамках какого окна выполняем RMA-операции

Атомарная запись (с операцией) в память target-процесса

```
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)
```

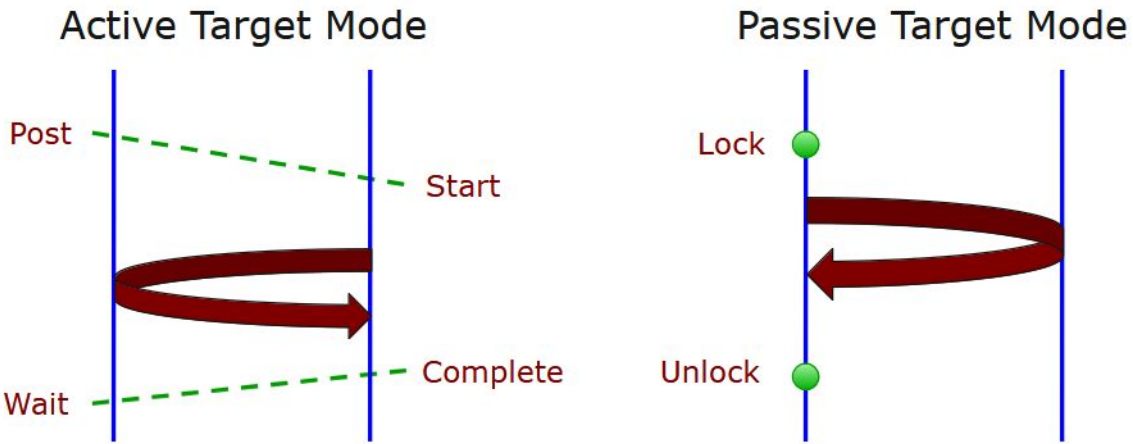
Тот же самый Accumulate, но в result_addr возвращаем значение на таргете ДО выполнения коммуникации.

Что делает? MPI_NO_OP

Почему мы так не писали раньше и в чём потенциальные трудности программирования с использованием RMA?

Синхронизация в RMA

Синхронизация бывает активной (в синхронизацию вовлекается target-процесс. Как будто target-процесс делает аналог MPI_Wait) и пассивной (происходит без участия target-процесса в синхронизации. Вспоминаем мьютексы в pthread/C++)



Синхронизация в RMA

```
MPI_Win_fence(int assert, MPI_Win win)
```

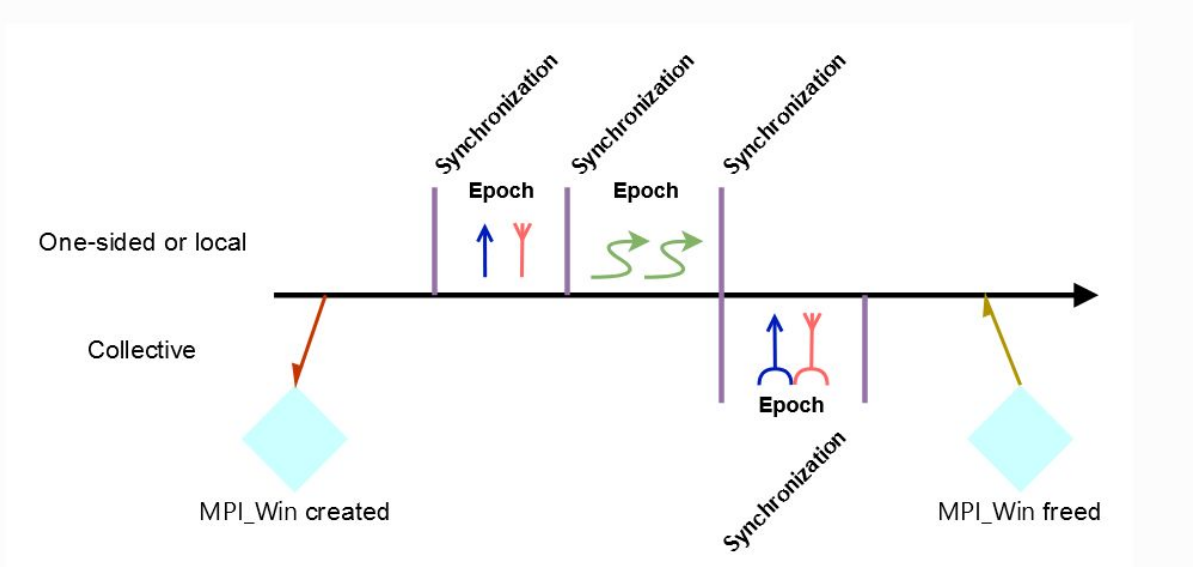
Синхронизация всех операций, *инициированных к данному времени* процессами, работающими с окнами

assert - проверка на некоторое условие (рекомендуется ставить 0)

win - в пределах какого окна действуем

MPI Epoch

Часть программы, находящаяся между двумя операциями синхронизации, называется **эпохой**



MPI Epoch

В пределах одной эпохи:

Не гарантируется очередность различных операций MPI_Put и MPI_Get от одного процесса

Не гарантируется очередность MPI_Put операций от различных MPI процессов

Не гарантируется корректный результат при использовании MPI_Put + MPI_Accumulate (MPI_Get_Accumulate)

Гарантируется очередность операций MPI_Accumulate (MPI_Get_Accumulate) от одного процесса

Задание

Прочитайте комментарий к теме и заданию в директории этого семинара

Используя для межпроцессного общения только операции MPI RMA, реализовать:

- Процесс с rank = 0 генерирует уникальный seed для каждого процесса из MPI_COMM_WORLD и раздаст их по процессам
- Каждый процесс генерирует K целочисленных ключей, причём функция генерации ключа должна быть тяжёлой (например, с использованием цикла). Также каждый процесс выделяет массив размера L.
- Для каждого сгенерированного ключа определяется номер процесса-владельца. Определите этот сценарий так, чтобы в общем случае разбиение было близко к равномерному (например, можно взять остаток от деления на comm_size, чтобы каждый процесс из MPI_COMM_WORLD обладал в итоге примерно равным числом ключей)
- Ключ необходимо записать в массив ключей, хранящийся на владельце. Если число записей в массиве владельца = L, процесс-генератор выводит ключ сразу же в консоль и запись к владельцу не производится
- Сценарий, когда сгенерировали все ключи, а потом начали их раздавать, недопустим. Отправляем данные владельцам на лету.
- Вид массива (который хранится на владельце) можете сделать любым (через одномерный список, вектор…)
- Запуски для отчёта выполняем на Polus через mpisubmit.pl. Построить график (таблицу) T(P), S(P), E(P) при фиксированном значении K, построить график (таблицу) T(K), S(K) при фиксированном значении P (возьмите >= 8).

Дедлайн: 15.12, 30.12(продлено)