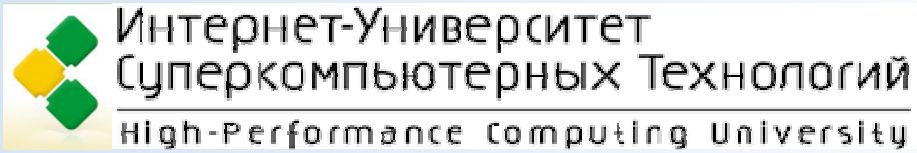
Основы параллельного программирования с использованием MPI

Лекция 2

Немнюгин Сергей Андреевич
Санкт-Петербургский государственный университет
кафедра вычислительной физики

snemnyugin@mail.ru



Лекция 2

Аннотация

В этой лекции продолжается обзор программных средств реализации параллелизма, затем дается краткий обзор истории создания Message Passing Interface, а также некоторых реализаций MPI. Объясняется роль демона mpd. Вводятся основные понятия и терминология.

План лекции

- Обзор программных средств реализации параллелизма.
- Спецификация MPI. История создания, версии.
- Основные понятия, терминология.

Высокоуровневые средства

Intel® Cilk™ Plus

Программа на **Intel® Cilk™ Plus** пишется в семантике последовательного программирования. Фрагменты для распараллеливания расщепляются на подзадачи, связанные отношениями подчинения («родитель»-«потомок»).

Программист, использующий **Cilk™ Plus** должен думать о том, ***что*** следует распараллелить, а не ***как***. В этом – одно из отличий от OpenMP-программирования.

Балансировкой занимается runtime-система. Балансировка выполняется методом захвата работы.

Дополнительные средства работы с массивами (расширенная индексная нотация – аналог сечений массивов в языке Fortran).

Гиперобъекты.

Векторизация.

Пример

```
class CilkSeidelSolver : public Solver {
public:
    virtual void Solve() {
        if (p == 0) return;
        const int N = p->GetMeshSize();
        double variation(precision), oldValue, diff;
        double step = p->Step(); int count = 0;

        double **u = p->U();
        double **f = p->F();

        while(variation >= precision) {
            count++;
            variation = 0;
            cilk::reducer_max<double> reducer(0);
            cilk_for(int i=1; i<N+1; i++) {
                double vm;
                vm = 0;
                for(int j=1; j<N+1; j++) {
                    oldValue = u[i][j];
```

Message Passing Interface (MPI)

Message Passing Interface (MPI) - *Интерфейс Передачи Сообщений*, спецификация разработанная в 1993—1994 годах группой MPI Forum, в состав которой входили представители академических и промышленных кругов. Она стала первым стандартом систем передачи сообщений.

Официальный сайт MPI
http://www.mpi-forum.org

Пример

```
#include <mpi.h>
#include <stdio.h>
int main ( int argc, char *argv[] ) {
        int ProcNum, ProcRank, tmp;
        MPI_Status status;
        MPI_Init ( &argc, &argv );
        MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
        MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
        if(ProcRank == 0){
                printf("Hello world from process %i \n", ProcRank);
                for(int i = 1; i < ProcNum; i++){
        MPI_Recv(&tmp,1,MPI_INT,MPI_ANY_SOURCE,0,MPI_COMM_WORLD, &status);
                        printf("Hello world from process %i \n", tmp);
                }
        }else{
                MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
        }
        MPI_Finalize();
        return 0;
}
```

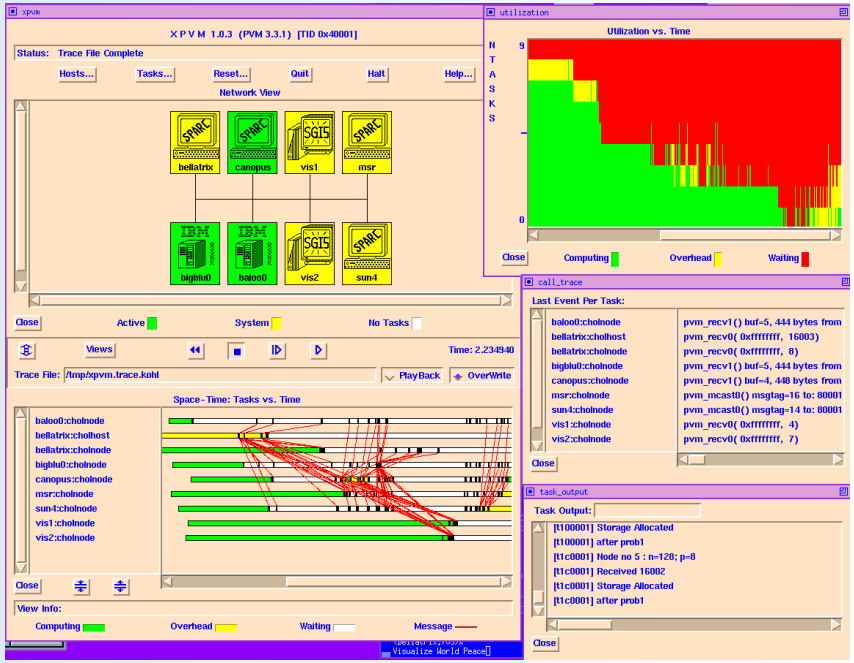
Parallel Virtual Machine (PVM)

PVM (Parallel Virtual Machine) - позволяет объединить разнородный набор компьютеров, связанных сетью, в общий вычислительный ресурс, который называют Параллельной Виртуальной Машиной.
Последняя версия 3.4.6.

Официальный сайт PVM
http://www.csm.ornl.gov/pvm

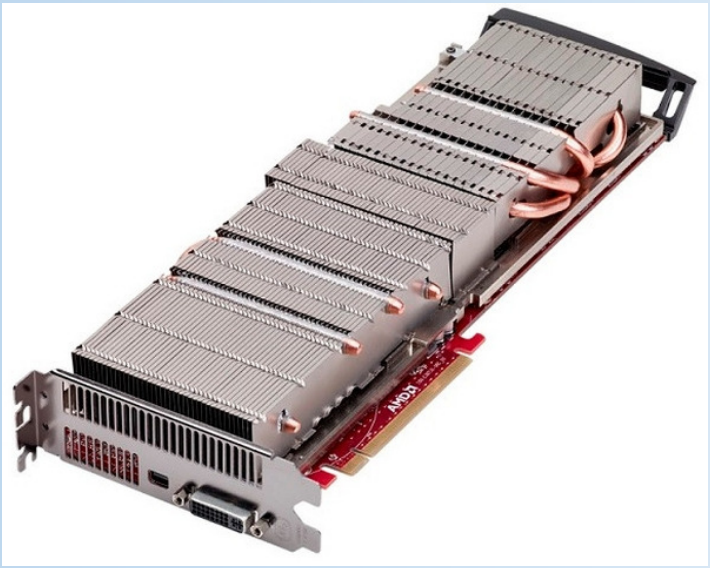
```
void network::field_send_cpy( struct cell* cell, int ptid, int time_step )
// get from cell
// send to ptid
{
  static error_handler bob("network::field_send",errname);
  int msgtag = time_step;
  double data[9];

  data[0] = cell->fp;
  data[1] = cell->gm;
  data[2] = cell->fm;
  data[3] = cell->gp;
  data[4] = cell->ex;
  data[5] = cell->ey;
  data[6] = cell->ez;
  data[7] = cell->by;
  data[8] = cell->bz;
  pvm_initsend( PvmDataDefault );
  pvm_pkdouble( data, 9, 1 );
  pvm_send( ptid, msgtag );
}
```



Программные инструменты для графических процессоров общего назначения

CUDA (NVIDIA)
OpenCL
OpenACC



CUDA (Compute Unified Device Architecture)

CUDA – архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров).
Сайт в Интернете:
https://developer.nvidia.com/category/zone/cuda-zone

```
#include <stdio.h>
__global__ void incKernel (float * data)
{
   int idx = blockIdx.x * blockDim.x + threadIdx.x;
   data [idx] = data [idx] + 1.0f;
}

int main ( int argc, char *  argv [] )
{
    int n        = 16 * 1024 * 1024;
intnumBytes = n * sizeof ( float );
float * a = new float [n];
for ( int i = 0; i < n; i++ )
a [i] = 0.0f;
    float * dev = NULL;
    cudaMalloc ((void**)&dev, numBytes);
    dim3 threads = dim3(512, 1);
    dim3 blocks  = dim3(n / threads.x, 1);
    cudaMemcpy(dev, a, numBytes, cudaMemcpyHostToDevice);
    incKernel<<<blocks, threads>>>(dev);
    cudaMemcpy(a, dev, numBytes, cudaMemcpyDeviceToHost);
    cudaFree(dev);
    delete a;
return 0;
}
```

OpenCL

Открытый, свободно распространяемый стандарт разработки приложений для гетерогенных (обычные процессоры+графические ускорители) параллельных вычислительных систем.

Прикладной программный интерфейс. Поддерживается собственная программная модель.

Сайт в Интернете:
http://www.khronos.org/opencl/

```
…
int
Device::createKernel()
{
    kernel = clCreateKernel(program, "rxFieldRestore", &status);
    if(!sdkObject.checkVal(status, CL_SUCCESS, "clCreateKernel failed."))
        return SDK_FAILURE;

    return SDK_SUCCESS;
…
int
Device::enqueueKernel()
{
          size_t globalThreads = owidth;
          size_t localThreads =  64;
    status = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &globalThreads,
                                    &localThreads, 0, NULL, &eventObject);
…
```

OpenACC

Набор директив компилятору, позволяющих описать участки кода программ на C/C++/Fortran, которые могут быть переданы для выполнения с центрального процессора на ускоритель.

Поддерживается собственная программная модель.

Ведётся работа по объединению OpenACC с OpenMP.

Сайт в Интернете:
http://www.openacc-standard.org

Пример

```
#pragma acc data copyin(aP[0:size], aW[0:size], aE[0:size], aS[0:size], aN[0:size],
b[0:size]) copy(temp_red[0:size_temp], temp_black[0:size_temp])

for (iter = 1; iter <= it_max; ++iter) {
  Real norm_L2 = 0.0;

  #pragma omp parallel for shared(aP, aW, aE, aS, aN, temp_black, temp_red) \
      reduction(+:norm_L2)
  #pragma acc kernels present(aP[0:size], aW[0:size], aE[0:size], aS[0:size], aN[0:size],
b[0:size], temp_red[0:size_temp], temp_black[0:size_temp])
  #pragma acc loop independent gang vector(4)
  for (int col = 1; col < NUM + 1; ++col) {
    #pragma acc loop independent gang vector(64)
    for (int row = 1; row < (NUM / 2) + 1; ++row) {
      int ind_red = col * ((NUM / 2) + 2) + row;
      int ind = 2 * row - (col % 2) - 1 + NUM * (col - 1);
      #pragma acc cache(aP[ind], b[ind], aW[ind], aE[ind], aS[ind], aN[ind])
…
```

Другие программные инструменты разработки высокопроизводительных приложений

Библиотеки линейной алгебры

ScaLAPACK – реализация LAPACK для систем с распределённой памятью (на основе MPI).

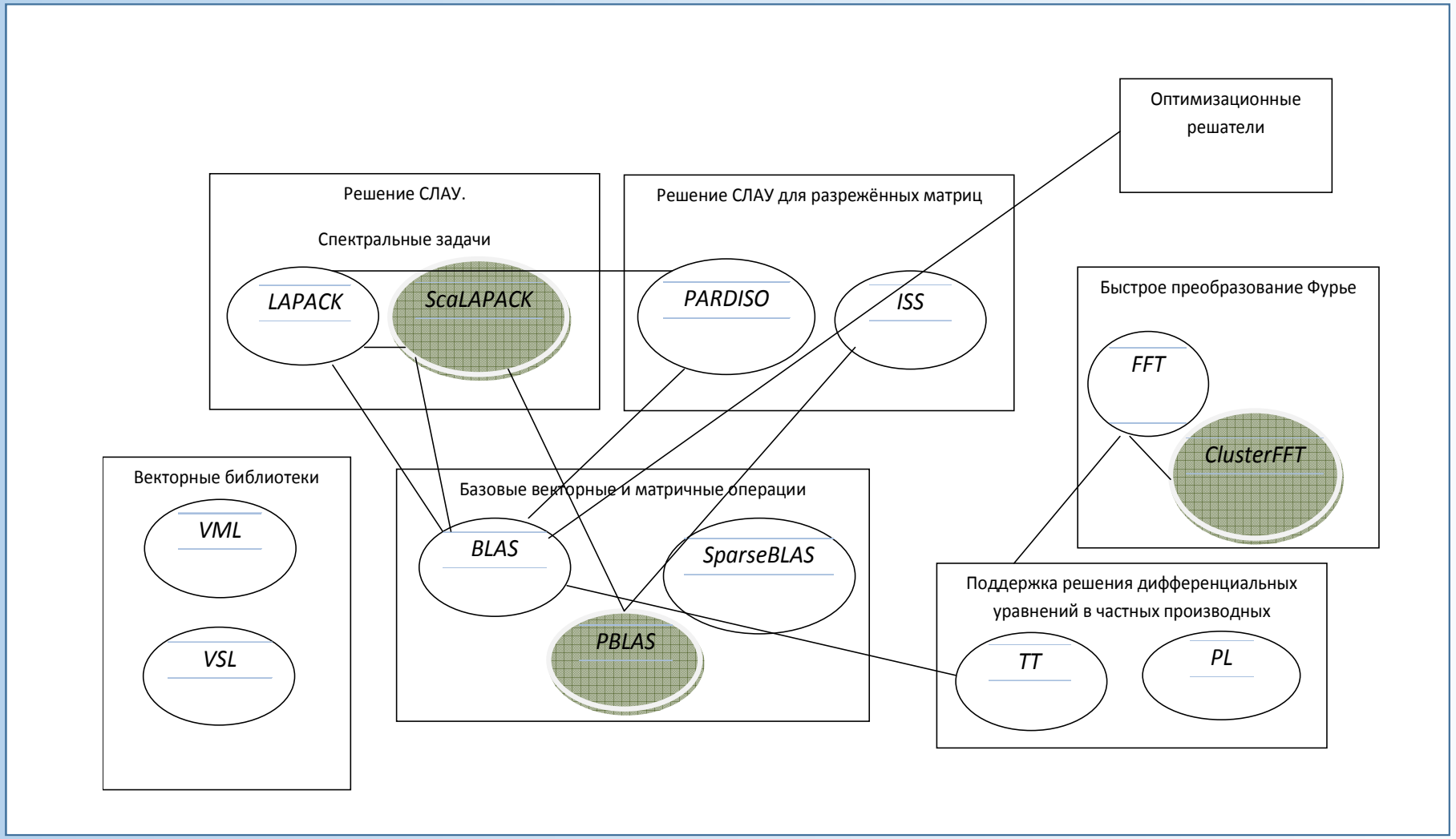
PLASMA – реализация LAPACK для многоядерных систем (Pthread, Linux).

MAGMA – реализация LAPACK для гибридных архитектур (с использованием графических ускорителей, Pthread, CUDA, Linux).

SuperLU - 3 варианта: последовательный, параллельный для систем с общей памятью (Pthreads, OpenMP) и параллельный для систем с распределённой памятью (MPI).

PaStiX – библиотека функций для решения СЛАУ с большими разреженными матрицами для вычислительных систем с распределённой памятью.

Intel® Math Kernel library



Intel ® Integrated Performance Primitives

Библиотека готовых компонентов для разработки мультимедийных приложений для вычислительных платформ Intel.

Включает модули для обработки сигналов и выполнения векторных и матричных операций, функции сжатия и распаковки речи и статических/динамических изображений, средства шифрования и обработки аудиоданных и текстовых строк.

Intel IPP обеспечивает прозрачное использование расширенных возможностей процессоров Intel, таких, как технология MMX, расширения набора команд Streaming SIMD Extensions и Streaming SIMD Extensions.

Библиотека Intel IPP оптимизирована для работы с разными процессорами Intel.

Библиотека Intel IPP поддерживает 32- и 64-битные операционные системы Windows и Linux, включая встраиваемые версии, такие как Windows Mobile.

Intel® Threading Building Blocks

Параллельные алгоритмы
parallel_for
parallel_for_each
parallel_invoke
parallel_do
parallel_scan
parallel_sort
parallel_[deterministic]_reduce

Macro Dataflow
parallel_pipeline
tbb::flow::...

Concurrent Containers
concurrent_hash_map
concurrent_unordered_{map,set}
concurrent_[bounded_]queue
concurrent_priority_queue
concurrent_vector

Диспетчер задач
task_group, structured_task_group
task
task_scheduler_init
task_scheduler_observer

Thread Local Storage
combinable
enumerable_thread_specific

Примитивы синхронизации
atomic, condition_variable
[recursive_]mutex
{spin,queuing,null} [_rw]_mutex
critical_section, reader_writer_lock

Размещение памяти
tbb_allocator
zero_allocator
cache_aligned_allocator
scalable_allocator

Потоки
std::thread

Спецификация MPI.
История создания. Версии

Интерес к системам передачи сообщений возник в 1980-е годы. Его следствием стало появление большого количества реализаций, создававшихся разными коллективами разработчиков и предназначенных для разных вычислительных систем.

Примеры:
p4, **PICL**, **PARMACS**, **PVM**, **TCGMSG**, **Zipcode**, **Express** и другие

Возникла необходимость координировать и стандартизировать этот процесс, поэтому в рамках конференции **Supercomputing'92** состоялось совещание, на котором было принято решение о разработке стандарта программного интерфейса обмена сообщениями.

Процесс стандартизации был поддержан индустрией:

C*onvex*, *Cray*, *IBM*, *Intel*, *Meiko*, *nCUBE*, *NEC*, *Thinking Machines*

Индустрия заинтересована в средстве разработки эффективных программ для высокопроизводительных вычислительных систем.

Сайт, на котором можно найти официальные документы MPI:
http://www.mpi-forum.org

Версия MPI-1 вышла в 1994 году.

Версия MPI-2 вышла в 1998 году, первая реализация появилась в 2002 году.

Версия MPI-2.2 вышла в 2009 году.

Версия MPI-3 вышла 21 сентября 2012 года.

Спецификация MPI-1

Содержит описание стандарта программного интерфейса обмена сообщениями.

Спецификация учитывает опыт предшествующих разработок и ориентирована на большую часть аппаратных платформ. Несмотря на то, что MPI рассчитано на использование с языками C/C++ и Fortran, семантика в значительной степени не зависит от языка.

В MPI-1 описываются интерфейсы процедур двухточечных и коллективных обменов, сбора информации, организации обменов в группах процессов, синхронизации процессов, виртуальные топологии, привязки к языкам программирования и т. д.

Спецификация MPI-2

Является дальнейшим развитием MPI.

Новое в MPI-2 (по сравнению с MPI-1):

- возможность создания новых процессов во время выполнения MPI-программы (в MPI-1 количество процессов фиксировано, крах одного приводит к краху всей программы);
- новые разновидности двухточечных обменов (односторонние обмены);
- новые возможности коллективных обменов;
- поддержка внешних интерфейсов;
- поддержка многопоточности;
- операции параллельного ввода-вывода с файлами.

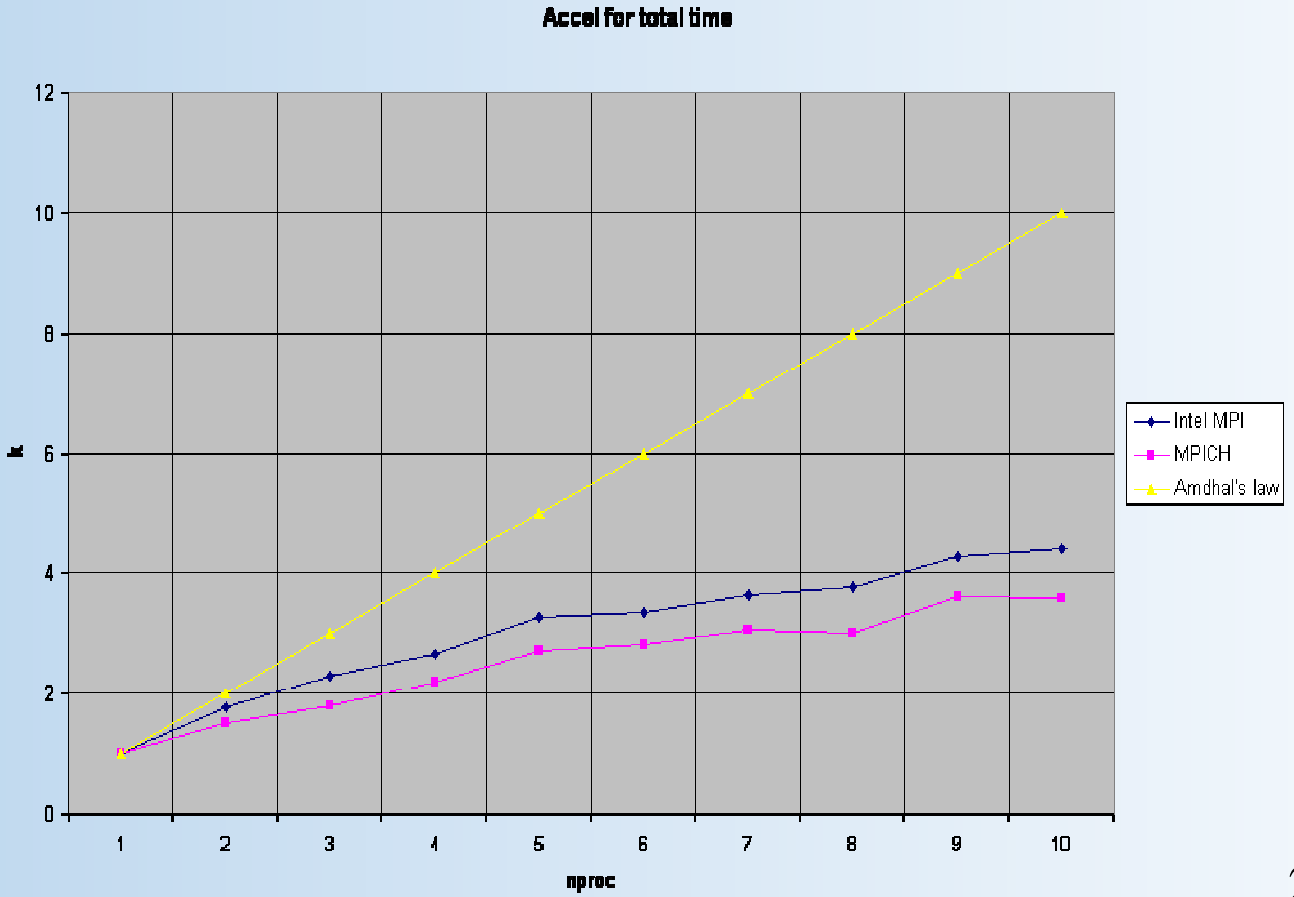
Спецификация MPI-3

Новое в MPI-3 (по сравнению с MPI-2):

- поддержка неблокирующих коллективных операций;
- топологические коллективные операции;
- новые возможности коллективных обменов, в том числе «разрежённые» коллективные обмены;
- расширенная поддержка RMA (Remote Memory Access) – односторонние обмены;
- новые интерфейсы для работы с инструментальными средствами;
- удалены или изменены некоторые функции, типы данных и т.д.;
- другие изменения.

Масштабируемость MPI-программ. Сравнение двух реализаций

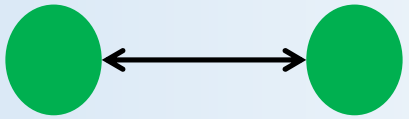
Зависимость ускорения параллельной версии кода ACE от количества узлов:



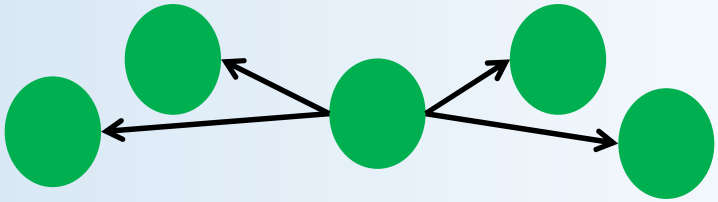
Основные понятия, терминология

Целевая архитектура

Двухточечные обмены используются для организации локальных и неструктурированных коммуникаций.

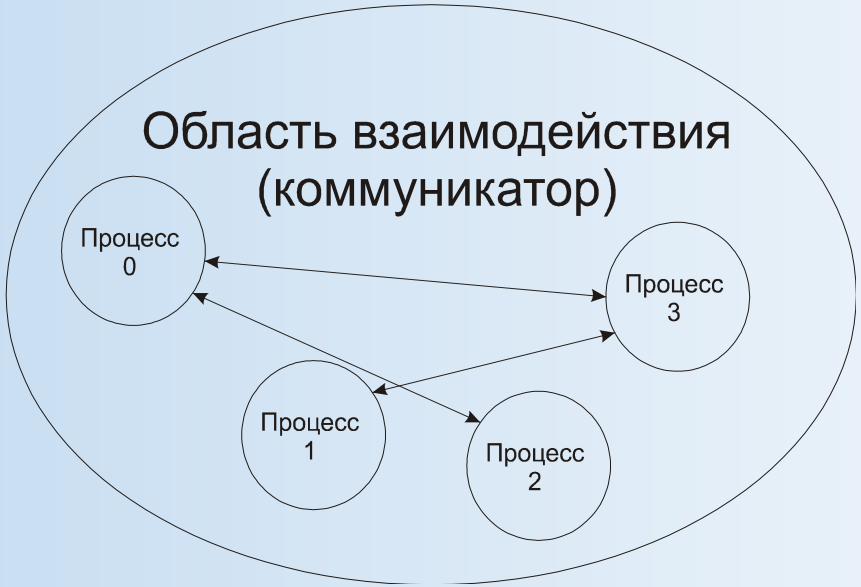


При выполнении глобальных операций используются *коллективные обмены*.



Асинхронные коммуникации реализуются с помощью запросов о получении сообщений.

Область взаимодействия (область связи) определяет группу процессов. Все процессы, принадлежащие одной области взаимодействия, могут обмениваться сообщениями. Описывает это множество процессов специальная информационная структура, которая называется *коммуникатором*. Она позволяет скрыть от программиста внутренние коммуникационные структуры.



Коммуникаторы, создаваемые по умолчанию:

- **MPI_COMM_WORLD** - содержит все процессы;
- **MPI_COMM_SELF** - коммуникатор, содержащий только вызывающий процесс;
- **MPI_COMM_NULL** - пустой коммуникатор.

Сообщение содержит пересылаемые данные и служебную информацию:

- идентификатор процесса-отправителя сообщения (*ранг* процесса);
- адрес, по которому размещаются пересылаемые данные процесса-отправителя;
- тип пересылаемых данных;
- количество данных (размер буфера сообщения для того, чтобы принять сообщение, процесс должен отвести для него достаточный объем оперативной памяти);
- идентификатор процесса, который должен получить сообщение;
- адрес, по которому должны быть размещены данные процессом-получателем;
- идентификатор коммуникатора, описывающего область взаимодействия, внутри которой происходит обмен.

Ранг источника дает возможность различать сообщения, приходящие от разных процессов

Тег - задаваемое пользователем целое число (от 0 до 32767), идентификатор сообщения.

Теги могут использоваться для соблюдения определенного порядка приема сообщений.