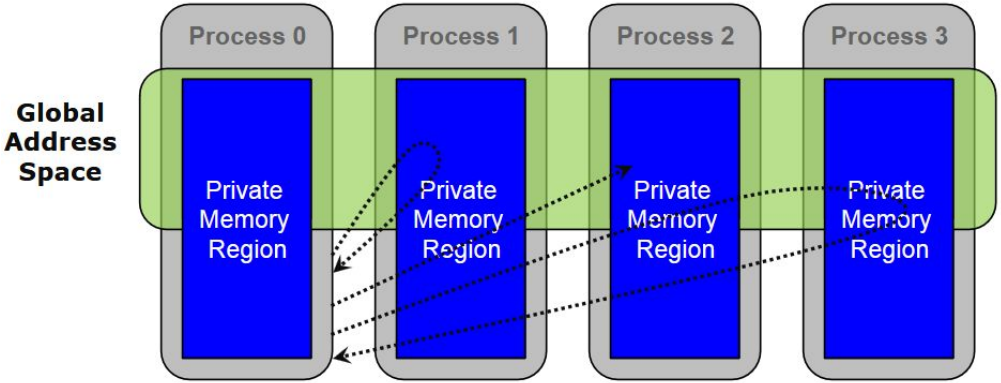
Средства и системы параллельного программирования

#11. MPI RMA

Что такое RMA

Что такое RMA

- 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 и функционал отправки сообщения возложить на *коммуникационную сеть*

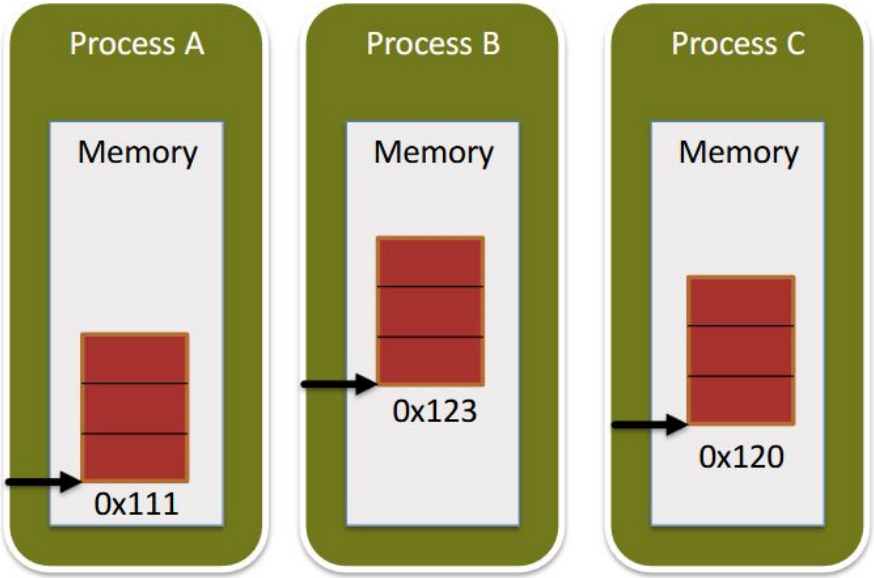


RMA vs RDMA

RDMA - (Remote Direct Memory Access) - технология, позволяющая получать доступ к памяти другого процесса без системных копирований (копируем сразу в шину передачи данных). В данном случае задействуется сетевой адаптер, а не CPU. Это снижает latency при передаче данных.

Реализации MPI, в которых внедрён механизм RMA, использует технологию RDMA.

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)
```

Те же самые параметры, что и у 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

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)
```

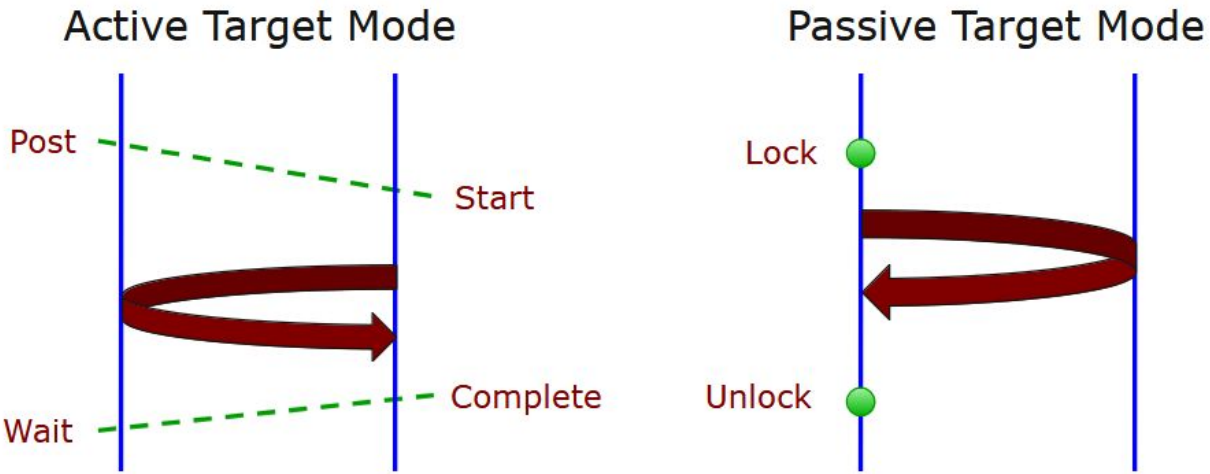
Тот же самый Accumulate, но в result_addr возвращаем значение на таргете ДО выполнения коммуникации.

Что делает? MPI_NO_OP

В чём потенциальные трудности программирования с использованием RMA?

Синхронизация в RMA

Синхронизация бывает активной (в синхронизацию вовлекается target-процесс) и пассивной (происходит без участия target-процесса в синхронизации)



Синхронизация в RMA

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

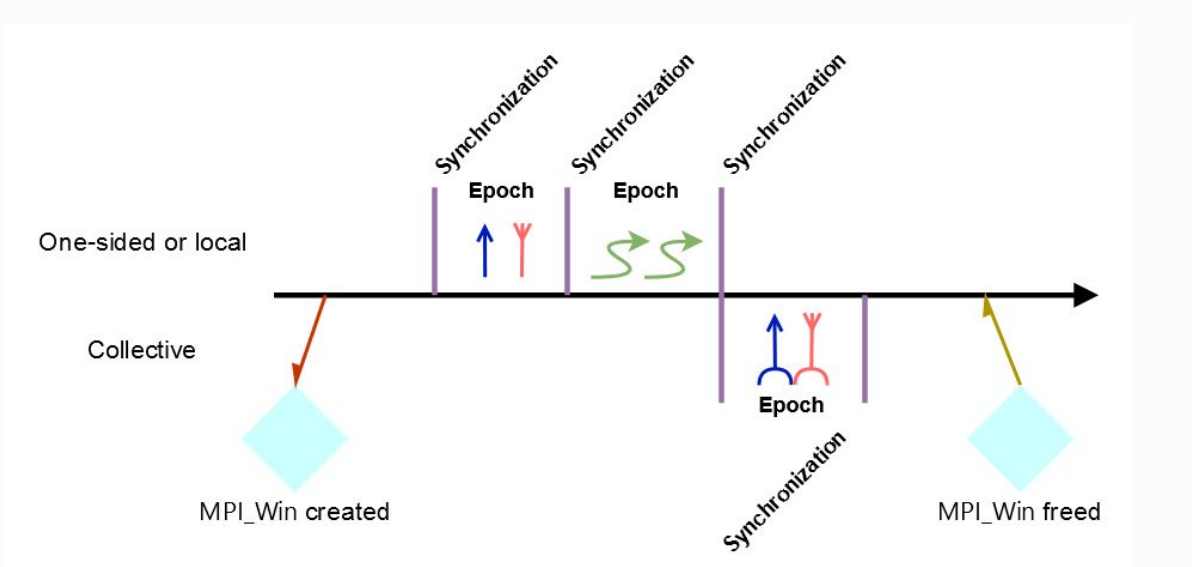
Синхронизация всех операций, *инициированных к данному времени* процессами, работающими с окнами

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

Не гарантируется очередность различных MPI_Put операций

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

Задание

Умножение матрицы на вектор A*b = c; (A_{N*N}, b_{N*1}, c_{N*1})

Требования:

Изначально на каждом процессе - прямоугольный блок матрицы A

Вектор b генерируется полностью процессом с ранком 0, остальные процессы могут его прочесть и забрать себе

Задача - произвести умножение матрицы на вектор с использованием **только** односторонних операций (RMA) для обмена данными.

Задание (2)

Требования:

Должна использоваться 2D процессная решётка, причём для составных P > 2 размерность каждого измерения должна быть больше 1

Тип данных - произвольный (float, int, double)

Результирующий вектор c можно собрать на процессе 0 операцией типа gather

Построить график T(P) при фиксированном большом значении N.

P = {1, 2, 4, 8, 12, 16}

Дедлайн: 25.12, 30.12