Параллельная обработка больших графов

Занятие 9

А.С. Семенов

dislab.org

НИЦ ВТ

DISLab

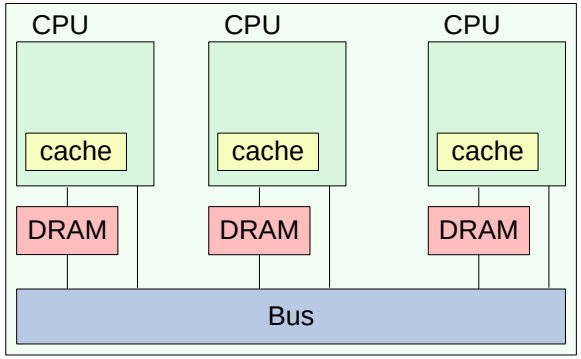
Общие методы оптимизации программ для систем с распределенной памятью

Библиотека MPI

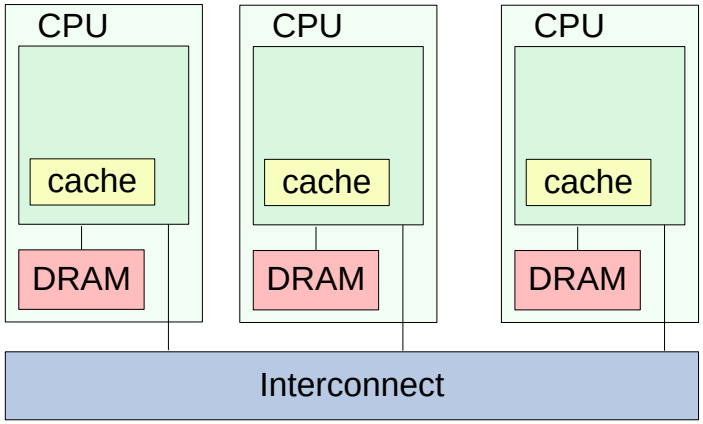
Классификация вычислительных систем

	Общее адресное пространство	**Раздельное адресное пространство**
Общая память	SMP (Symmetric Multiprocessing)	–
Распределенная память	NUMA (Non-Uniform Memory Access)	MPP (Massively Parallel Processing)

Общая память



Распределенная память



Программные модели:

- С общей памятью: pthreads, OpenMP, Intel TBB
- С распределенной памятью:
  - с глобальным доступом – SHMEM, UPC
  - передача сообщений – MPI

Факторы, влияющие на производительность MPI-программ

Вычислительная система

- Процессор – тип, частота, количество
- Подсистема памяти – конфигурация памяти и кэшей, пропускные способности DRAM-cache-CPU
- Сеть
- Операционная система

Коммуникационная сеть

- Топология, правила маршрутизации
- Аппаратная поддержка коллективных операций, синхронизаций
- Характеристики производительности в разных режимах работы

Приложение

- Алгоритм, его масштабируемость
- Соотношение объема коммуникаций к вычислениям
- Пространственная характеристика использования памяти и сети, размеры сообщений
- Балансировка нагрузки
- Ввод/вывод

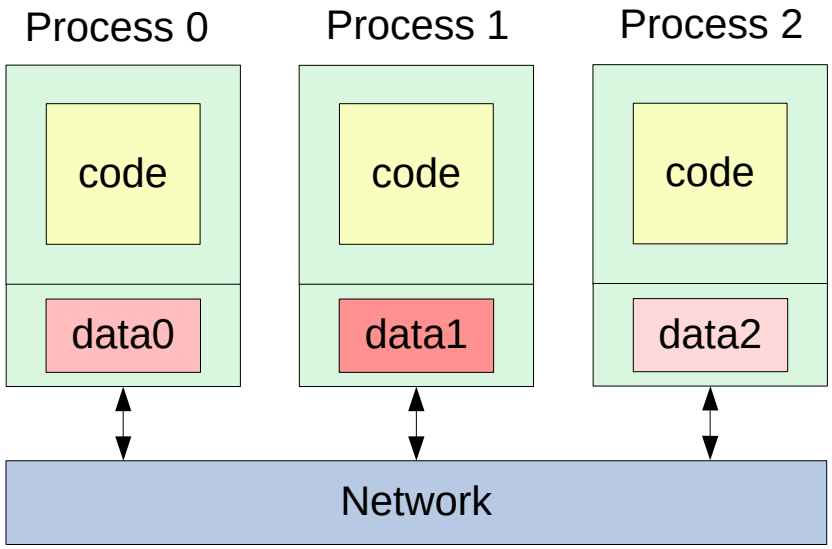
Характеристики реализации MPI и способы использования библиотеки

Message Passing Interface, MPI

- Основная цель: обеспечение переносимости программ, эффективная реализация, поддержка гетерогенных параллельных архитектур
- MPI-1 draft 1992, MPI-1.1 – 1995
- MPI-2 draft 1996, MPI-2.2 – 2009
- MPI-3 draft 2010
- Поддерживается на всех суперкомпьютерах, C, Fortran
- MPI-1 – около 125 функций, но часто используется лишь небольшое подмножество
  - Существуют эффективные реализации, используется большинством приложений
- Состав MPI-1:
  - Функции инициализации и завершения
  - Операции точка-точка
  - Коллективные операции

MPI

- Single Program Multiple Data (SPMD)
- Каждый процесс выполняет один и тот же код, но работает в своем адресном пространстве
- Все процессы порождаются во время запуска задачи, порождение новых не допускается
- Взаимодействие – при помощи отправки и приема сообщений



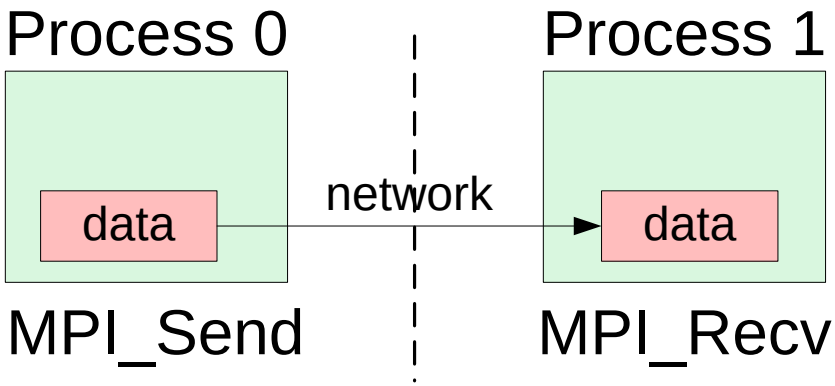
Коммуникаторы MPI

- Коммуникатор представляет группу процессов, которые могут общаться друг с другом
- Большинство функций MPI требуют в качестве аргумента коммуникатор
- ***MPI_COMM_WORLD*** собирает все MPI-процессы задачи
- Можно создавать новые коммуникаторы
- ***MPI_Comm_size*** *(comm, size, ierr)* возвращает количество процессов в коммуникаторе
- ***MPI_Comm_rank*** *(comm, rank, ierr)* возвращает номер процесса в коммуникаторе
- Чаще всего эти функции вызываются только после инициализации

```
program simple
      include ‘mpif.h’
      integer ierr, np, mype
      call MPI_Init(ierr)
      call MPI_Comm_size(MPI_COMM_WORLD, np , ierr)
      call MPI_Comm_rank(MPI_COMM_WORLD, mype, ierr)
      :
      call MPI_Finalize(ierr)
end program
```

Операции точка-точка

- ***MPI_Send (sendbuf, count, type, dest, tag, comm, ierr)*** – блокирующая посылка, возвращает значение, когда данные посланы из буфера
- ***MPI_Recv (recvbuf, count, type, source, tag, comm, status, ierr)*** – блокирующий прием, возвращает значение, когда данные получены в буфер



Тип MPI	**Тип Fortran**
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)

Протоколы передачи данных в MPI: Eager

Сообщение сразу посылается на узел-получатель, но, возможно, там буферизуется

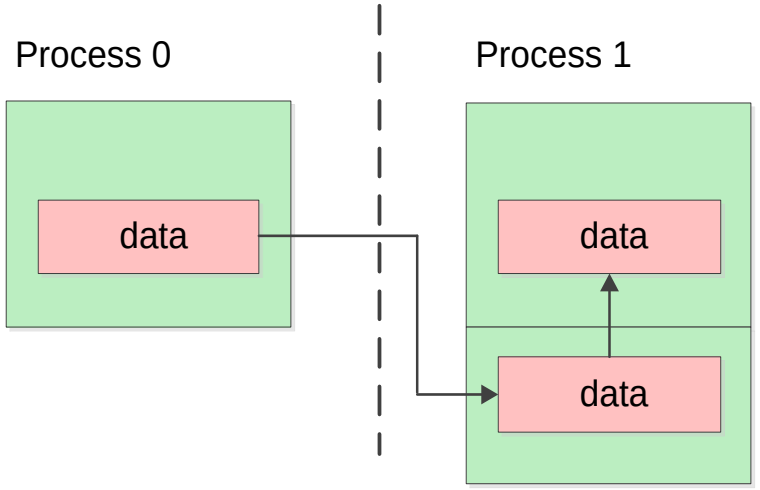
Преимущества:

- Уменьшается коммуникационная задержка

Недостатки:

- Требуется буферизация – протокол не масштабируем
- Расход памяти, даже если это не требуется
- Возможно переполнение буфера
- Может потребоваться дополнительное копирование

Используется для коротких сообщений



Протоколы передачи данных в MPI: Rendezvous

Когда нельзя предсказать состояние буфера приема или когда достигнуты ограничения eager

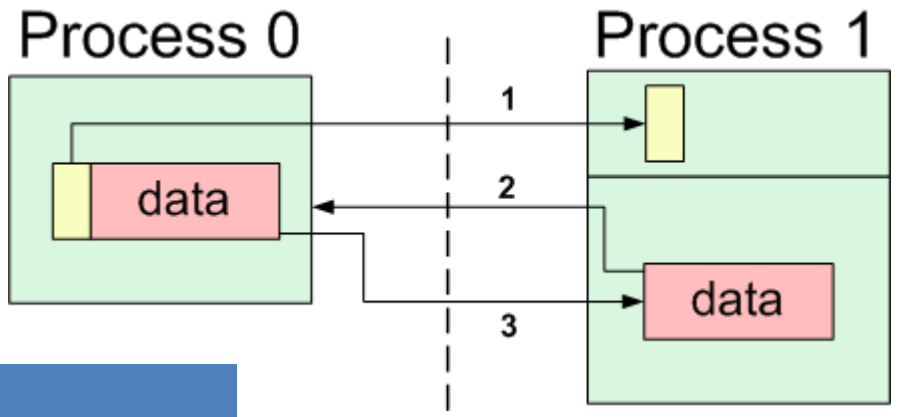
Преимущества:

- Масштабируем по сравнению с eager
- Требуется небольшой объем памяти для хранения метаданных сообщения
- Устойчивость к переполнению памяти на принимающем узле
- Отсутствие копирования данных

Недостатки:

- Растет задержка передачи сообщения из-за дополнительных коммуникаций

Используется для длинных сообщений



Переменные окружения:

- EAGER_LIMIT
- BUFFER_MEM

Эффективность программ:

Выдавать Recv раньше Send

Буферизация сообщений

Системная буферизация:

- На стороне отправителя
- На стороне получателя
- Может отсутствовать
- При некоторых условиях может присутствовать, при некоторых – нет, например, eager и rendezvous

Преимущество системной буферизации:

- Повышение производительности за счет возможной асинхронной передачи
- Если операция приема не была выдана, отправитель может продолжить работу

Недостатки:

- Требуется дополнительное копирование
- Буфер – конечный ресурс. Переполнение буфера – аварийное завершение или простой
- Не очевидно для программиста, когда используется буфер – зависимость от реализации MPI
- Программа, работающая при одних условиях, может не работать при других

- Размер системного буфера может задаваться переменной окружения
- Правильная MPI-программа не полагается на размер буфера.
- Небезопасная программа в большинстве ситуаций может работать корректно

Пример потенциально небезопасной программы

```
if (rank == 0) then
      do i = 1, LARGE_N
            call MPI_Send ( data, EAGER_LIMIT, MPI_REAL, 1, tag,
            MPI_COMM_WORLD, ierr)
       end do
elseif (rank == 1) then
      do i = 1, LARGE_N
            call MPI_Recv ( data, EAGER_LIMIT, MPI_REAL, 0, tag,
            MPI_COMM_WORLD, status, ierr )
            ! Do long computation
       end do
endif
```

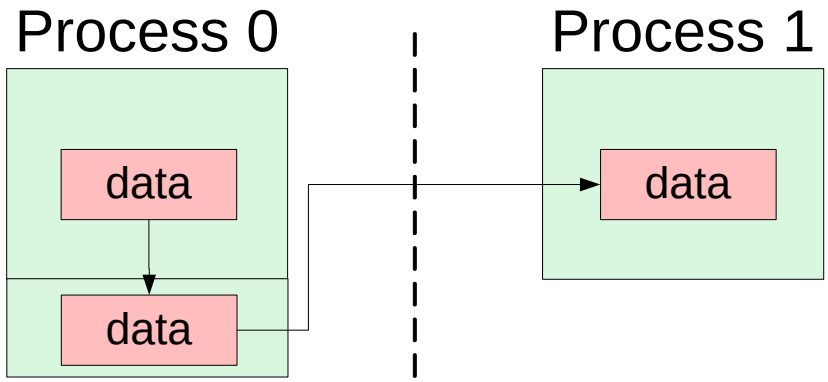
Пользовательская буферизация сообщений

- ***MPI_Bsend*** *(sendbuf, count, type, dest, tag, comm, ierr)*
- завершается, когда сообщение скопируется в буфер

Преимущества:
- Немедленное завершение (предсказуемость)
- Асинхронная передача

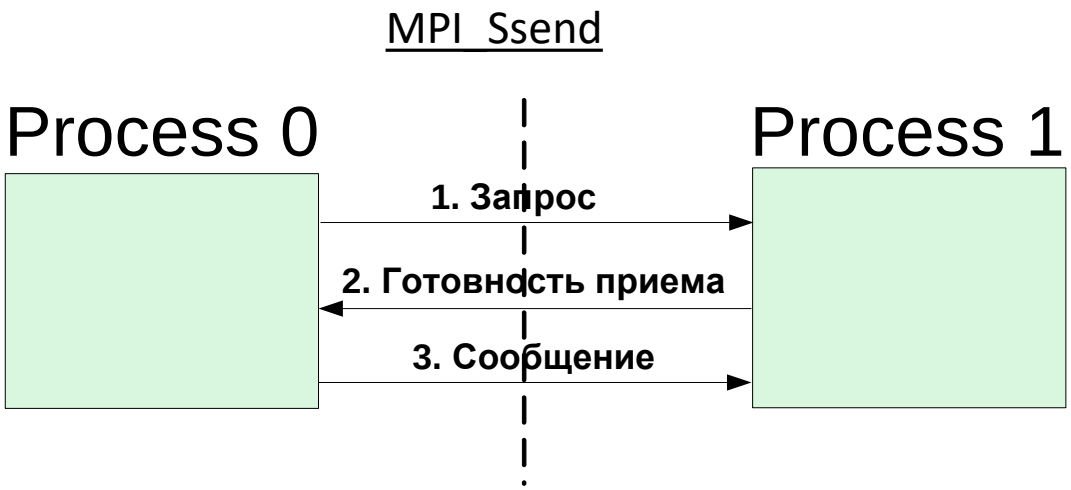
Недостатки:
- Требуется дополнительное копирование
- Требуется явно выделять буфер ***MPI_Buffer_attach*** *(buffer, size, ierr)* и контролировать его использование
- ***MPI_Buffer_detach*** *(buffer, size, ierr)*

Эффективность программ:
Из-за копирования лучше не использовать MPI_Bsend для длинных сообщений



Режимы передачи

Режим	Условие завершения
MPI_Send	Зависит от реализации (буферизация или начат прием)
MPI_Bsend	Завершается сразу после копирования
MPI_Ssend	Завершается только при условии начала приема
MPI_Rsend	Завершается после отправки (если прием был выдан) или ошибка



Решение проблемы

```
if (rank == 0) then
      do i = 1, LARGE_N
            call MPI_Ssend ( data, EAGER_LIMIT, MPI_REAL, 1, tag,
            MPI_COMM_WORLD, ierr)
       end do
elseif (rank == 1) then
      do i = 1, LARGE_N
            call MPI_Recv ( data, EAGER_LIMIT, MPI_REAL, 0, tag,
            MPI_COMM_WORLD, status, ierr )
            ! Do long computation
       end do
endif
```

Блокирующие и неблокирующие операции

Блокирующие

- Возвращают управление только если можно использовать буфер
- Для отправителя это означает, что модификация буфера не повлияет на отправляемое сообщение, но не подразумевает, что сообщение доставлено
- Может быть синхронным или асинхронным (с буферизацией)

Неблокирующие

- ***MPI_Isend*** *(sendbuf, count, type, dest, tag, comm, request, ierr)*
- ***MPI_Irecv*** *(recvbuf, count, type, source, tag, comm, request, ierr)*
- Сразу возвращают управление
- Нельзя использовать буфер до тех пор, пока операция не закончится (при помощи *MPI_Wait (request, status, ierr)*
- Отсутствует лишняя буферизация
- Возможно выполнение коммуникаций на фоне вычислений и, как следствие, выигрыш в производительности

```
call MPI_Irecv  ( A, count, MPI_REAL,  src, 0, MPI_COMM_WORLD,        reqs[0],
ierr )
call MPI_Isend ( B, count, MPI_REAL, dest, 0, MPI_COMM_WORLD,        reqs[1],
ierr )
 ! do computation
call MPI_Waitall ( 2, reqs, stats, ierr )
```

Эффективность программ:

- Стремиться использовать неблокирующие операции

Тупиковые (дедлоковые) ситуации

Дедлок 1

```
if (rank == 0) then
      call MPI_Recv ( …, 1, … )
      call MPI_Send ( …, 1, … )
elseif (rank == 1) then
      call MPI_Recv ( …, 0, … )
      call MPI_Send ( …, 0, … )
endif
```

Потенциальный дедлок 2

```
if (rank == 0) then
      call MPI_Send ( …, 1, … )
      call MPI_Recv ( …, 1, … )
elseif (rank == 1) then
      call MPI_Send ( …, 0, … )
      call MPI_Recv ( …, 0, … )
endif
```

Разрешение дедлоковых ситуаций

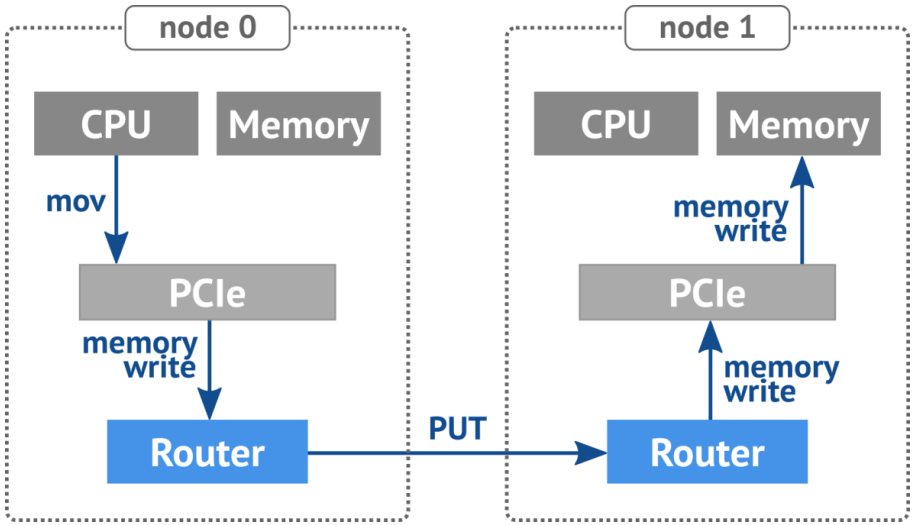
Решение

```
if (rank == 0) then
        call MPI_Irecv ( …, 1, …, request, …)
        call MPI_Send (…, 1, … )
elseif (rank == 1) then
        call MPI_Irecv ( …, 0, …, request, … )
        call MPI_Send ( …, 0, … )
}
call MPI_Wait ( request, status, ierr )
```

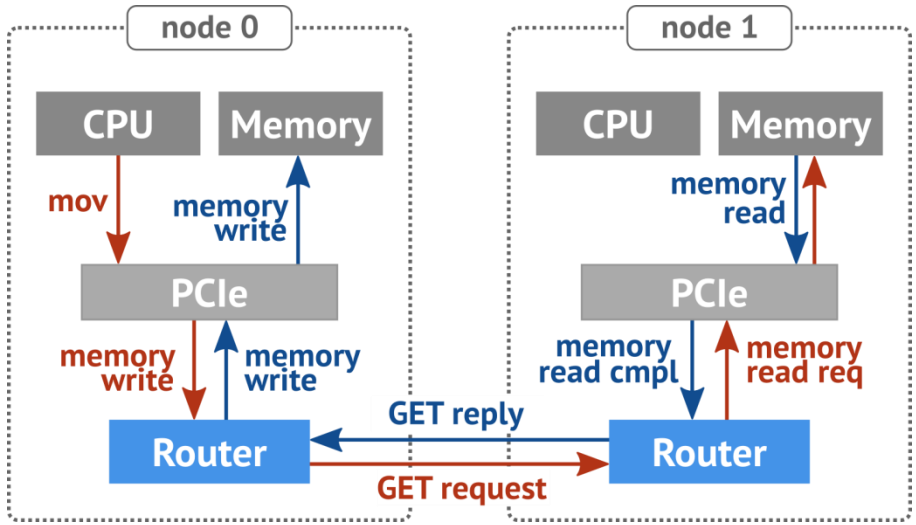
	Proc 0	Proc 1
Дедлок 1	Recv/Send	Recv/Send
Дедлок 2	Send/Recv	Send/Recv
Решение 1	Send/Recv	Recv/Send
Решение 2	SendRecv	SendRecv
Решение 3	Irecv/Send, Wait	Irecv/Send, Wait
Решение 4	BSend/Recv	BSend/Recv

Механизм передачи RDMA. Сеть Ангара

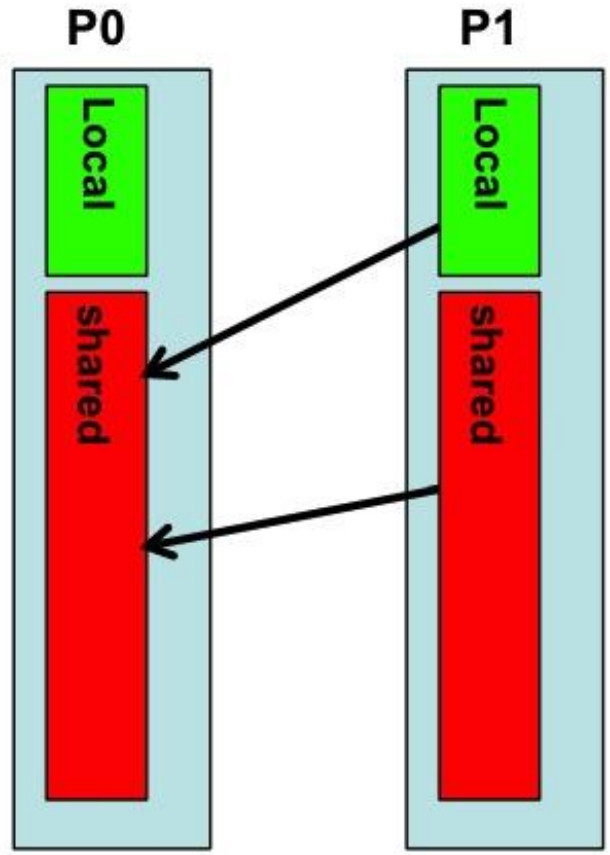
Удалённая запись



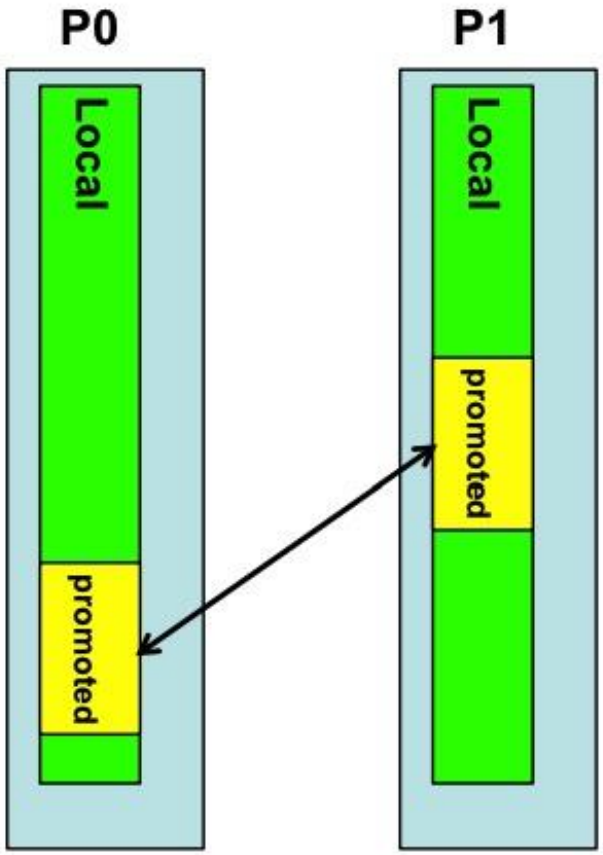
Удалённое чтение



Односторонние и двусторонние коммуникации



put/get operations



send/recv operations

MPI-3

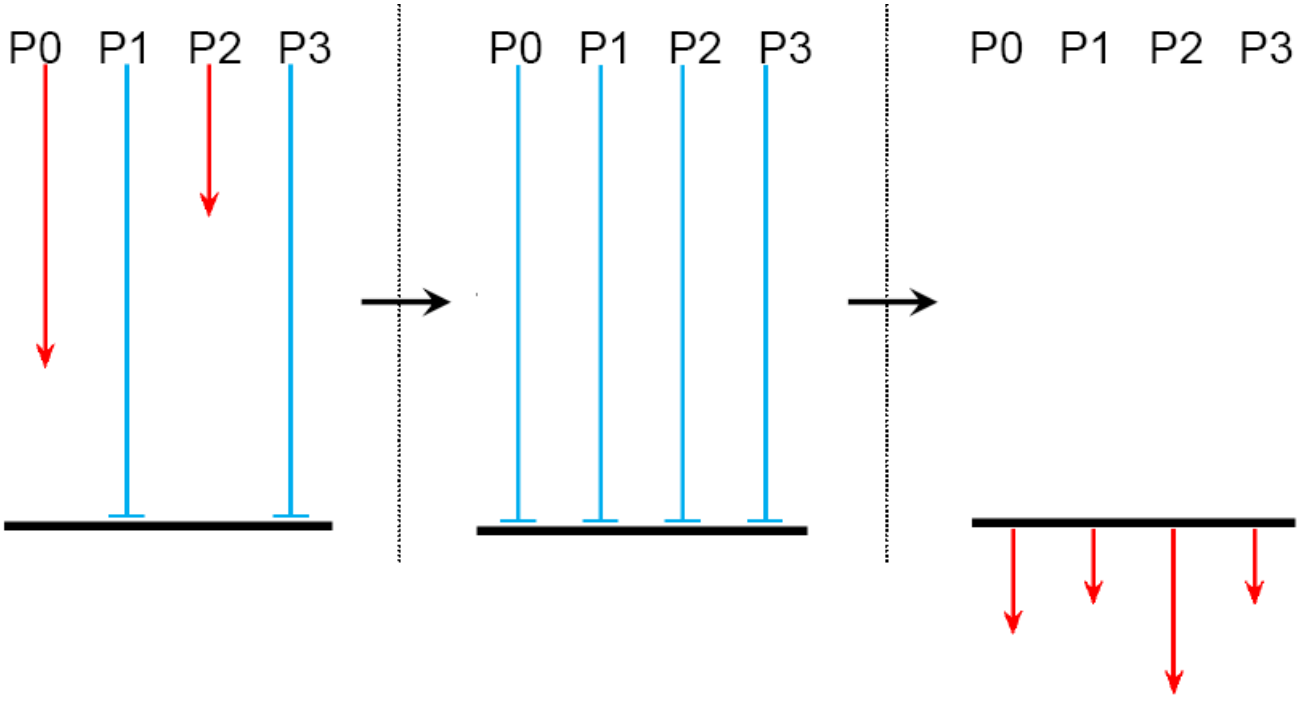
- ***MPI_Put*** *(sendbuf, count, type, dest, dest_disp, dest_count, dest_type, win)*
- ***MPI_Rput*** *(sendbuf, count, type, dest, dest_disp, dest_count, dest_type, win, request)*
  - *MPI_Test, MPI_Wait*
- **MPI_Win_flush** (dest, win)
- **MP_Win_flush_all** (win)

Коллективные операции

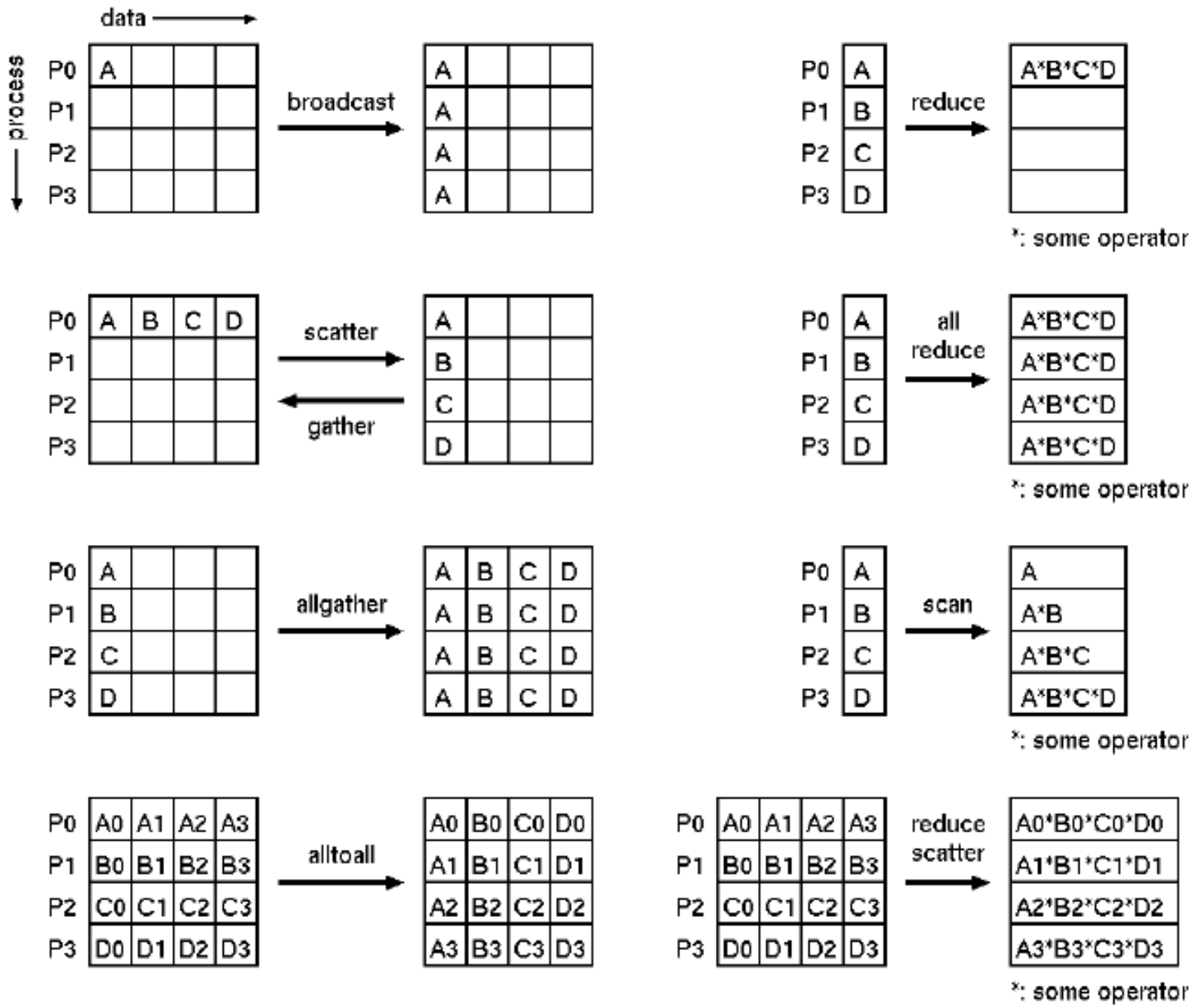
- Участвуют все процессы в коммуникаторе
- 3 типа операций
  - Синхронизация (MPI_Barrier)
  - Перемещение данных (MPI_Bcast / Scatter(v) / Gather(v) / Allgather(v) / Alltoall(v))
  - Сбор данных и вычисление (MPI_Reduce / Allreduce / Scan / Reduce_scatter)
- Блокирующие операции
- Не используют теги
- Могут быть использованы только с предопределенными типами MPI
- Очень эффективны

Барьерная синхронизация

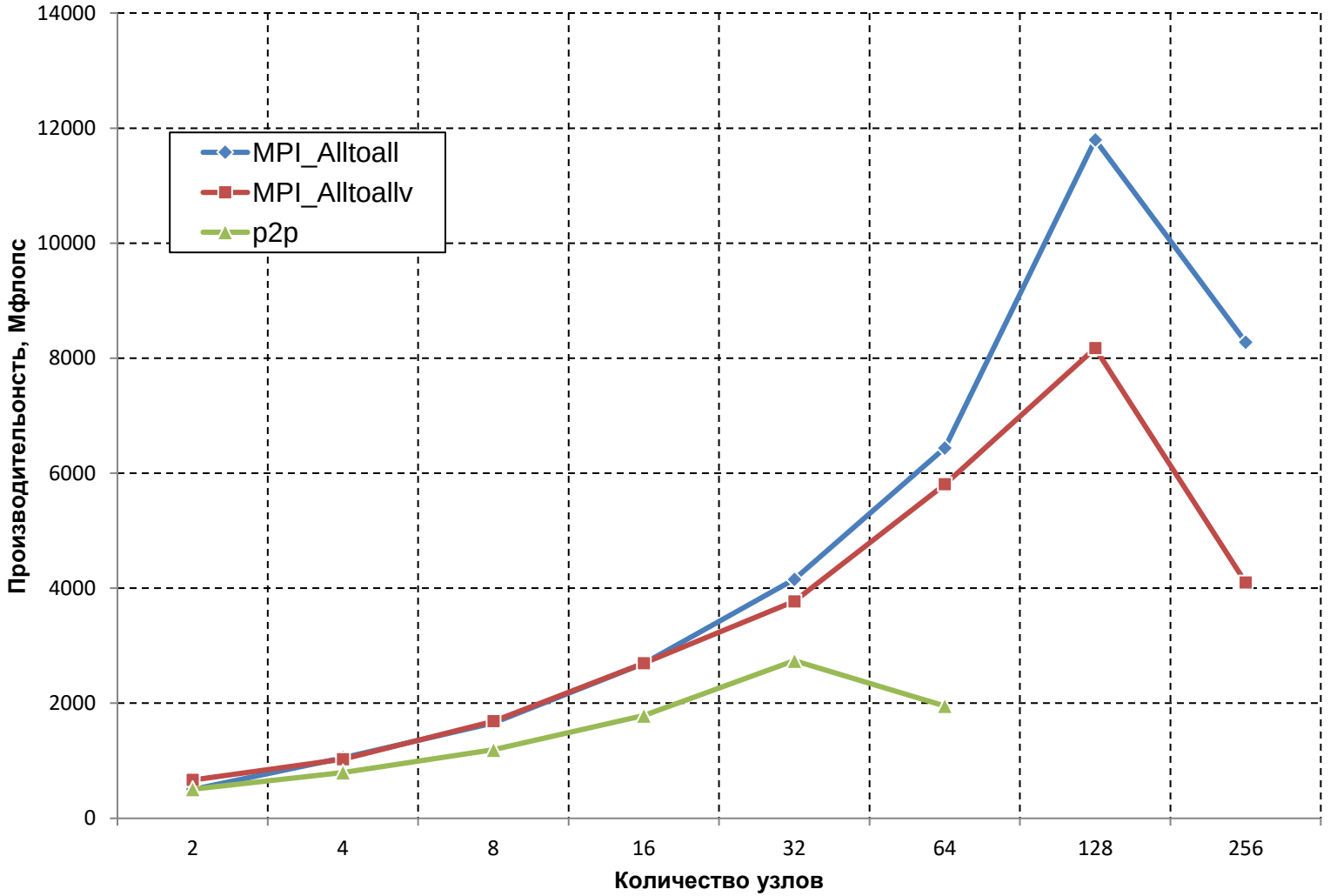
- ***MPI_Barrier (comm, ierr)***
- Синхронизует все процессы в коммуникаторе, каждый процесс ждет, пока все остальные процессы не достигнут барьера
  - Для измерения времени
  - Для разделения фаз вычислений
- Для продолжения корректной работы все процессы должны выйти на барьер



Коллективные операции



Умножение разреженной матрицы на вектор. Сравнение рассылки точка-точка (p2p) с коллективными операциями (MPI_Alltoall и MPI_Alltoallv). Суперкомпьютер «Ломоносов»



Эффективность программ: Избегать использовать коллективные операции с суффиксом -v

Рекомендации

- Объединять мелкие сообщения в большие
- Минимизировать объемы коммуникаций
- Выполнять вычисления и коммуникации одновременно
- Операции «точка-точка»
  - Для длинных сообщений избегать буферизации при отправке сообщения (Recv заранее)
  - Для которых сообщений искать наиболее подходящий вариант использования функций передачи
- Коллективные операции
  - Использовать, если возможно (вместо операций «точка-точка»
  - Избегать использовать операции с суффиксом -v
- Не увлекаться использованием собственных типов данных
- Возможна тонкая настройка при помощи переменных окружения реализации MPI
- Использовать профилирование (Vampir, Scalasca, Jumpshot, …)

Сравнение характеристик доступа к ресурсам, Intel E5-2680 v3, 2.5 ГГц

Параметр	Задержка, нс (такты)	ПС, ГБ/с
Регистр	(1)	–
Кэш L1	1.6 (4)	240
Кэш L2	4.4 (11)	160
Кэш L3	16 (40)	80
Память своего сокета	60	~55
Память чужого сокета	100	~30
Ломоносов, Infiniband 4x QDR	1670	3
Infiniband 4x FDR	1000 (3000)	6.2
Сеть Ангара	MPI – 1000 нс, SHMEM – 600 нс	