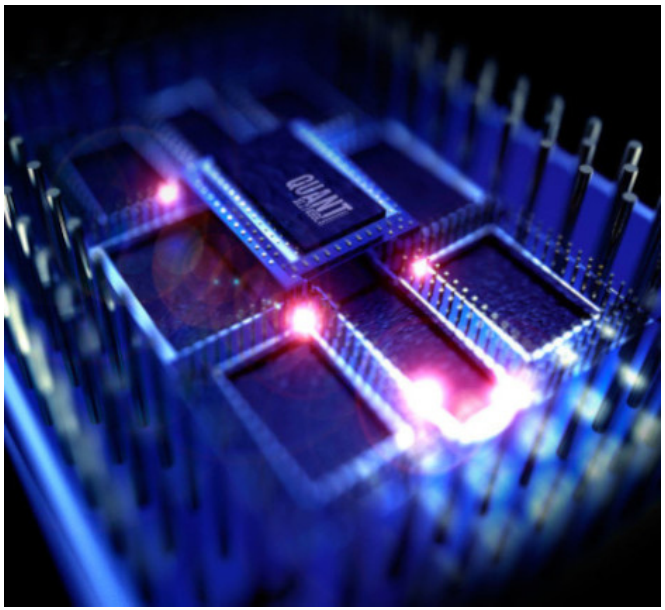
Введение в MPI – 2



П.А. Тараканов

Кафедра астрофизики
математико-механического факультета
Санкт-Петербургского государственного
университета

12 апреля 2022 г.

Асинхронные сообщения

- При передаче сообщений такого типа выполнение процесса продолжается сразу после передачи.
- Теоретически использование таких сообщений увеличивает быстродействие.
- Однако становится больше и источников возможных ошибок.
- Асинхронные сообщения не всегда удачно поддерживаются аппаратурой и программной реализацией MPI.

Передача асинхронного PtP-сообщения

- Для передачи асинхронного сообщения используется процедура `MPI_Isend(Buf, Count, Datatype, Dest, MsgTag, Comm, Request, Err)`.
- Практически все параметры совпадают с аналогичными для блокирующих сообщений.
- `Request` — целое (в C/C++ типа `MPI_Request`), используется для идентификации операции пересылки.
- Как и для сообщений с блокировкой, есть варианты процедуры:
  - `MPI_Ibsend` — передача с буферизацией.
  - `MPI_Issend` — передача с синхронизацией.
  - `MPI_Irsend` — передача при условии, что получатель готов к приему.

Асинхронный прием сообщения

- Для приема сообщения используется процедура `MPI_Irecv(Buf, Count, Datatype, Source, MsgTag, Comm, Request, Err)`.
- Смысл параметров аналогичен блокирующему приему.
- `Request` — идентификатор операции.
- Блокирующие и асинхронные передачи и получения можно смешивать (блокирующая передача может приниматься асинхронно и наоборот).

Информация о принимаемом

- Есть процедура `MPI_Iprobe(Source, Msgtag, Comm, Flag, Status, Err)`, позволяющая «попробовать» принять сообщение.
- Смысл и почти все параметры аналогичны `MPI_Probe`.
- Параметр `Flag` — типа `logical`, он равен `.TRUE.`, если сообщение уже может быть принято, и `.FALSE.` — если еще нет.

Режим ожидания

- Процедура `MPI_Wait(Request, Status, Err)` ожидает завершения асинхронной передачи с идентификатором операции `Request`. Когда передача заканчивается, в `Status` попадают параметры переданного (как в блокирующем случае), а `Request` становится равным `MPI_REQUEST_NULL`.
- Процедура `MPI_Waitall(Count, Requests, Statuses, Err)` ждет сразу `Count` асинхронных операций, заданных массивом `Requests` длины `Count`. Результаты попадают в двумерный массив `Statuses` (вторая размерность должна равняться `Count`).
- Процедура `MPI_Waitany(Count, Requests, Index, Status, Err)` ждет завершения одной из асинхронных операций. `Index` — номер в массиве `Requests` операции, которая завершилась, `Status` — ее (и только ее) параметры.

Тестирование завершения асинхронных передач

- Процедура `MPI_Test(Request, Flag, Status, Err)` проверяет завершенность асинхронной операции с идентификатором `Request`. В `Flag` попадает `.TRUE.`, если операция завершилась, остальные параметры стандартны.
- Есть процедуры `MPI_Testall(Count, Requests, Flag, Statuses, Err)` и `MPI_Testany(Count, Requests, Index, Flag, Status, Err)`. Смысл аналогичен процедурам ожидания.

Синхронизация процессов

- Единственная возможность синхронизации — барьеры.
- Процедура `MPI_Barrier(Comm, Err)`. Останавливает все процессы коммуниктора `Comm`, пока не завершится последний.

Массовая рассылка данных

- Процедура `MPI_Bcast(Buf, Count, Datatype, Root, Comm, Err)` рассылает данные всем процессам коммуникатора `Comm`. Все параметры обычные, `Root` — номер рассылающего процесса в этом коммуникаторе.
- Вызывать ее (и последующие процедуры) должны все потоки коммуникатора, отдельный прием данных какой-либо дополнительной процедурой не требуется.
- Процедура служит барьером для выполнения всех потоков.

Сборка и распределение данных

- Процедура
  `MPI_Gather(SBuf, SCount, SType, RBuf, RCount, RType, Root, Comm, Err)`
  собирает `SCount` элементов данных типа `SType` из массивов `SBuf` всех процессов в `Comm` в один общий буфер `RBuf` процесса `Root`. Порядок сборки происходит по порядку номеров процессов в коммуникаторе, `RCount` и `RType` задают количество и тип элементов, принимаемых от одного конкретного процесса (а не от всех сразу).
- Процедура
  `MPI_Scatter(SBuf, SCount, SType, RBuf, RCount, RType, Root, Comm, Err)`
  обратна предыдущей, она раздает `SCount` элементов данных типа `SType` из массива `SBuf` процесса `Root` коммуникатора `Comm` по массивам `RBuf`.

Глобальные операции

- Процедура
  `MPI_Reduce(SBuf, RBuf, Count, Datatype, Operation, Root, Comm, Err)`
  выполняет `Count` операций `Operation` (независимых) над элементами массивов `SBuf` всех процессов коммуникатора. Результаты складываются в соответствующие элементы массива `RBuf` в процессе `Root`.
- Операции бывают такими: `MPI_MAX`, `MPI_MIN`, `MPI_MINLOC`, `MPI_MAXLOC`, `MPI_SUM`, `MPI_PROD`, `MPI_LAND`, `MPI_LOR`, `MPI_LXOR`, `MPI_BAND`, `MPI_BOR`, `MPI_BXOR` (`L` — для логических данных, `B` — побитовые).
- Существует возможность определить нестандартную операцию.

Большая зачетная задача

- В разных областях физики используется уравнение Пуассона, связывающее между собой, например, гравитационный потенциал φ и распределение плотности ρ, которая этот потенциал создает:

$$\Delta\varphi = -4\,\pi\,G\,\rho. \tag{1}$$

- Решим соответствующую *двумерную* задачу, выбрав единицы так, что $4\pi G = 1$:

$$\frac{\partial^2\varphi}{\partial x^2} + \frac{\partial^2\varphi}{\partial y^2} = -\rho. \tag{2}$$

- Аппроксимируя данные на квадратной сетке $N \times N$ с шагом h, получаем

$$\frac{\varphi_{i+1,j} - 2\cdot\varphi_{i,j} + \varphi_{i-1,j}}{h^2} + \frac{\varphi_{i,j+1} - 2\cdot\varphi_{i,j} + \varphi_{i,j-1}}{h^2} = -\rho_{i,j} \tag{3}$$

Большая зачетная задача

- Тогда отсюда можно выразить $\varphi_{i,j}$ и получить простой итерационный процесс

$$\varphi_{i,j}^{(k+1)} = \frac{1}{4}\left(\varphi_{i+1,j}^{(k)} + \varphi_{i-1,j}^{(k)} + \varphi_{i,j+1}^{(k)} + \varphi_{i,j-1}^{(k)} + \rho_{i,j}\,h^2\right). \qquad (4)$$

- **Задача:** решить уравнение Пуассона итерациями с использованием MPI, разбив расчетную область на четыре равных квадрата и «выдав» каждый квадрат отдельному процессу.
- Граничные условия: $\varphi_{i,j} = 0$ при $i = 1$ или $i = N$ или $j = 1$ или $j = N$. Распределение плотности (точнее, величин $\rho_{i,j}\,h^2$) может быть произвольным.

Большая зачетная задача

- Исходные данные (матрица $\rho_{i,j}\, h^2$) содержится в файле `data.dat` в формате:
  - В первой строке — после символа ♯ и пробела размер матрицы N (одно натуральное число).
  - В последующих N строках — $\rho_{i,j}\, h^2$. В каждой строке файла содержатся элементы одной строки матрицы, отделенные друг от друга одним или несколькими пробелами.
  - Можно предполагать, что заданное N заведомо является четным.
- Результаты должны выводиться в виде файлов в подкаталог `DATA` через каждые q шагов (параметр задать самостоятельно) в виде файлов того же формата с названиями `data001.dat`, `data002.dat` и т.д.
- Следует предусмотреть остановку программы после выполнения какого-то фиксированного числа шагов.
- Важно: данные не должны быть общими; процессы, проводящие вычисления для каждой из четырех областей, передают друг другу информацию только о пограничных значениях $\varphi_{i,j}$.

СПАСИБО ЗА ВНИМАНИЕ!