Введение в MPI

НОЦ МФТИ 2013

Основы параллельного программирования

- Параллельное программирование
- Основные архитектуры параллельных вычислительных систем
- Модели параллельного программирования
- Декомпозиция

Параллельное программирование

- Уменьшение времени работы программы
- Позволяет решать очень большие задачи
- Одновременное решение нескольких задач
- Использование удаленных распределенных ресурсов
- Ограничения на создание последовательных вычислительных систем (с точки зрения физики и экономики)

Параллельные архитектуры

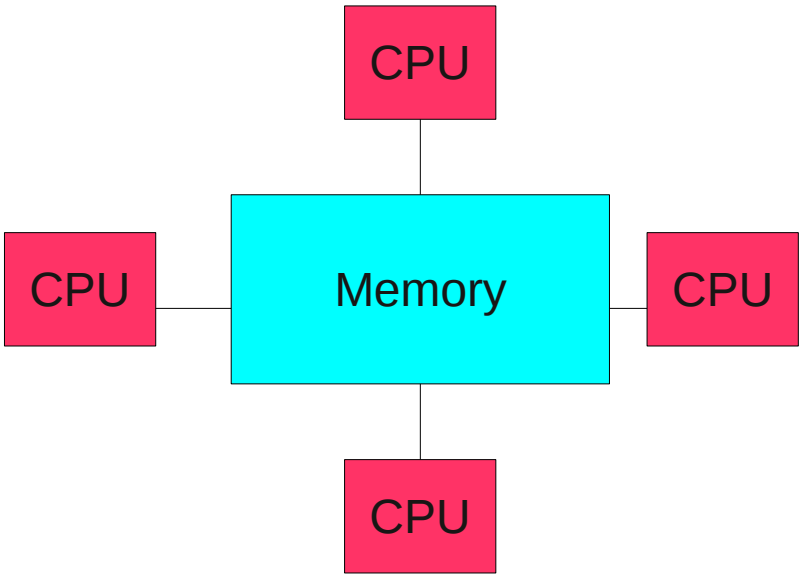
- Архитектуры с общей памятью (shared)
- Архитектуры с распределенной памятью (distributed)
- Гибридные архитектуры (hybrid)

Общая память

- Основная черта — все процессоры имеют доступ в глобальное адресное пространство памяти
- Несколько процессоров могут работать независимо, но использовать одну и туже память
- Любые изменения памяти одним процессором сразу видны другим
- Архитектуру можно разделить на две по типу доступа к памяти: UMA и NUMA

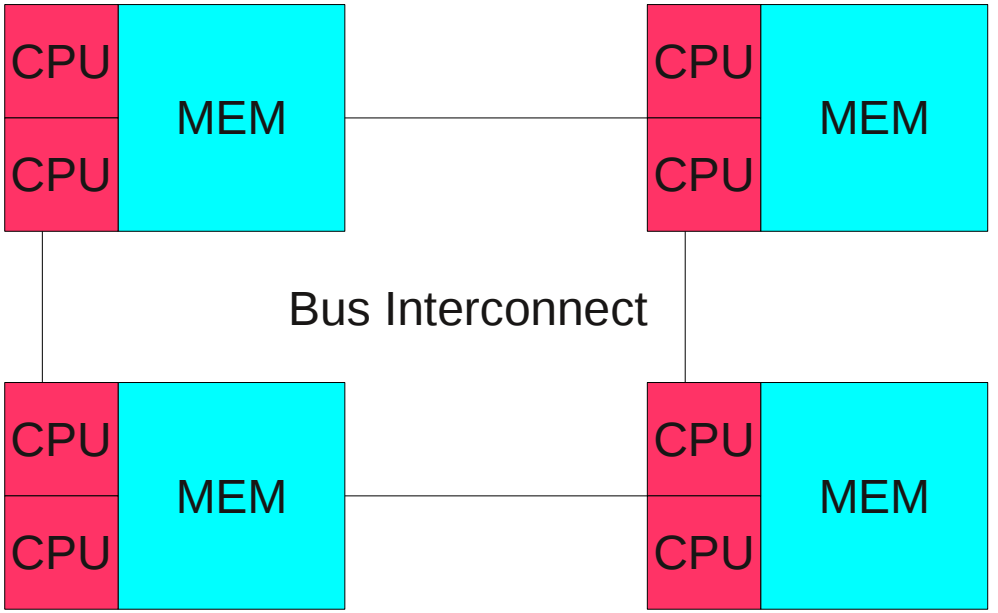
Uniform Memory Access (UMA)

- Большинство современных машин с симметричными мультипроцессорами (SMP)
- Одинаковые процессоры подключены к одной памяти
- Доступ памяти занимает одно и тоже время



Non Uniform Memory Access (NUMA)

- Обычно представляет физическую связку нескольких SMP
- Любой SMP имеет доступ к памяти другого SMP
- Доступ к различным частям памяти различен



Общая память

Преимущества:

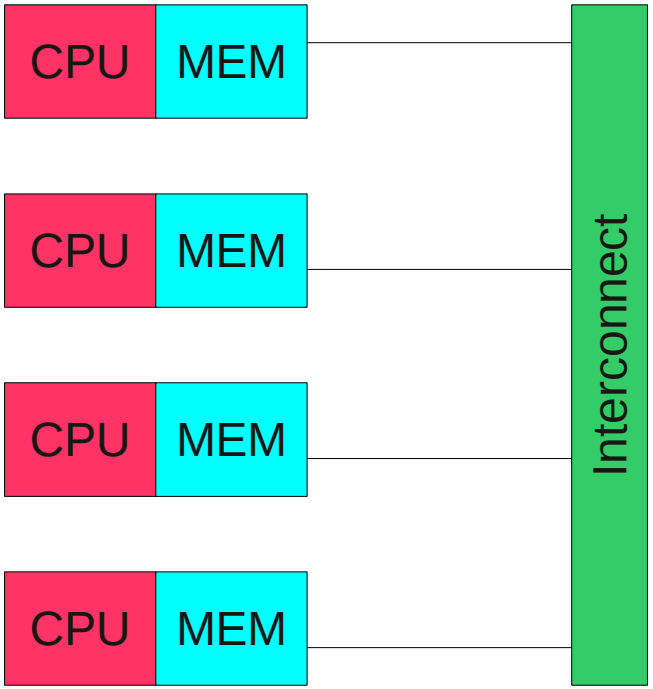
- Простота разработки
- Доступ к данным быстрый и идентичный для всех процессоров

Недостатки:

- Трудно расширяемая архитектура
- Программисту требуется брать на себя заботу о синхронизации работы с памятью
- Очень дорогие машины

Распределенная память

- Наличие интерконнекта для доступа к памяти
- У каждого процессора свое адресное пространство
- Действие одного процесса на память никак не видны другим
- Для доступа к данным другого процессора от программиста требует явно указать как и куда будут доставлены данные



Распределенная память

Достоинства:

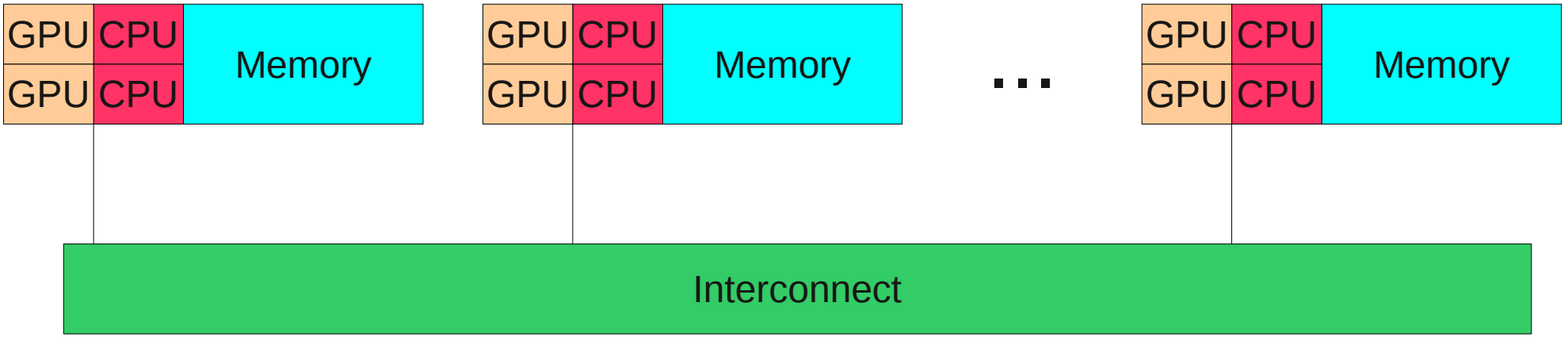
- Расширяемость
- Каждый процесс имеет быстрый доступ к своей памяти
- Более дешевые решения

Недостатки:

- Программист ответственен за коммуникацию между процессорами
- Сложность переноса существующих алгоритмов

Гибридные архитектуры

- Самые мощные современные суперкомпьютеры
- Shared memory компоненты могут быть как SMP CPU так и GPU
- Наиболее перспективная в ближайшем будущем архитектура



Модели параллельного программирования

- Потоки
- Передача сообщений
- Параллелизм по данным
- Гибридные
- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD)

Потоки

- Общее адресное пространство
- Один процесс может иметь несколько нитей исполнения - потоков
- Каждый поток также имеет внутреннюю память
- Общение через глобальную память
- Множество механизмов синхронизации
- Основные реализации — Posix и OpenMP

Передача сообщений

- Набор процессов со своими адресными пространствами
- Обмен данными путем приема и передачи сообщений
- Передача данных обычно требует совместных действий от нескольких процессов (прием-передача, коллективная рассылка)
- Основные реализации - MPI

Параллелизм по данным

- Основное ускорение касается работы над определенным блоком данных
- Несколько процессов работают над одной и той-же структурой данных, однако над ее разными частями
- Все процессы делают одну и туже операцию
- Реализации — High Performance Fortran, различные компиляторы

Гибридные

- Комбинирует несколько выше описанных моделей
- Наиболее часто бывает в виде связки OpenMP + MPI
- Позволяет комбинировать различные архитектуры GPU + MPI

Single Program Multiple Data (SPMD)

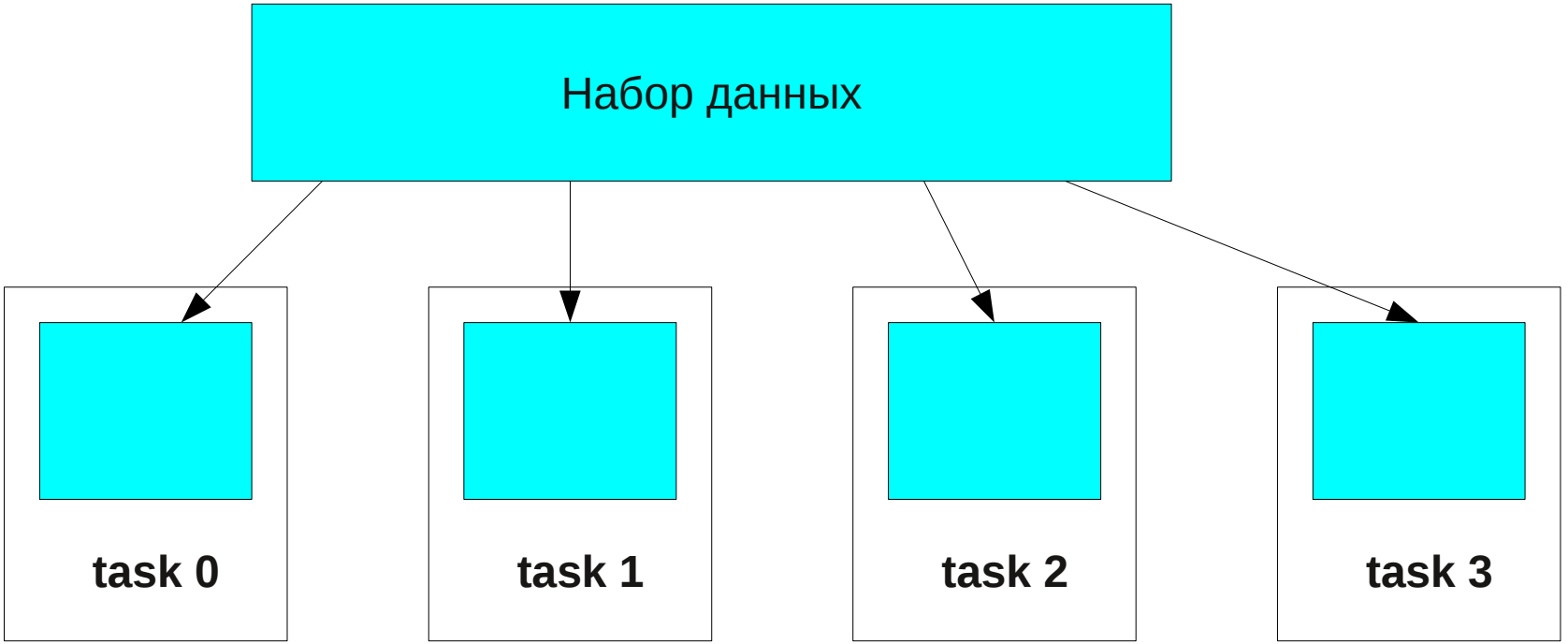
- Высокоуровневая модель, может быть построена на вышеописанных моделях
- Single Program: одна и та же программа работает над своей копией данных
- Multiple Data: все программы могут использовать разные данные
- Наиболее часто используется в распределенной памяти

Multiple Program Multiple Data (MPMD)

- Высокоуровневая модель, может быть построена на вышеописанных моделях
- Multiple Program: запускается несколько программ одновременно
- Multiple Data: программы могут использовать различные наборы данных
- Не так распространена как SPMD, в основном используется при функциональной декомпозиции

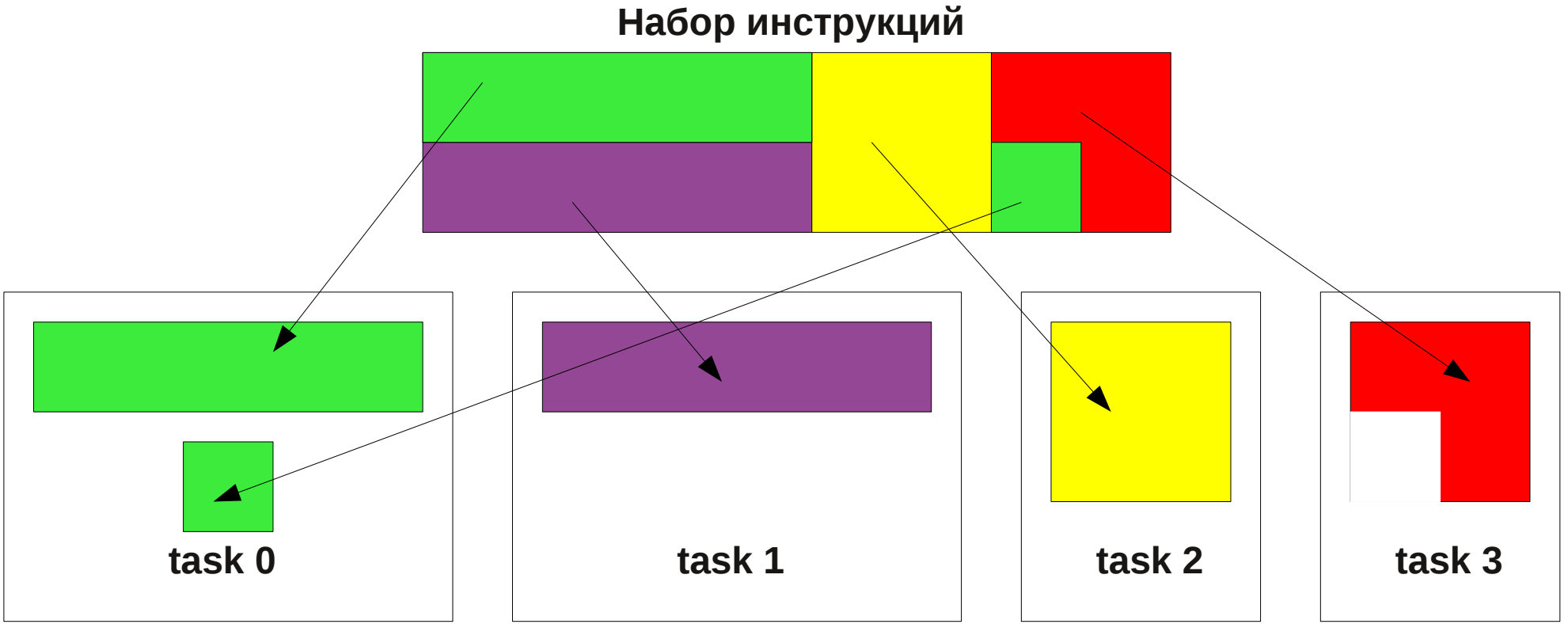
Domain Decomposition

Данные разделаются и каждая параллельная программа обрабатывает свою часть данных



Functional Decomposition

Задача разбивается на подзадачи и каждая программа делает свою подзадачу



Что такое MPI?

- MPI = Message Passing Interface — Интерфейс для передачи сообщений
- MPI — не библиотека, а спецификация для программистов и пользователей. На основе спецификации может быть написана библиотека.
- Основная цель MPI — предоставить широко используемый стандарт для написания приложений с обменом данных
- Интерфейс есть для языков C/C++ и Fortran

История

- Зародился из множества технологий в 1992-1994 годах
- 1980-1990 — появление суперкомпьютеров с разделяемой памятью
- Апрель 1992 — начата работа над спецификацией MPI, были обсуждены основные идеи и функциональность, дальше шла работа над спецификацией (Center for Research on Parallel Computing, Williamsburg, Virginia)

Почему надо использовать MPI?

- **Стандарт** — единственный стандарт HPC
- **Переносимость кода** — нет необходимости менять код при использовании разных платформ
- **Производительность** — производители сами заботятся о скорости работы библиотек
- **Функциональность** — только стандарт MPI-1 предоставляет более 115 функций
- **Доступность** — множество свободных реализаций

Модель программирования

- Дает виртуальный интерфейс ко всем моделям программирования с распределенной памятью
- Железо:
  - Компьютеры с распределенной памятью — изначально разрабатывалась для них
  - Общая память — дает виртуальную распределенную память
  - Гибридные — современные версии MPI дают большие возможности для работы на гибридных архитектурах
- Явный параллелизм

Общая структура MPI программ

Подключение библиотек MPI

Описание типов, функций и т.д.

Последовательный код

Начало программы

Инициализация MPI библиотек

Начало параллельного кода

...

Вычисления и обмены данными

...

Завершение работы MPI

Конец параллельного кода

Последовательный код

Завершение работы программы

Коммуникаторы и группы

- Специальные обьекты — группы и коммуникаторы, используются для определения какие процессы могут взаимодействовать друг с другом
- Большинство функций MPI требуют в качестве параметра коммуникатор
- MPI_COMM_WORLD — глобальный коммуникатор, включающий в себя все процессы

Нумерация процессов

- Внутри коммуникатора каждый процесс имеет уникальный номер (rank, task id)
- Целое число, номера последовательно идут от 0
- Используются в качестве адресатов при передаче сообщений
- Для контроля выполнения программы на разных процессах (if (rank == 0))

Пример

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int  numtasks, rank, rc;

    rc = MPI_Init(&argc,&argv);
    If (rc != MPI_SUCCESS) {
          printf ("Error starting MPI program. Terminating.\n");
          MPI_Abort(MPI_COMM_WORLD, rc);
    }

    MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    printf ("Number of tasks= %d My rank= %d\n",
numtasks,rank);

    /******* вычислительный код  *******/

    MPI_Finalize();
}
```

Компиляция

- Предоставляет свою обертку для стандартного компилятора в системе
- Названия компиляторов в среде Linux: mpicc, mpiCC, mpicxx, mpic++
- По сути вызывает обычный компилятор (gcc, icc и т. д.) с флагами
- Для работы библиотеки необходимо подключить заголовочный файл mpi.h

Запуск

- Для инициализации окружения MPI необходим запуск через ее программы
- Например: mpirun -np 2 ./hello — запустить программу в 2 потока
- Сильно зависит от версии библиотеки
- Для запуска на нескольких узлах использует rsh/ssh протокол

Вопросы.