# Семинар 8
# Стандарт MPI (часть 1)

Михаил Курносов

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2015

# Вычислительные системы с распределенной памятью


- **Вычислительная система с распределенной памятью** (distributed memory computer system) – совокупность вычислительных узлов, взаимодействие между которыми осуществляется через коммуникационную сеть (InfiniBand, Gigabit Ethernet, Cray Gemeni, Fujitsu Tofu, …)
- Каждый узел имеет множество процессоров/ядер, взаимодействующих через разделяемую память (shared memory)
- Процессоры могут быть многоядерными, ядра могут поддерживать одновременную многопоточность (SMT, Hyper-Threading)

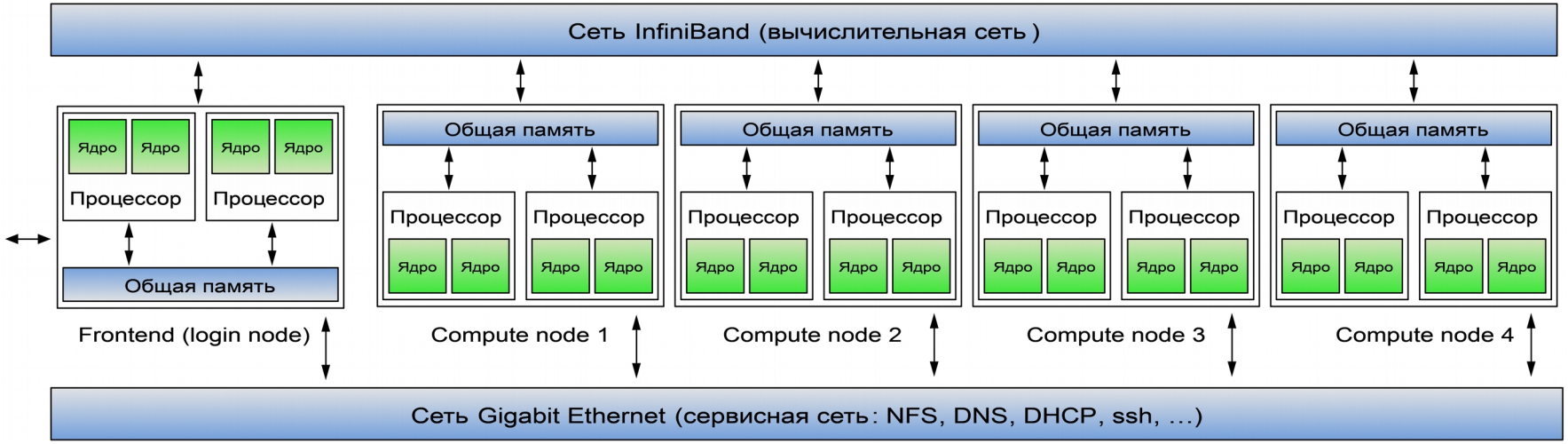
- http://www.top500.org
- http://www.green500.org
- http://www.graph500.org
- http://top50.supercomputers.ru

# Вычислительные системы с распределенной памятью


- **Вычислительная система с распределенной памятью** (distributed memory computer system) – совокупность вычислительных узлов, взаимодействие между которыми осуществляется через коммуникационную сеть (InfiniBand, Gigabit Ethernet, Cray Gemeni, Fujitsu Tofu, …)
- Каждый узел имеет множество процессоров/ядер, взаимодействующих через разделяемую память (shared memory)
- Процессоры могут быть многоядерными, ядра могут поддерживать одновременную многопоточность (SMT, Hyper-Threading)

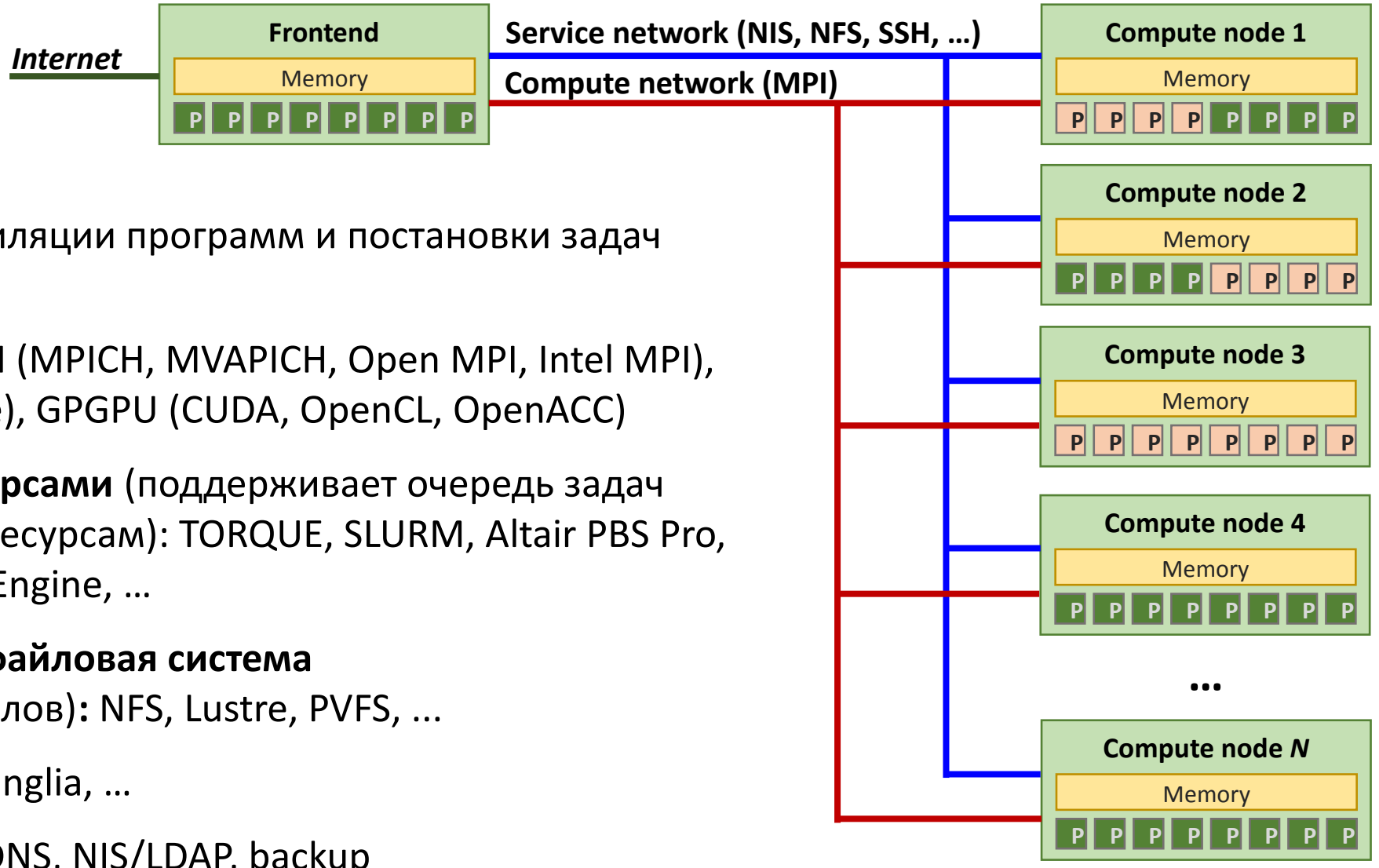
- http://www.top500.org
- http://www.green500.org
- http://www.graph500.org
- http://top50.supercomputers.ru

# Вычислительные кластеры (computer cluster)




- **Вычислительные кластеры строятся на базе <u>свободно доступных для приобретения компонентов</u>**
- Вычислительные узлы: 2/4-процессорные узлы, 1 – 8 GiB оперативной памяти на ядро (поток)
- Коммуникационная сеть (сервисная и для обмена сообщениями)
- Подсистема хранения данных (дисковый массивы, параллельные и сетевые файловые системы)
- Система бесперебойного электропитания
- Система охлаждения
- Программное обеспечение: GNU/Linux (NFS, NIS, DNS, …), MPI (MPICH2, Open MPI), TORQUE/SLURM

# Программное обеспечение вычислительных кластеров




- **Frontend** – узел для компиляции программ и постановки задач в очередь
- **Средства разработки:** MPI (MPICH, MVAPICH, Open MPI, Intel MPI), OpenMP (GCC, Intel, Oracle), GPGPU (CUDA, OpenCL, OpenACC)
- **Система управления ресурсами** (поддерживает очередь задач и контролирует доступ к ресурсам): TORQUE, SLURM, Altair PBS Pro, IBM Load Leveler, Sun GridEngine, …
- **Сетевая (параллельная) файловая система** (доступ к /home со всех узлов)**:** NFS, Lustre, PVFS, ...
- **Система мониторинга:** Ganglia, …
- **Сетевые сервисы**: DHCP, DNS, NIS/LDAP, backup

# Рейтинги мощнейших ВС

- www.top500.org – решение системы линейных алгебраических уравнений методом LU-факторизации (High-Performance Linpack, FLOPS – Floating-point Operations Per Seconds)
- www.graph500.org – алгоритмы на графах (построение графа, обход в ширину, TEPS – Traversed Edges Per Second)
- www.green500.org – главный критерий – энергоэффективность (объем потребляемой эленктроэнергии, kW)
- http://top50.supercomputers.ru – рейтинг мощнейших вычислительных систем СНГ (тест High-Performance Linpack)
- **Как создать свой тест производительности?**

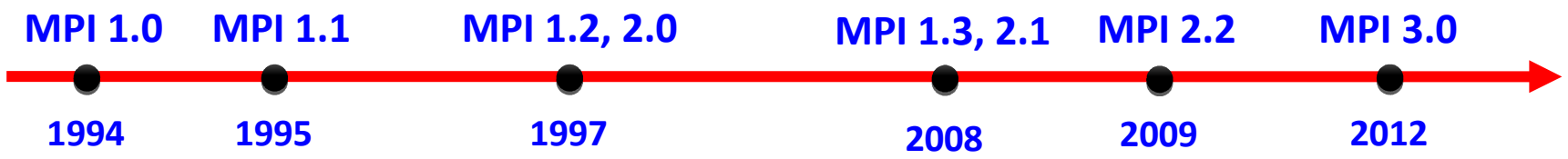
# www.top500.org (Ноябрь 2014)

| | NAME | SPECS | SITE | COUNTRY | CORES | RMAX PFLOP/S | POWER MW |
|---|---|---|---|---|---|---|---|
| 1 | Tianhe-2 (Milkyway-2) | NUDT, Intel Ivy Bridge (12C, 2.2 GHz) & Xeon Phi (57C, 1.1 GHz), Custom interconnect | NSCC Guangzhou | China | 3,120,000 | 33.9 | 17.8 |
| 2 | Titan | Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect | DOE/SC/ORNL | USA | 560,640 | 17.6 | 8.2 |
| 3 | Sequoia | IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect | DOE/NNSA/LLNL | USA | 1,572,864 | 17.2 | 7.9 |
| 4 | K computer | Fujitsu SPARC64 VIIIfx (8C, 2.0GHz), Custom interconnect | RIKEN AICS | Japan | 705,024 | 10.5 | 12.7 |
| 5 | Mira | IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect | DOE/SC/ANL | USA | 786,432 | 8.59 | 3.95 |

- Среднее количество вычислительных ядер в системе: **46 288**
- Среднее количество ядер на сокет (процессор): **9** (4, 6, 8, 10, 12, 14, 16)
- Среднее энергопотребление: **836** kW
- Коммуникационная сеть: Infiniband (44%), Gigabit Ethernet (25%), 10 Gigabit Ethernet (15%), Custom (10%), Cray (3%)
- Процессоры: Intel (> 80%), IBM Power, AMD Opteron, SPARC64, ShenWei, NEC
- Ускорители (14% систем): Intel Xeon Phi, NVIDIA GPU, ATI/AMD GPU, PESY-SC
- Операционная система: GNU/Linux (96%), IBM AIX (11 шт.), Microsoft (1 шт.)

# Стандарт MPI

- **Message Passing Interface (MPI)** – это стандарт на программный интерфейс коммуникационных библиотек для создания параллельных программ в модели передачи сообщений (message passing)
- Стандарт определяет интерфейс для языков программирования C и Fortran
- Стандарт де-факто для систем с распределенной памятью (“ассемблер” в области параллельных вычислений на системах с распределенной памятью)
- Обеспечивает переносимость программ на уровне исходного кода между разными ВС (Cray, IBM, NEC, Fujitsu, ...)




http://www.mpi-forum.org

http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

# Стандарт MPI

MPI: A Message-Passing Interface Standard
Version 3.0

Message Passing Interface Forum

September 21, 2012

## 3.2 Blocking Send and Receive Operations

### 3.2.1 Blocking Send

The syntax of the blocking send operation is given below.

MPI_SEND(buf, count, datatype, dest, tag, comm)

| | | |
|---|---|---|
| IN | buf | initial address of send buffer (choice) |
| IN | count | number of elements in send buffer (non-negative integer) |
| IN | datatype | datatype of each send buffer element (handle) |
| IN | dest | rank of destination (integer) |
| IN | tag | message tag (integer) |
| IN | comm | communicator (handle) |

```
int MPI_Send(const void* buf, int count, MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)

MPI_Send(buf, count, datatype, dest, tag, comm, ierror) BIND(C)
    TYPE(*), DIMENSION(..), INTENT(IN) ::  buf
    INTEGER, INTENT(IN) ::  count, dest, tag
    TYPE(MPI_Datatype), INTENT(IN) ::  datatype
    TYPE(MPI_Comm), INTENT(IN) ::  comm
```

http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

# Реализации MPI

- **MPICH** (Open source, Argone NL, http://www.mcs.anl.gov/research/projects/mpich2)
- Производные от MPICH2:
  MVAPICH2 (MPICH2 for InfiniBand), IBM MPI, Cray MPI, Intel MPI, HP MPI, Microsoft MPI

- **Open MPI** (Open source, BSD License, http://www.open-mpi.org)
- Производные от Open MPI: Oracle MPI

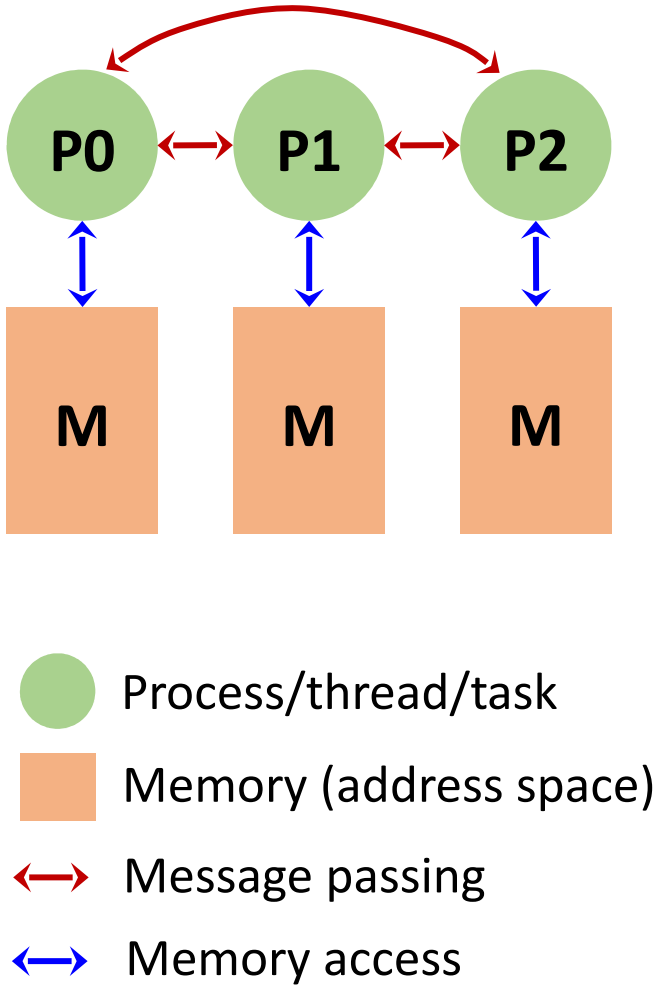
- Высокоуровневые интерфейсы
  - C++: Boost.MPI
  - Java: Open MPI Java Interface, MPI Java, MPJ Express, ParJava
  - C#: MPI.NET, MS-MPI
  - Python: mpi4py, pyMPI

# Отличия реализаций MPI

- **Спектр поддерживаемых архитектур процессоров:**
  Intel, IBM, ARM, Fujitsu, NVIDIA, AMD
- **Типы поддерживаемых коммуникационных технологий/сетей:**
  InfiniBand, 10 Gigabit Ethernet, Cray Gemeni, IBM PERCS/5D torus, Fujitsu Tofu, Myrinet, SCI, ...
- **Протоколы дифференцированных обменов двусторонних обменов (Point-to-point):**
  хранение списка процессов, подтверждение передачи (ACK), буферизация сообщений, ...
- **Коллективные операции обменов информацией:** коммуникационная сложность алгоритмов, учет структуры вычислительной системы (torus, fat tree, ...), неблокирующие коллективные обмены (MPI 3.0, методы хранение collective schedule)
- **Алгоритмы вложения графов программ в структуры вычислительных систем** (MPI topology mapping)
- **Возможность выполнения MPI-функций в многопоточной среде и поддержка ускорителей** (GPU NVIDIA/AMD, Intel Xeon Phi)

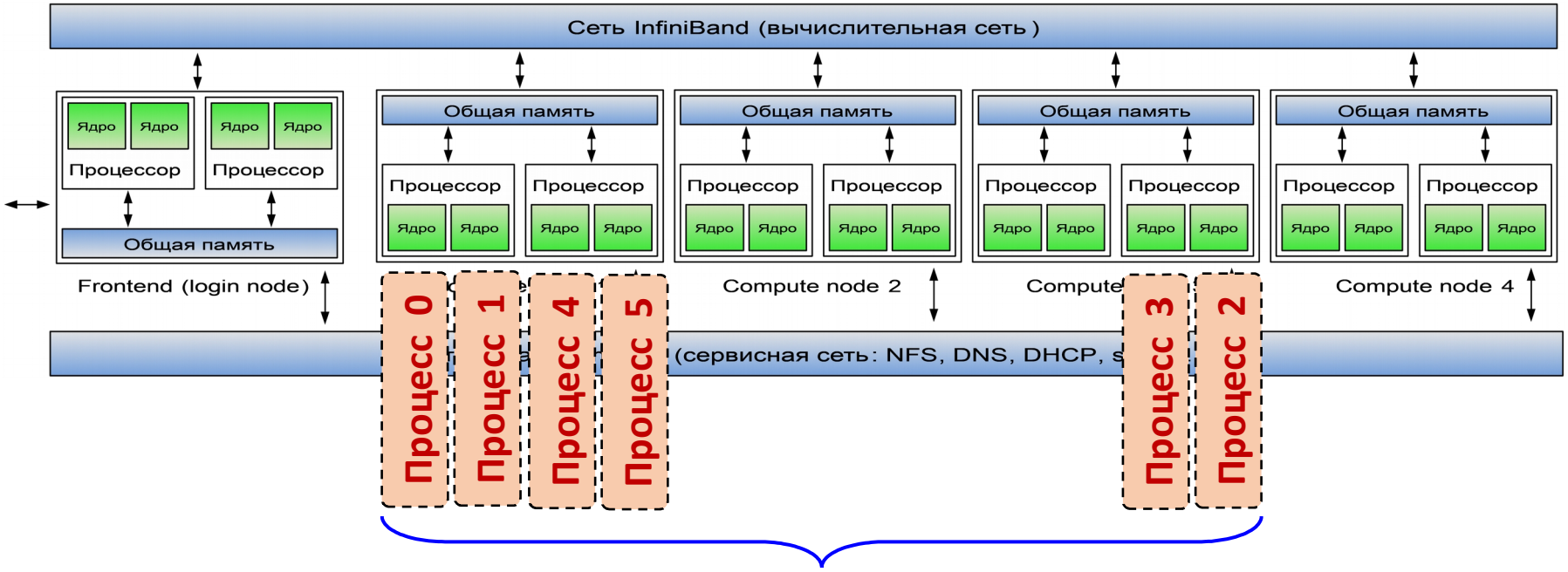
# Модель программирования

- Программа состоит из N параллельных процессов, которые порождаются при запуске программы (MPI 1) или могут быть динамически созданы во время выполнения (MPI 2)
- Каждый процесс имеет уникальный идентификатор [0, N – 1] и изолированное адресное пространство (SPMD)
- Процессы взаимодействуют путем передачи сообщений (message passing)
- Процессы могут образовывать группы для реализации коллективных операций обмена информацией



# Понятие коммуникатора (communicator)

- **Коммуникатор (communicator)** – группа процессов, образующая логическую область для выполнения коллективных операций между процессами
- В рамках коммуникатора процессы имеют номера: 0, 1, …, N – 1
- Все MPI-сообщения должны быть связаны с определенным коммуникатором




Коммуникатор **MPI_COMM_WORLD** включает все процессы

# Программный интерфейс MPI

- Заголовочный файл mpi.h
  **#include <mpi.h>**

- Функции, типы данных и именованные константы имеют префикс «MPI_»
  MPI_Init, MPI_Init_thread, MPI_Send

- Функции возвращают MPI_SUCCESS или код ошибки

- Результаты возвращаются через аргументы функций

```
int MPI_Send(const void *buf,
             int count,
             MPI_Datatype datatype,
             int dest,
             int tag,
             MPI_Comm comm);
```

# Hello, MPI World!

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

int main(int argc, char *argv[])
{
    int commsize, rank, len;
    char procname[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &commsize);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(procname, &len);

    printf("Hello, MPI World! Process %d of %d on node %s.\n",
           rank, commsize, procname);

    MPI_Finalize();
    return 0;
}
```

# Компиляция MPI-программ

```
# Программа на C
$ mpicc -Wall -o hello ./hello.c

# Программа на C++
$ mpicxx -Wall -o hello ./hello.cpp

# Программа на Fortran
$ mpif90 -o hello ./hello.f90
```

# Запуск MPI-программы на кластере (система SLURM)

```
# Формируем паспорт задачи (job-файл)
$ cat task.job
#!/bin/bash

#SBATCH --nodes=2 --ntasks-per-node=8
#SBATCH --job-name=MyTask

mpiexec ./hello

# Ставим задачу в очередь
$ sbatch ./task.job
Submitted batch job 4285
```

**Ресурсный запрос**

nodes=2 — два узла кластера
ntasks-per-node=8 — 8 процессов на узле

man sbatch

# Запуск MPI-программы на кластере (система SLURM)

```
# Проверяем состояние задачи в очереди
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              4285     debug   MyTask mkurnoso  R       0:01      2 cn[3-4]

# Проверяем результат
$ cat ./slurm-4285.out
Hello, MPI World! Process 12 of 16 on node cn4.
Hello, MPI World! Process 13 of 16 on node cn4.
Hello, MPI World! Process 14 of 16 on node cn4.
Hello, MPI World! Process 15 of 16 on node cn4.
Hello, MPI World! Process 8 of 16 on node cn4.
Hello, MPI World! Process 9 of 16 on node cn4.
Hello, MPI World! Process 10 of 16 on node cn4.
Hello, MPI World! Process 11 of 16 on node cn4.
Hello, MPI World! Process 4 of 16 on node cn3.
Hello, MPI World! Process 5 of 16 on node cn3.
Hello, MPI World! Process 6 of 16 on node cn3.
Hello, MPI World! Process 0 of 16 on node cn3.
Hello, MPI World! Process 1 of 16 on node cn3.
Hello, MPI World! Process 2 of 16 on node cn3.
Hello, MPI World! Process 3 of 16 on node cn3.
Hello, MPI World! Process 7 of 16 on node cn3.
```

# Подсчет количества простых чисел (serial verion)

```
const int a = 1;
const int b = 10000000;

int is_prime_number(int n)
{
    int limit = sqrt(n) + 1;
    for (int i = 2; i <= limit; i++) {
        if (n % i == 0)
            return 0;
    }
    return (n > 1) ? 1 : 0;
}

int count_prime_numbers(int a, int b)
{
    int nprimes = 0;

    if (a <= 2) {
        nprimes = 1;      /* Count '2' as a prime number */
        a = 2;
    }
    if (a % 2 == 0)       /* Shift 'a' to odd number */
        a++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    for (int i = a; i <= b; i += 2) {
        if (is_prime_number(i))
            nprimes++;
    }
    return nprimes;
}
```

Определят, является ли число n простым
O(sqrt(n))

Подсчитывает количество простых числе в интервале [a, b]

# Подсчет количества простых чисел (MPI version)

```
const int a = 1;
const int b = 10000000;

int is_prime_number(int n)
{
    int limit = sqrt(n) + 1;
    for (int i = 2; i <= limit; i++) {
        if (n % i == 0)
            return 0;
    }
    return (n > 1) ? 1 : 0;
}

int count_prime_numbers(int a, int b)
{
    int nprimes = 0;

    if (a <= 2) {
        nprimes = 1;      /* Count '2' as a prime number */
        a = 2;
    }
    if (a % 2 == 0)       /* Shift 'a' to odd number */
        a++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    for (int i = a; i <= b; i += 2) {
        if (is_prime_number(i))
            nprimes++;
    }
    return nprimes;
}
```

Распределим итерации цикла
между процессами MPI-программы

# Подсчет количества простых чисел (MPI version)

```
int main(int argc, char **argv)
{
    MPI_Init(&argc, &argv);

    // Start serial version in process 0
    double tserial = 0;
    if (get_comm_rank() == 0)
        tserial = run_serial();

    // Start parallel version
    double tparallel = run_parallel();

    if (get_comm_rank() == 0) {
        printf("Count prime numbers on [%d, %d]\n", a, b);
        printf("Execution time (serial): %.6f\n", tserial);
        printf("Execution time (parallel): %.6f\n", tparallel);
        printf("Speedup (processes %d): %.2f\n", get_comm_size(), tserial / tparallel);
    }

    MPI_Finalize();
    return 0;
}
```

# Подсчет количества простых чисел (MPI version)

```
double run_serial()
{
    double t = MPI_Wtime();
    int n = count_prime_numbers(a, b);
    t = MPI_Wtime() - t;

    printf("Result (serial): %d\n", n);
    return t;
}

double run_parallel()
{
    double t = MPI_Wtime();
    int n = count_prime_numbers_par(a, b);
    t = MPI_Wtime() - t;
    printf("Process %d/%d execution time: %.6f\n", get_comm_rank(), get_comm_size(), t);

    if (get_comm_rank() == 0)
        printf("Result (parallel): %d\n", n);
    return t;
}
```

# Подсчет количества простых чисел (MPI version)

```
void get_chunk(int a, int b, int commsize, int rank, int *lb, int *ub)
{
    /*
     * This algorithm is based on OpenMP 4.0 spec (Sec. 2.7.1, default schedule for loops)
     * For a team of commsize processes and a sequence of n items, let ceil(n ? commsize) be the integer q
     * that satisfies n = commsize * q - r, with 0 <= r < commsize.
     * Assign q iterations to the first commsize - r processes, and q - 1 iterations to the remaining r processes.
     */
    int n = b - a + 1;
    int q = n / commsize;
    if (n % commsize)
        q++;
    int r = commsize * q - n;

    int chunk = q;                              /* Compute chunk size for the process */
    if (rank >= commsize - r)
        chunk = q - 1;

    *lb = a;                                    /* Determine start item for the process */
    if (rank > 0) {
        if (rank <= commsize – r)               /* Count sum of previous chunks */
            *lb += q * rank;
        else
            *lb += q * (commsize - r) + (q - 1) * (rank - (commsize - r));
    }
    *ub = *lb + chunk - 1;
}
```

# Подсчет количества простых чисел (MPI version)

```
Count prime numbers on [1, 10000000]
Process 0/8 execution time: 0.466901
Process 1/8 execution time: 0.780994
Process 2/8 execution time: 1.126007
Process 3/8 execution time: 1.125990
Process 4/8 execution time: 1.556572
Process 5/8 execution time: 1.358095
Process 6/8 execution time: 1.556561
Process 7/8 execution time: 1.556546
Execution time (serial): 8.970650
Execution time (parallel): 0.466901
Result (parallel): 664579
Result (serial): 664579
Speedup (processes 8): 19.21
```

**Проблема 1**
Неравномерная загрузка процессов
(load imbalance)

**Проблема 2**
Ускорение в 19 раз?

# Подсчет количества простых чисел (MPI version)

```
int count_prime_numbers_par(int a, int b)
{
    int nprimes = 0;

    int lb, ub;
    get_chunk(a, b, get_comm_size(), get_comm_rank(), &lb, &ub);

    /* Count '2' as a prime number */
    if (lb <= 2) {
        nprimes = 1;
        lb = 2;
    }

    /* Shift 'a' to odd number */
    if (lb % 2 == 0)
        lb++;

    /* Loop over odd numbers: a, a + 2, a + 4, ... , b */
    for (int i = lb; i <= ub; i += 2) {
        if (is_prime_number(i))
            nprimes++;
    }

    int nprimes_global;
    MPI_Reduce(&nprimes, &nprimes_global, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    return nprimes_global;
}
```

**Проблема 1**

Неравномерная загрузка процессов

```
Process 0: 0, 1, 2, 3
Process 1: 4, 5, 6, 7
Process 3: 8, 9, 10, 11
...
```

$T_{is\_prime\_number} = O(\mathrm{sqrt}(i))$

# Подсчет количества простых чисел (MPI version)

```
int main(int argc, char **argv)
{
    MPI_Init(&argc, &argv);

    // Start serial version in process 0
    double tserial = 0;
    if (get_comm_rank() == 0)
        tserial = run_serial();

    // Start parallel version
    double tparallel = run_parallel();

    if (get_comm_rank() == 0) {
        printf("Count prime numbers on [%d, %d]\n", a, b);
        printf("Execution time (serial): %.6f\n", tserial);
        printf("Execution time (parallel): %.6f\n", tparallel);
        printf("Speedup (processes %d): %.2f\n", get_comm_size(), tserial / tparallel);
    }

    MPI_Finalize();
    return 0;
}
```

**Проблема 2**

Неправильное вычисление коэффициента ускорения

```
Speedup = tserial / tparallel
```

tparallel – время процесса 0

За tparallel надо брать наибольшее время выполнения процессов

# Подсчет количества простых чисел (MPI version)

**Решение проблемы 2**

$$S = \frac{T_{serial}}{\max(T_0, T_1, ..., T_{N-1})}$$

```
double run_parallel()
{
    double t = MPI_Wtime();
    int n = count_prime_numbers_par(a, b);
    t = MPI_Wtime() - t;
    printf("Process %d/%d execution time: %.6f\n",
           get_comm_rank(), get_comm_size(), t);

    if (get_comm_rank() == 0)
        printf("Result (parallel): %d\n", n);

    // Собираем в процессе 0 максимальное из времен выполнения
    double tmax;
    MPI_Reduce(&t, &tmax, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    return tmax;
}
```

# Подсчет количества простых чисел (MPI version)

```
int count_prime_numbers_par(int a, int b)
{
    int nprimes = 0;

    /* Count '2' as a prime number */
    int commsize = get_comm_size();
    int rank = get_comm_rank();

    if (a <= 2) {
        a = 2;
        if (rank == 0)
            nprimes = 1;
    }

    for (int i = a + rank; i <= b; i += commsize) {
        if (i % 2 > 0 && is_prime_number(i))
            nprimes++;
    }
    int nprimes_global = 0;
    MPI_Reduce(&nprimes, &nprimes_global, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    return nprimes_global;
}
```

**Решение проблемы 1**

Циклическое распределение итераций (round-robin)

# Задание

- Разработать на MPI параллельную версию программы подсчета количества простых числе в заданном интервале — написать код функции count_prime_numbers_par()
- Провести анализ масштабируемости параллельной программы
- Шаблон программы находится в каталоге _task