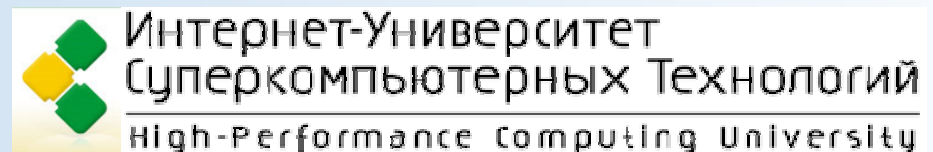


Основы параллельного программирования с использованием MPI

Лекция 1

Немнюгин Сергей Андреевич
Санкт-Петербургский государственный университет
кафедра вычислительной физики

snemnyugin@mail.ru



План лекции

- Обзор курса.
- Модели программирования: последовательная и параллельная.
- Программные инструменты параллельного программирования:
 - инструменты низкого уровня;
 - инструменты высокого уровня.

Обзор курса

Курс *Основы параллельного программирования с использованием MPI* посвящён одному из основных инструментов разработки параллельных программ - *Интерфейсу Обмена Сообщениями* (MPI - Message Passing Interface).

В рамках данного курса мы:

- познакомимся с особенностями параллельной модели программирования;
- познакомимся с терминологией, основными понятиями, концепциями и архитектурой MPI;
- разберём способы организации обменов разного типа: двухточечных, коллективных, блокирующих, неблокирующих, односторонних и т. д.;
- разберём работу с пользовательскими типами, виртуальными топологиями, организацией параллельного ввода-вывода;
- познакомимся с динамическим анализом параллельных MPI-приложений.

Предполагается разбор примеров программ, демонстрация приемов практической работы.

Рекомендуется выполнение домашних заданий!

Литература и другие источники



С.Немнюгин, О.Стесик. Параллельное программирование для многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 2002.



В.В.Воеводин, Вл.В.Воеводин. Параллельные вычисления СПб.: БХВ-Петербург, 2002.



В.П.Гергель. Современные языки и технологии параллельного программирования. Московский государственный университет им. М.В.Ломоносова, 2012.

Портал **PARALLEL.RU**

Лекция 1

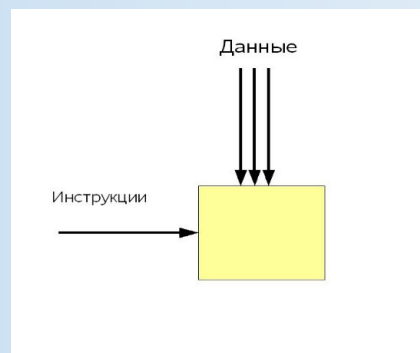
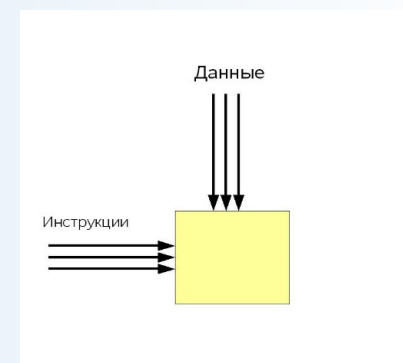
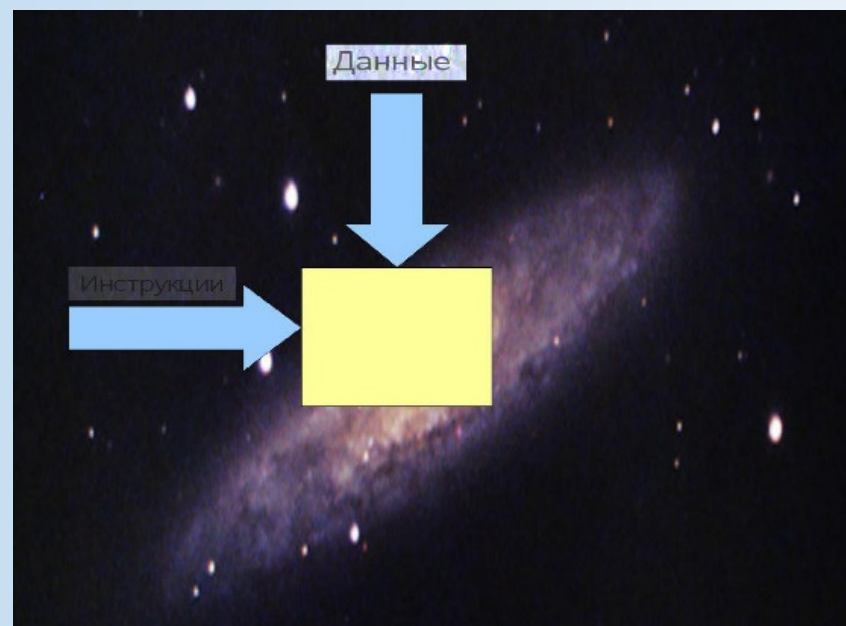
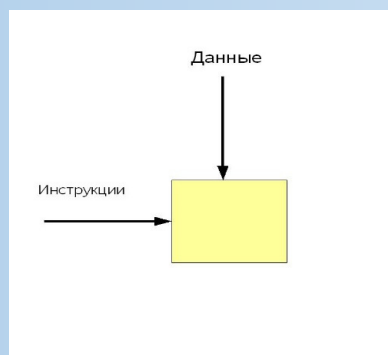
В первой лекции обсуждаются последовательная и параллельная модели программирования.

Рассматриваются различные парадигмы параллельного программирования и их программные реализации (POSIX Threads, Microsoft Windows API, OpenMP, MPI, PVM и т. д.).

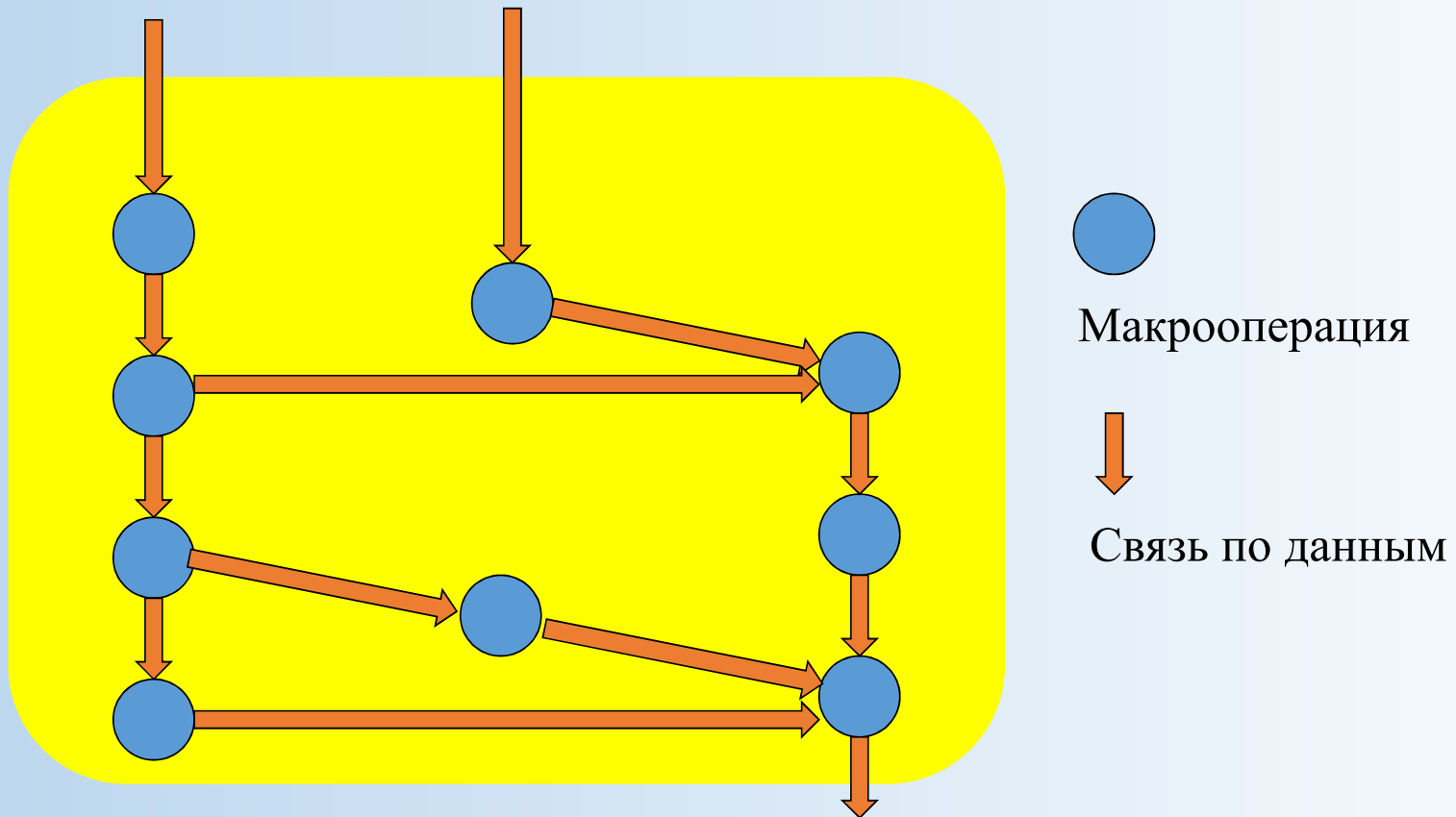


Последовательная и параллельная модели программирования

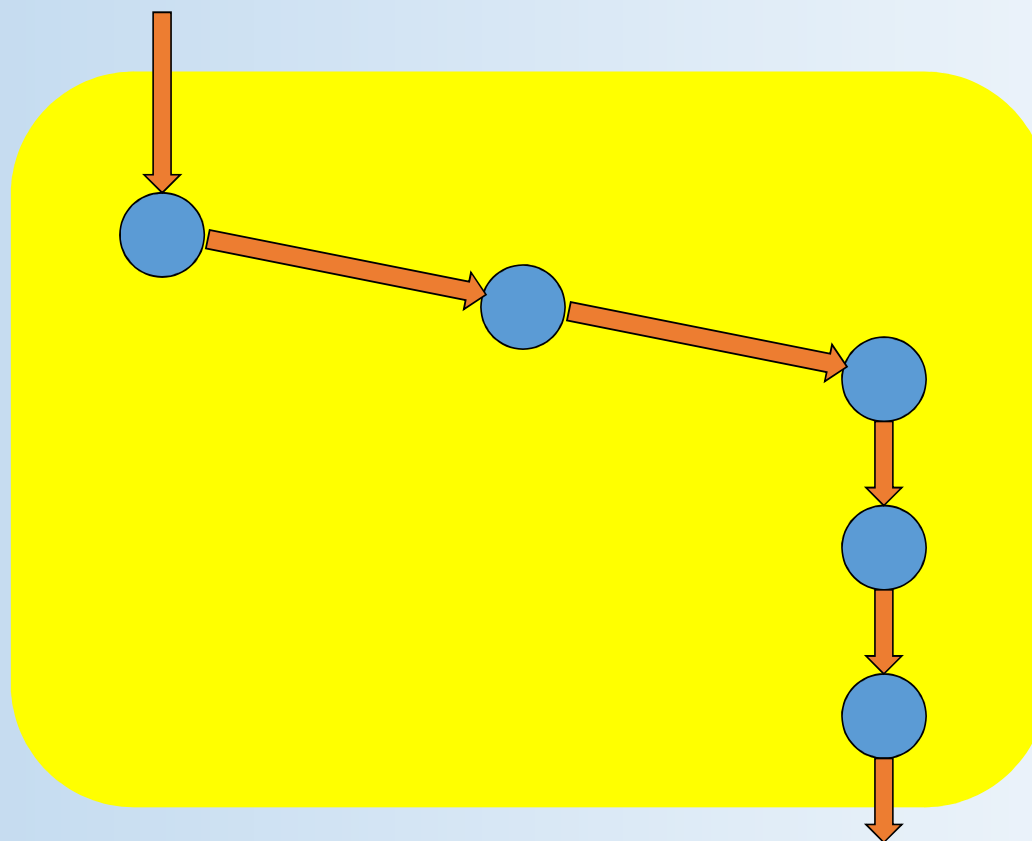
Идея Флинна



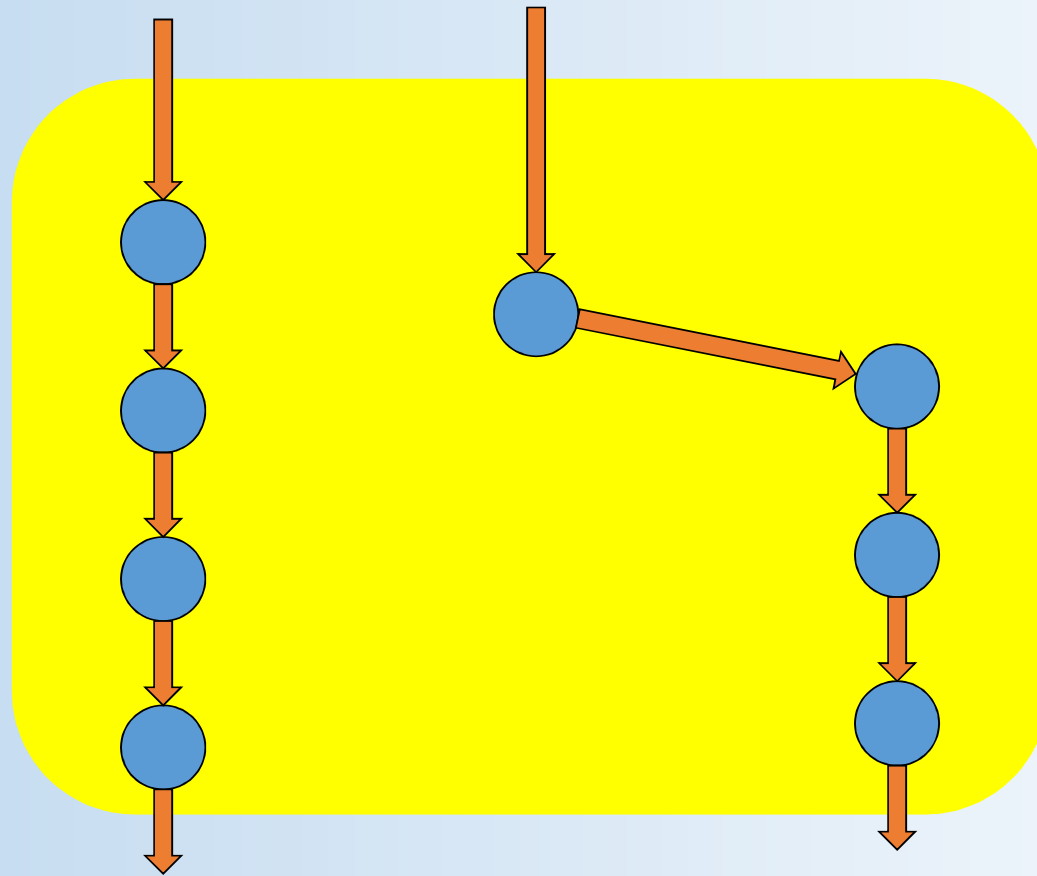
Что происходит с данными внутри программы. Информационный граф



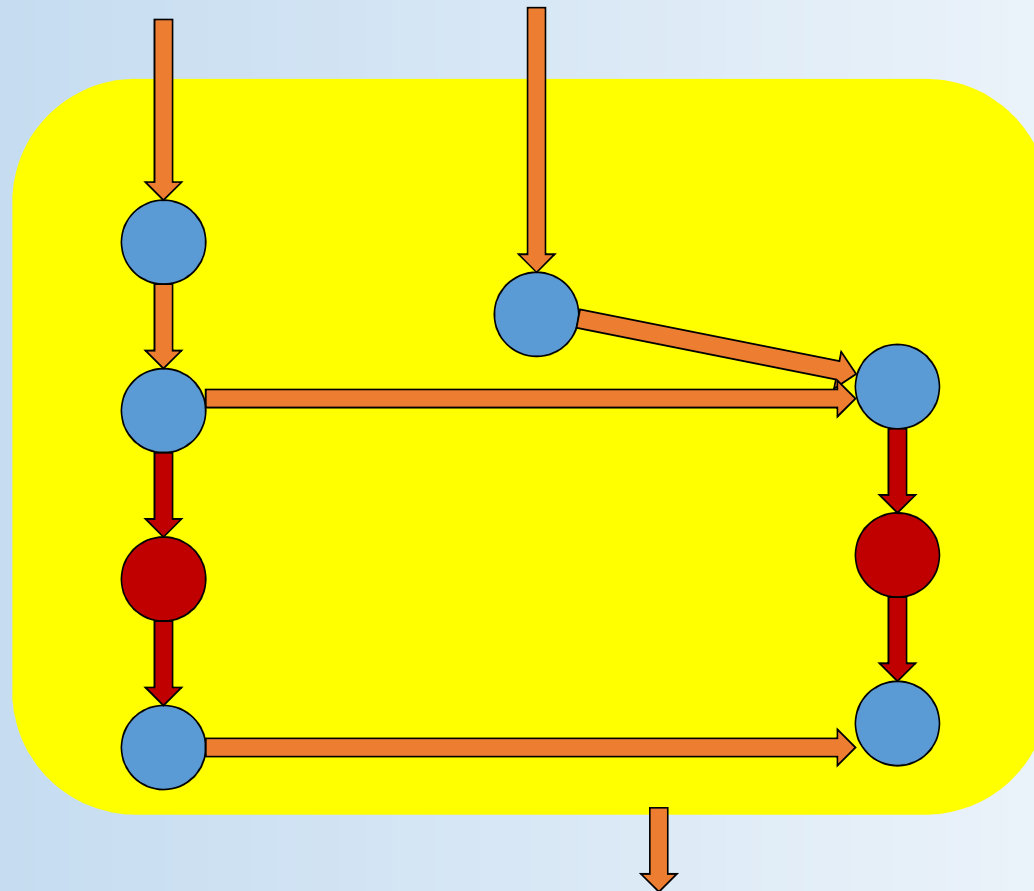
Последовательная программа



Параллельная программа



Общий случай



Последовательный и распараллеливаемый компоненты программы

Последовательная модель программирования

Последовательную модель программирования характеризуют:

- невысокая производительность, ограниченная производительностью единственного процессора ;
- применение стандартных языков программирования;
- хорошая переносимость на уровне исходного кода.

Параллельная модель программирования

Параллельную модель программирования характеризуют:

- возможность добиться высокой производительности;
- применение специальных приемов и (программных) инструментов программирования;
- повышенная трудоемкость программирования;
- возможны проблемы с переносимостью программ.

Параллельные программы должны:

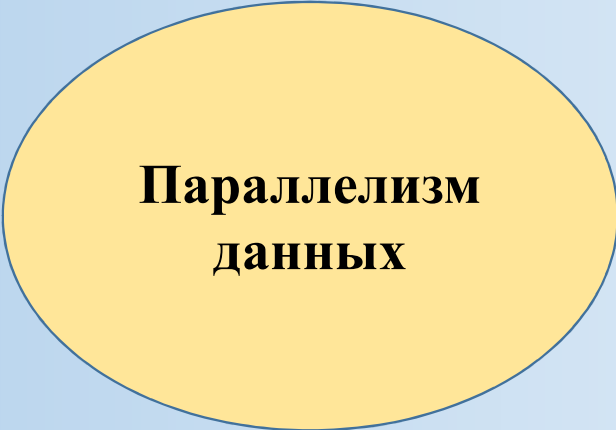
- обладать достаточной степенью параллелизма;
- быть хорошо масштабируемыми;
- обладать детерминированным поведением;
- эффективными.

Почему параллельное программирование более трудоемкое, чем последовательное?

Программист должен заботиться:

- об управлении работой множества процессов;
- об организации межпроцессных пересылок данных;
- о вероятности тупиковых ситуаций (взаимных блокировках);
- о нелокальном и динамическом характере ошибок;
- о возможной утрате детерминизма («гонки за данными»);
- о масштабируемости;
- о сбалансированной загрузке вычислительных узлов.

Две парадигмы параллельного программирования



**Параллелизм
данных**



**Параллелизм
задач**

Параллелизм данных

Подход, основанный на параллелизме данных, характеризуется тем, что:

- Одна операция применяется сразу к нескольким элементам массива данных. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах (ядрах) параллельной вычислительной системы.
- Обработкой данных управляет одна программа.
- Пространство имен является глобальным.
- Параллельные операции над элементами массива выполняются одновременно на всех доступных данной программе процессорах.

От программиста требуется:

- ✓ задание опций векторной или параллельной оптимизации транслятору;
- ✓ задание директив параллельной компиляции;
- ✓ использование специализированных языков параллельных вычислений, а также библиотек подпрограмм, специально разработанных с учетом конкретной архитектуры компьютера и оптимизированных для этой архитектуры.

Параллелизм задач

Параллелизм задач характеризуется тем, что:

- Вычислительная задача разбивается на несколько относительно самостоятельных подзадач. Каждая подзадача выполняется на своем процессоре (ориентация на архитектуру MIMD).
- Для каждой подзадачи пишется своя собственная программа на обычном языке программирования (чаще всего это Fortran или C/C++).
- Подзадачи должны обмениваться результатами своей работы, получать исходные данные. Обмен осуществляется вызовом процедур специализированной библиотеки. Программист может контролировать распределение данных между различными процессорами и различными подзадачами, а также обмен данными.

Достоинства подхода:

- ✓ большая гибкость и большая свобода, предоставляемая программисту в разработке программы, эффективно использующей ресурсы параллельного компьютера;
- ✓ возможность достижения максимального быстродействия.

Разработка параллельной программы/алгоритма

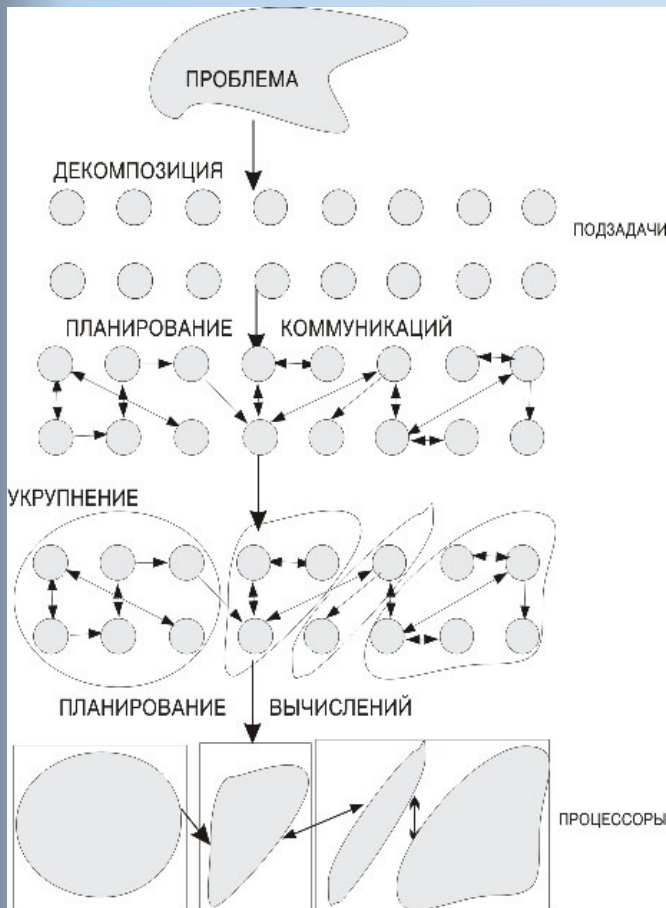
Схема Фостера

Декомпозиция

На этом этапе выполняются анализ, оценка возможности распараллеливания. Задача и связанные с ней данные разделяются на более мелкие части — подзадачи и фрагменты структур данных. Особенности архитектуры конкретной вычислительной системы на данном этапе могут не учитываться.

Требования

- количество подзадач после декомпозиции должно быть достаточно большим;
- следует избегать лишних вычислений и пересылок данных;
- подзадачи должны быть примерно одинакового размера.



Декомпозиция

Независимость

- **независимость по данным** - данные, обрабатываемые одной частью программы, не модифицируются другой ее частью;
- **независимость по управлению** - порядок выполнения частей программы может быть определен только во время выполнения программы (наличие зависимости по управлению предопределяет последовательность выполнения);
- **независимость по ресурсам** - обеспечивается достаточным количеством компьютерных ресурсов (объемом памяти, количеством функциональных узлов и т. д.);
- **зависимость по выводу** - возникает, если две подзадачи производят запись в одну и ту же переменную, а зависимость по вводу-выводу, если операторы ввода/вывода двух или нескольких подзадач обращаются к одному файлу (или переменной).

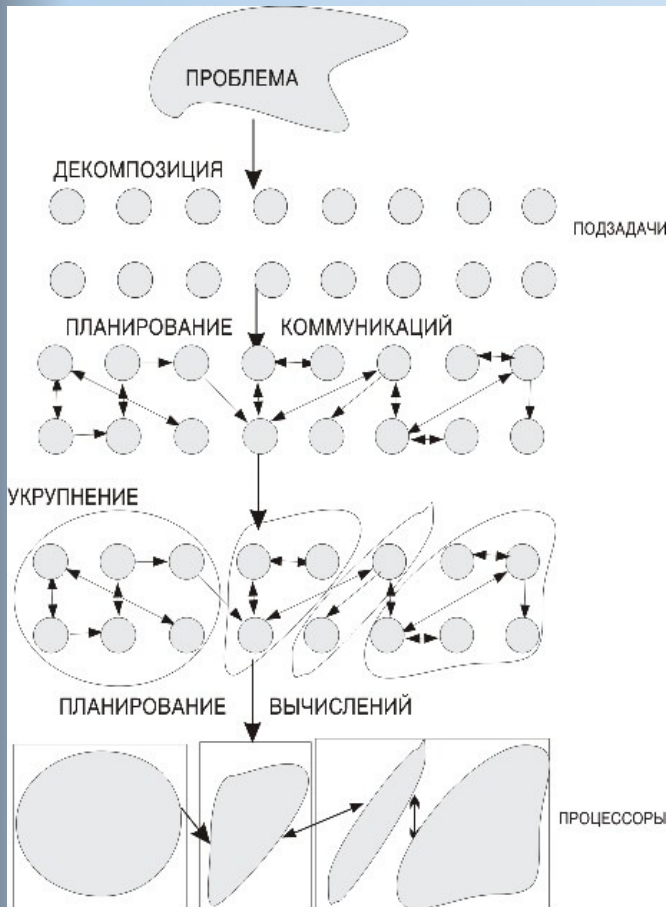
Проектирование коммуникаций

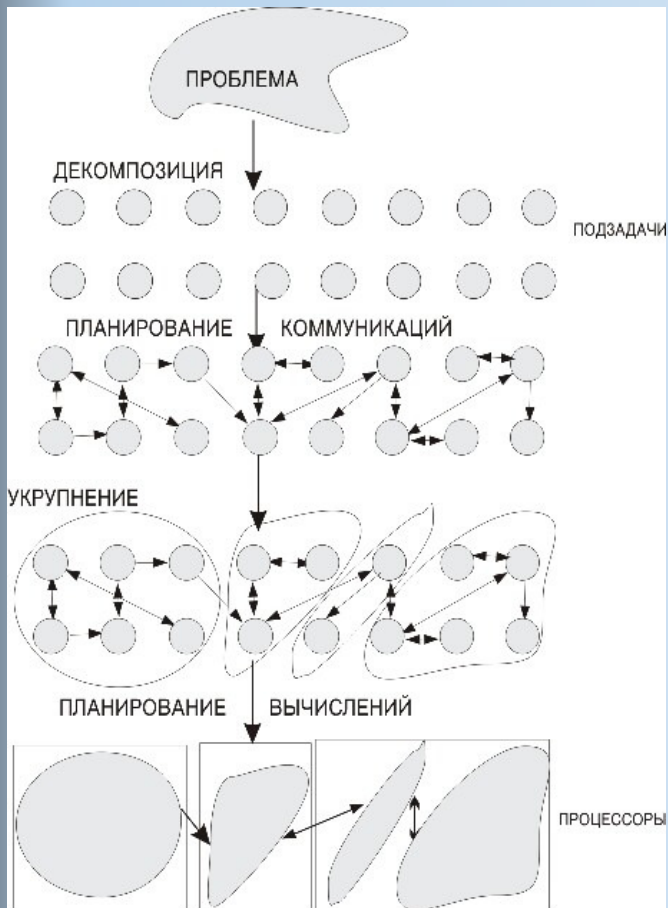
Типы коммуникаций:

- **локальные** - каждая подзадача связана с небольшим набором других подзадач;
- **глобальные** - каждая подзадача связана с большим числом других подзадач;
- **структурированные** - каждая подзадача и подзадачи, связанные с ней, образуют регулярную структуру (например, с топологией решетки);
- **неструктурированные** - подзадачи связаны произвольным графом;
- **синхронные** - отправитель и получатель данных координируют обмен;
- **асинхронные** - обмен данными не координируется.

Рекомендации по проектированию коммуникаций:

- количество коммуникаций у подзадач должно быть примерно одинаковым, иначе приложение становится плохо масштабируемым;
- там, где это возможно, следует использовать локальные коммуникации.





Укрупнение (агломерация)

На этапе укрупнения учитывается архитектура вычислительной системы, при этом часто приходится объединять (укрупнять) задачи, полученные на первых двух этапах, для того чтобы их число соответствовало числу процессоров.

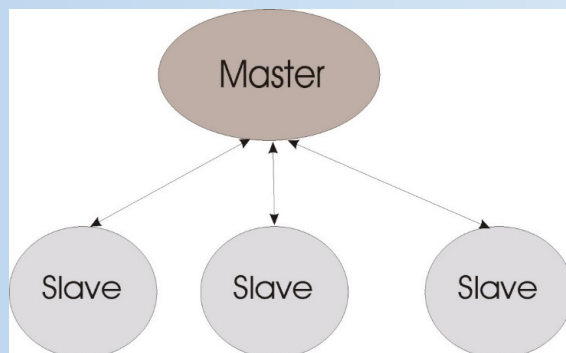
Требования

- снижение затрат на коммуникации;
- если при укрупнении приходится дублировать вычисления или данные, это не должно приводить к потере производительности и масштабируемости программы;
- результирующие (под)задачи должны иметь примерно одинаковую трудоемкость;
- сохранение масштабируемости.

Организация параллельной программы

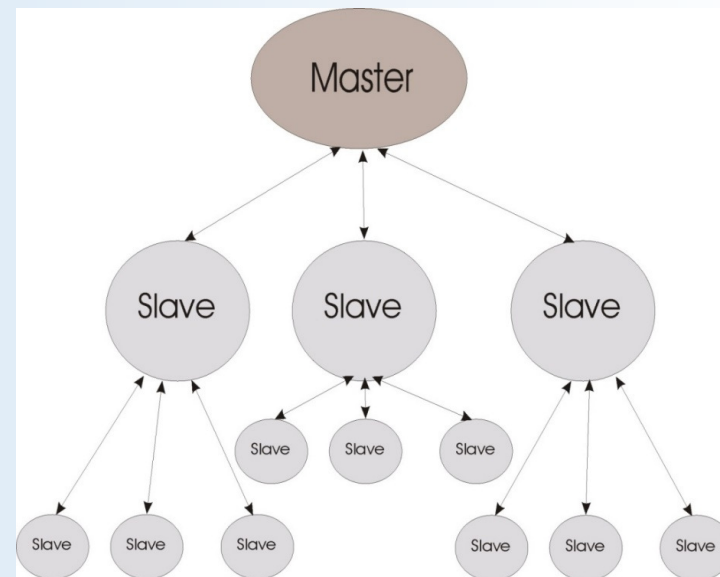
Простая схема хозяин/работник (Master/slave)

- Главная задача отвечает за размещение подчиненных задач.
- Подчиненная задача получает исходные данные для обработки от главной задачи и возвращает ей результат работы.



Иерархическая схема хозяин/работник (Master/slave)

- Подчиненные задачи разделены на непересекающиеся подмножества и у каждого из этих подмножеств есть своя главная задача.
- Главные задачи подмножеств управляются одной "самой главной" задачей.



Программные инструменты разработки параллельных программ

Низкоуровневые средства

Системные вызовы операционной системы (UNIX/Linux)

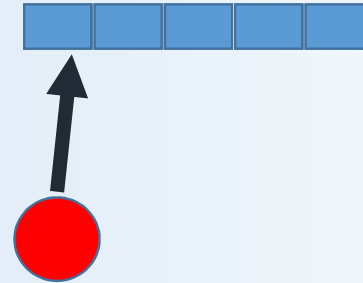
IPC (InterProcess Communications)

- именованные каналы;
- общая память;
- сообщения;
- сокеты;
- семафоры.

IPC. Сообщения. Пример

Сервер

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "mesg.h"
main()
{
    Message                message;
    key_t                  key;
    int                    msgid, length;
    message.mtype = 1L;
    if ((key = ftok("server", 'A')) < 0){
        printf("Невозможно получить ключ\n"); exit(1); }
    if ((msgid = msgget(key, 0)) < 0){
        printf("Невозможно получить доступ к очереди\n"); exit(1); }
    if ((length = sprintf(message.buf,
                          "Здравствуй, Мир!\n")) < 0){
        printf("Ошибка копирования в буфер\n"); exit(1); }
    if (msgsnd(msgid, (void *) &message, length, 0) !=0){
        printf("Ошибка записи сообщения в очередь\n");
        exit(1); }
    if (msgctl(msgid, IPC_RMID, 0) < 0){
        printf("Ошибка удаления очереди\n"); exit(1); }
    exit(0);
}
```

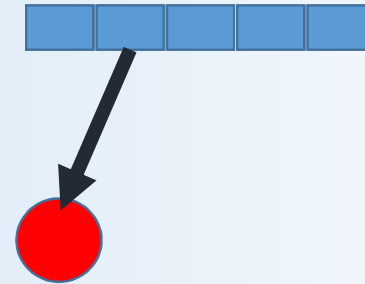


Клиент

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "mesg.h"
main()
{
    message  message;
    key_t    key;
    int msgid, length, n;

    if ((key = ftok("server", 'A')) < 0){
        printf("Невозможно получить ключ\n"); exit(1); }
    message.mtype=1L;
    if ((msgid = msgget(key, PERM | IPC_CREAT)) < 0){
        printf("Невозможно создать очередь\n"); exit(1); }
    n = msgrcv(msgid, &message, sizeof(message), message.mtype, 0);
    if (n > 0) {
        if (write(1, message.buf, n) != n) {
            printf("Ошибка вывода\n"); exit(1); }
    }
    else { printf("Ошибка чтения сообщения\n"); exit(1); }

    exit(0);
}
```



POSIX Threads

Стандарт **POSIX** реализации потоков (нитей) выполнения, определяющий API для создания и управления ими.

POSIX.1c, Расширения потоков (IEEE Std 1003.1c-1995)

- Создание, управление и завершение выполнения потоков
- Планировщик потоков
- Синхронизация потоков
- Обработка сигналов

Реализации стандарта содержат:

- ✓ функции управления потоками
- ✓ функции синхронизации потоков

POSIX Threads. Пример

```
#include <stdio.h>
#include "gettimeofday.h"
#include <pthread.h>

#define gNumThreads 1
#define N 100000000

double a[N + 1], b[N + 1], sum;
int i, j;
double start, stop;

const int gNumSteps = N;
double gVectorSum = 0;

void *threadFunction(void *arg)
{
    int i;
    int myNum = *((int *)arg);

    double partialSum = 0; // локально по отношению к каждому потоку

    for ( i = myNum; i < gNumSteps; i += gNumThreads ) // каждый gNumThreads-й шаг
    {
        partialSum += a[i] * b[i]; // параллельное вычисление сумм каждым потоком
    }
    gVectorSum += partialSum; // сложения частных сумм и получение результата

    return 0;
}
```

```
int main()
{
    pthread_t tid[gNumThreads];
    int      tNum[gNumThreads], i, j;

    // инициализация вектора
    for (j = 0; j < N; j++)
    {
        a[j] = 1.031; b[j] = 1.057;
    }

    printf("Computed value of vector sum: ");
    start = wcgettimeofday();

    for (i = 0; i < gNumThreads; i++)
    {
        tNum[i] = i;
        pthread_create(&tid[i], NULL, threadFunction, &tNum[i]);
    }

    for (i = 0; i < gNumThreads; i++)
        pthread_join(tid[i], NULL);

    stop = wcgettimeofday();

    printf("sum = %f\n", gVectorSum);

    printf("time = %g\n", stop - start);

}
```

Windows API

В Microsoft Windows имеется возможность разработки многопоточных приложений на C++ с помощью «стандартных» системных средств – прикладного программного интерфейса операционной системы.

Ссылка

<http://msdn.microsoft.com>

Windows API. Пример

```
#include <windows.h>
#include <stdio.h>
#define N 100000000

double a[N + 1], b[N + 1], sum;
int i, j;
double start, stop;

const int gNumSteps = N;
const int gNumThreads = 1;
double gVectorSum = 0;
CRITICAL_SECTION gCS;

DWORD WINAPI threadFunction(LPVOID pArg)
{
    int i;
    int myNum = *((int *)pArg);
    double partialSum = 0; // локально по отношению к каждому потоку
    for(i=myNum*(gNumSteps/gNumThreads); i<(myNum+1)*(gNumSteps/gNumThreads); i++)
// используется каждый gNumThreads-й шаг
    {
        partialSum += a[i] * b[i]; // вычисление частных сумм каждым потоком
    }
    EnterCriticalSection(&gCS);
    gVectorSum += partialSum; // сложение частного результата с глобальным
    LeaveCriticalSection(&gCS);
    return 0;
}
```

```
int main()
{
    HANDLE threadHandles[gNumThreads];
    int tNum[gNumThreads], i, j;

    for (j = 0; j < N; j++)
    {
        a[j] = 1.031; b[j] = 1.057;
    }

    printf("Computed value of dot product: ");
    InitializeCriticalSection(&gCS);
    for ( i = 0; i < gNumThreads; ++i )
    {
        tNum[i] = i;
        threadHandles[i] = CreateThread(NULL,          // атрибуты безопасности
                                        0,            // размер стека
                                        threadFunction, // функция потока
                                        (LPVOID)&tNum[i], // данные для функции потока
                                        0,            // режим запуска потока
                                        NULL); // возвращаемый идентификатор потока
    }
    WaitForMultipleObjects(gNumThreads, threadHandles, TRUE, INFINITE);
    DeleteCriticalSection(&gCS);
    printf("sum = %f\n", gVectorSum);
}
```

Высокоуровневые средства

Open Multi-Processing (OpenMP)

OpenMP - стандарт программного интерфейса приложений для параллельных систем с общей памятью. Поддерживает языки C, C++, Fortran.

Разработкой стандарта занимается OpenMP ARB (Architecture Board). Постоянные члены Совета:

- **AMD** (Greg Stoner)
- **ARM** (Chris Adeniyi-Jones)
- **Convey Computer** (Kirby Collins)
- **Cray** (James Beyer/Luiz DeRose)
- **Fujitsu** (Eiji Yamanaka)
- **HP** (Sujoy Saraswati)
- **IBM** (Kelvin Li)
- **Intel** (Xinmin Tian)
- **NEC** (Kazuhiro Kusano)
- **NVIDIA** (Jeff Larkin)
- **Oracle Corporation** (Nawal Copty)
- **Red Hat** (Matt Newsome)
- **ST Microelectronics** (Christian Bertin)
- **Texas Instruments** (Andy Fritsch)

Первая версия OpenMP появилась в 1997 (Fortran) / 1998 (C/C++) годах. Последняя версия OpenMP 3.0 (2008 год).

OpenMP 4.0 (2013 год):

«Our main aim is close on the OpenMP 4.0 Draft.

Topics to be covered in this meeting include: Tasking, Fortran 2003 base language support, Affinity, Accelerators, Error Models, and SIMD»

Официальный сайт OpenMP

<http://openmp.org>

OpenMP. Пример

```
#include <windows.h>
#include <stdio.h>
#define N 100000000

double a[N + 1], b[N + 1];
int i;
double start, stop;
double gDotProduct = 0;

int main()
{
    // initialize vectors
    for (i = 0; i < N; i++)
    {
        a[i] = 1.034; b[i] = 1.057;
    }
    printf("Computed value of vector sum: ");
    start = omp_get_wtime();
    #pragma omp parallel for reduction(+:gDotProduct)
    for ( i = 0; i < N; i++ )
    {
        gDotProduct += a[i] * b[i];
    }

    stop = omp_get_wtime();

    printf("sum = %f\n", gDotProduct);
    printf("time = %g\n", stop - start);
}
```

Заключение

В этой лекции мы рассмотрели:

- особенности последовательной и параллельной моделей программирования;
- Более и менее распространенные инструменты параллельного программирования.

Задание на дом

- В чём вы видите ограничения последовательной модели программирования?
- Какие достоинства и недостатки у подходов, основанных на параллелизме данных и параллелизме задач?
- В чём достоинства и недостатки различных видов коммуникаций?
- Следуя приведённым в лекции ссылкам, познакомьтесь с основными особенностями наиболее известных инструментов разработки параллельных и многопоточных программ.