MPI

Введение

Message Passing Interface (MPI, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу.

Первая версия **MPI** разрабатывалась в 1993—1994 году, и MPI 1 вышла в 1994.

Стандарты MPI

- MPI 1 - 1994.
- MPI 1.1 -12 июня 1995 года
- MPI 2.0 - 18 июля 1997 года
- MPI 2.1 - сентябрь 2008 года
- MPI 2.2 - 4 сентября 2009 года
- **MPI 3.0 - 21 сентября 2012 года**

Реализации MPI

- **MPICH** — одна из самых распространенных реализация MPI, работает на UNIX и Windows-системах
- Open MPI — ещё одна свободная реализация MPI. Основана на более ранних проектах FT-MPI, LA-MPI, LAM/MPI и PACX-MPI.
- MPI/PRO for Windows NT — коммерческая реализация для Windows NT
- Intel MPI — коммерческая реализация для Windows / Linux
- **Microsoft MPI** входит в состав Compute Cluster Pack SDK. Основан на MPICH2, но включает дополнительные средства управления заданиями. Поддерживается спецификация MPI-2.
- HP-MPI — коммерческая реализация от HP
- SGI MPT — платная библиотека MPI от SGI
- Mvapich — свободная реализация MPI для Infiniband
- Oracle HPC ClusterTools — бесплатная реализация для Solaris SPARC/x86 и Linux на основе Open MPI
- MPJ — MPI for Java
- MPJ Express — MPI на Java

Microsoft MPI

Microsoft MPI (MS-MPI) is a Microsoft implementation of the Message Passing Interface standard for developing and runn applications on the Windows platform.

MS-MPI offers several benefits:

- Ease of porting existing code that uses MPICH.
- Security based on Active Directory Domain Services.
- High performance on the Windows operating system.
- Binary compatibility across different types of interconnectivity options.

MS-MPI downloads

The following are current downloads for MS-MPI:

- MS-MPI v7.1 (new!) - see Release notes
- Debugger for MS-MPI Applications with HPC Pack 2012 R2

Earlier versions of MS-MPI are available from the Microsoft Download Center.

Community resources

- Windows HPC MPI Forum
- Contact the MS-MPI Team

Microsoft high performance computing resources

- Featured tutorial: How to compile and run a simple MS-MPI program
- Featured guide: Set up a Windows RDMA cluster with HPC Pack and A8 and A9 instances to run MPI applications
- Microsoft High Performance Computing for Developers
- Microsoft HPC Pack (Windows HPC Server) Technical Library
- Azure HPC Scenarios

Related topics

MPI Reference

Microsoft MPI v7.1

Language: English Download

Stand-alone, redistributable and SDK installers for Microsoft MPI.

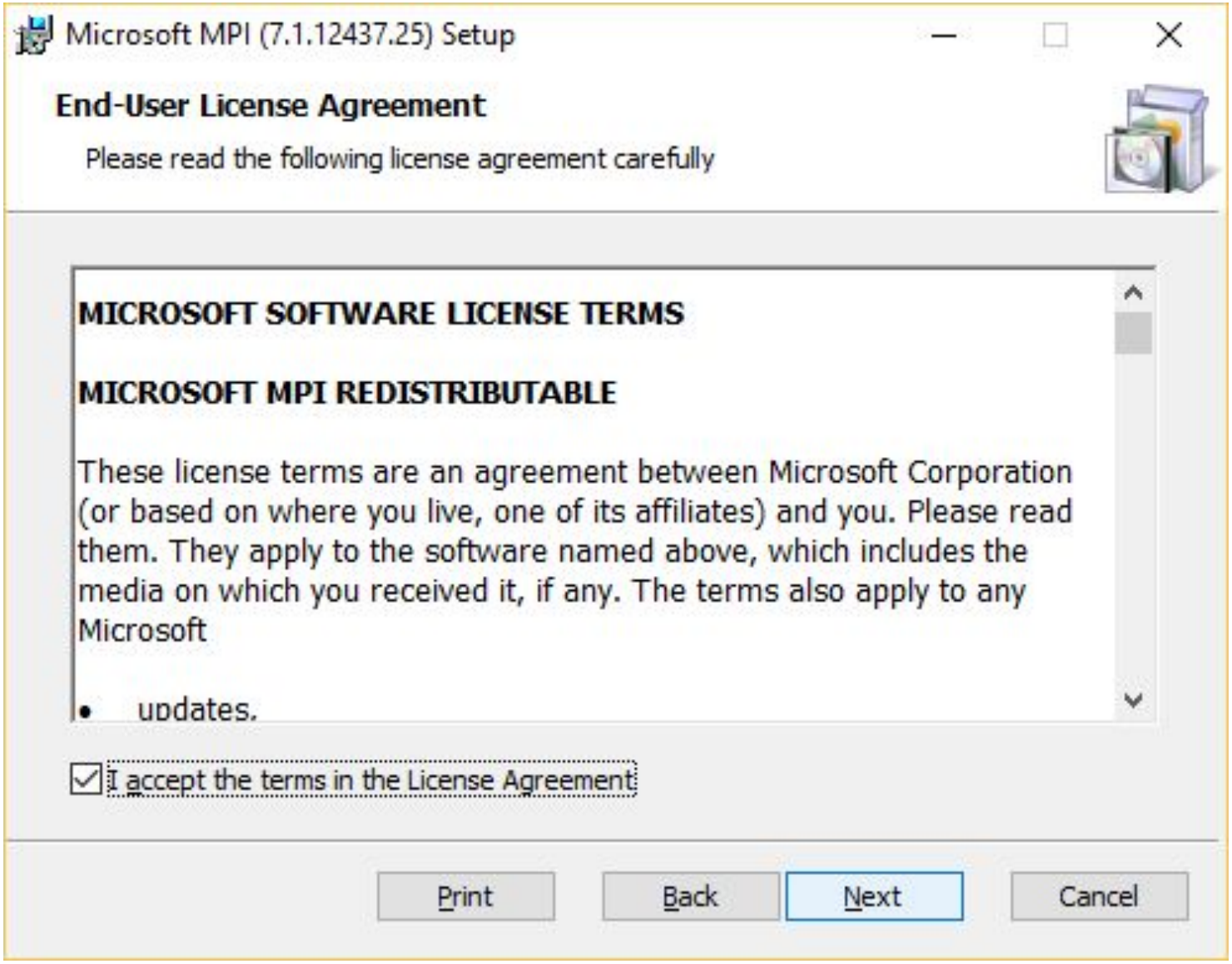
⊕ Details

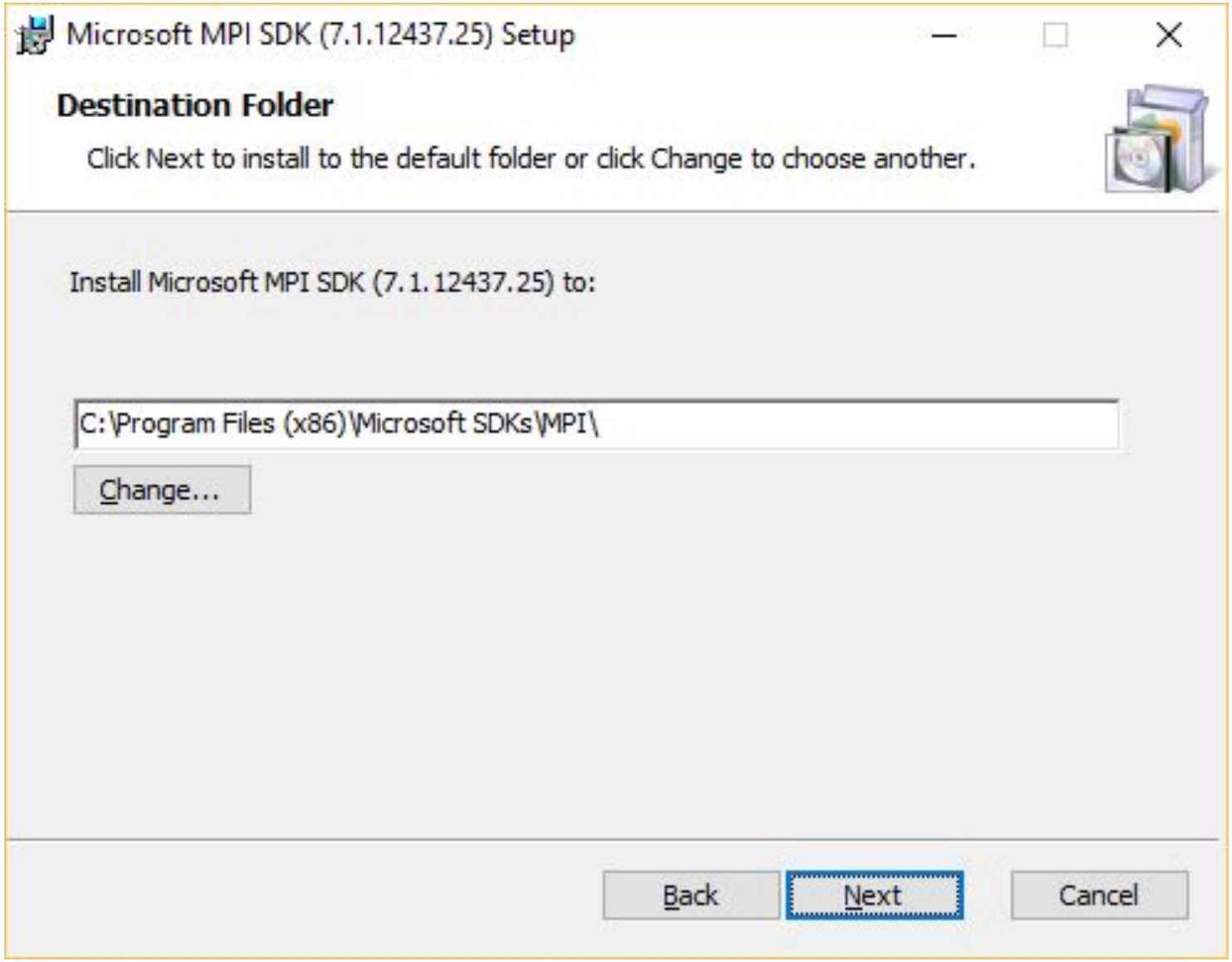
⊕ System Requirements

⊕ Install Instructions

Choose the download you want

File Name	Size
msmpisdk.msi	2.2 MB
MSMpiSetup.exe	5.1 MB


Microsoft MPI (7.1.12437.25) Setup
End-User License Agreement
Please read the following license agreement carefully
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT MPI REDISTRIBUTABLE
These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft
• updates.
I accept the terms in the License Agreement
Print
Back
Next
Cancel


Microsoft MPI SDK (7.1.12437.25) Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another.
Install Microsoft MPI SDK (7.1.12437.25) to:
C:\Program Files (x86)\Microsoft SDKs\MPI\
Change...
Back
Next
Cancel

Microsoft MPI

Microsoft MPI (MS-MPI) is a Microsoft implementation of the Message Passing Interface standard for developing and running pa applications on the Windows platform.

MS-MPI offers several benefits:

- Ease of porting existing code that uses MPICH.
- Security based on Active Directory Domain Services.
- High performance on the Windows operating system.
- Binary compatibility across different types of interconnectivity options.

MS-MPI downloads

The following are current downloads for MS-MPI:

- MS-MPI v7.1 (new!) - see Release notes
- Debugger for MS-MPI Applications with HPC Pack 2012 R2

Earlier versions of MS-MPI are available from the Microsoft Download Center.

Community resources

- Windows HPC MPI Forum
- Contact the MS-MPI Team

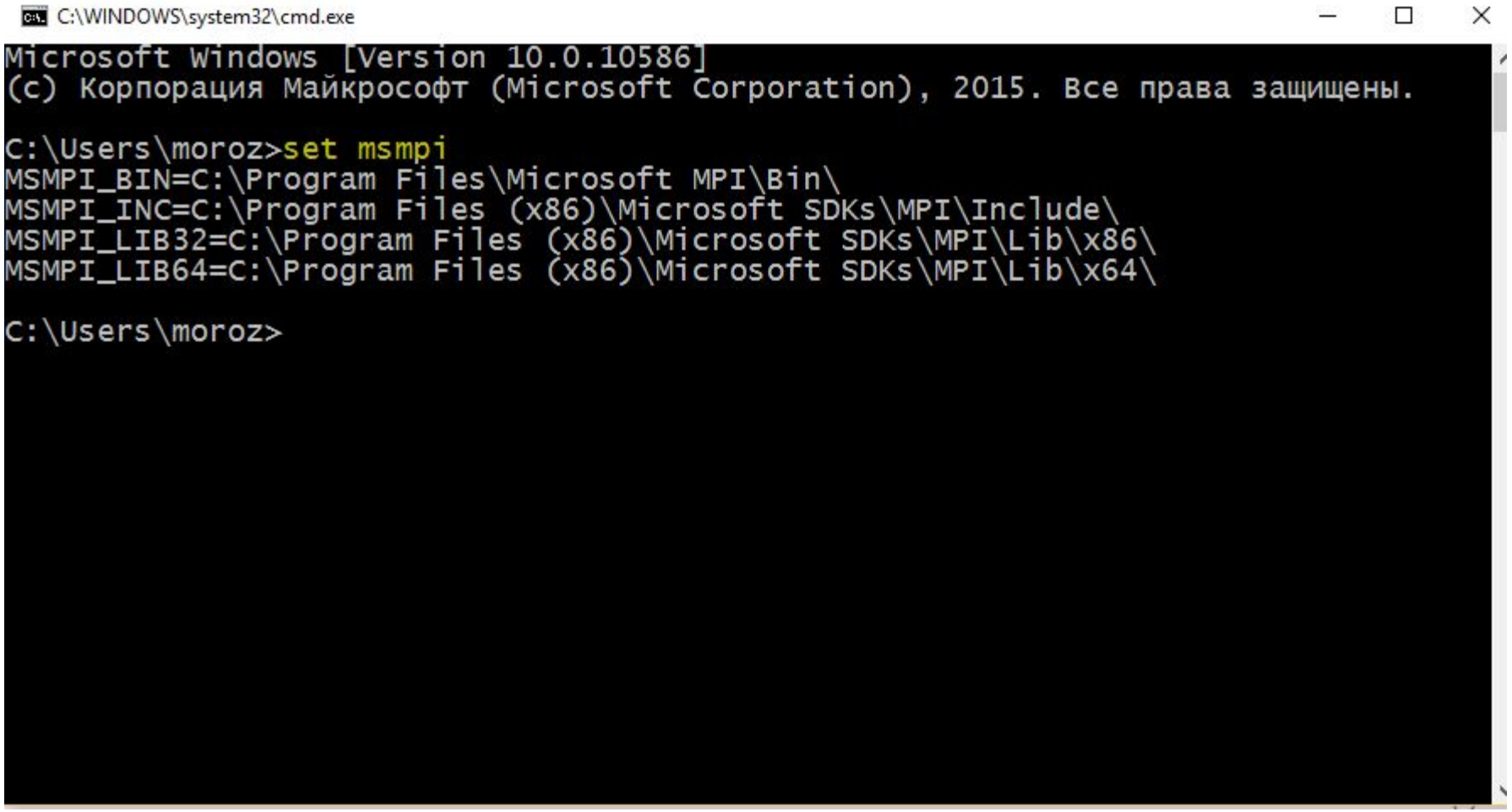
Microsoft high performance computing resources

- Featured tutorial: How to compile and run a simple MS-MPI program
- Featured guide: Set up a Windows RDMA cluster with HPC Pack and A8 and A9 instances to run MPI applications
- Microsoft High Performance Computing for Developers
- Microsoft HPC Pack (Windows HPC Server) Technical Library
- Azure HPC Scenarios

Related topics

MPI Reference

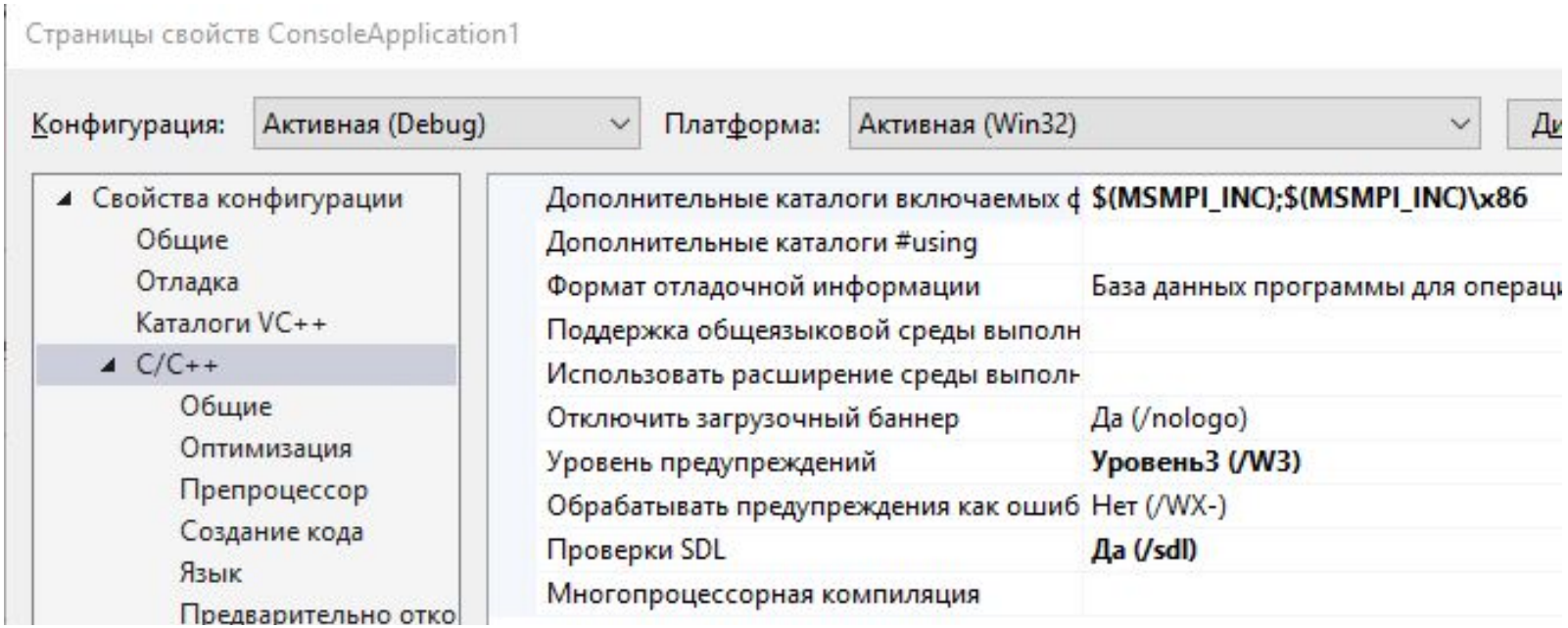
Настройка системы

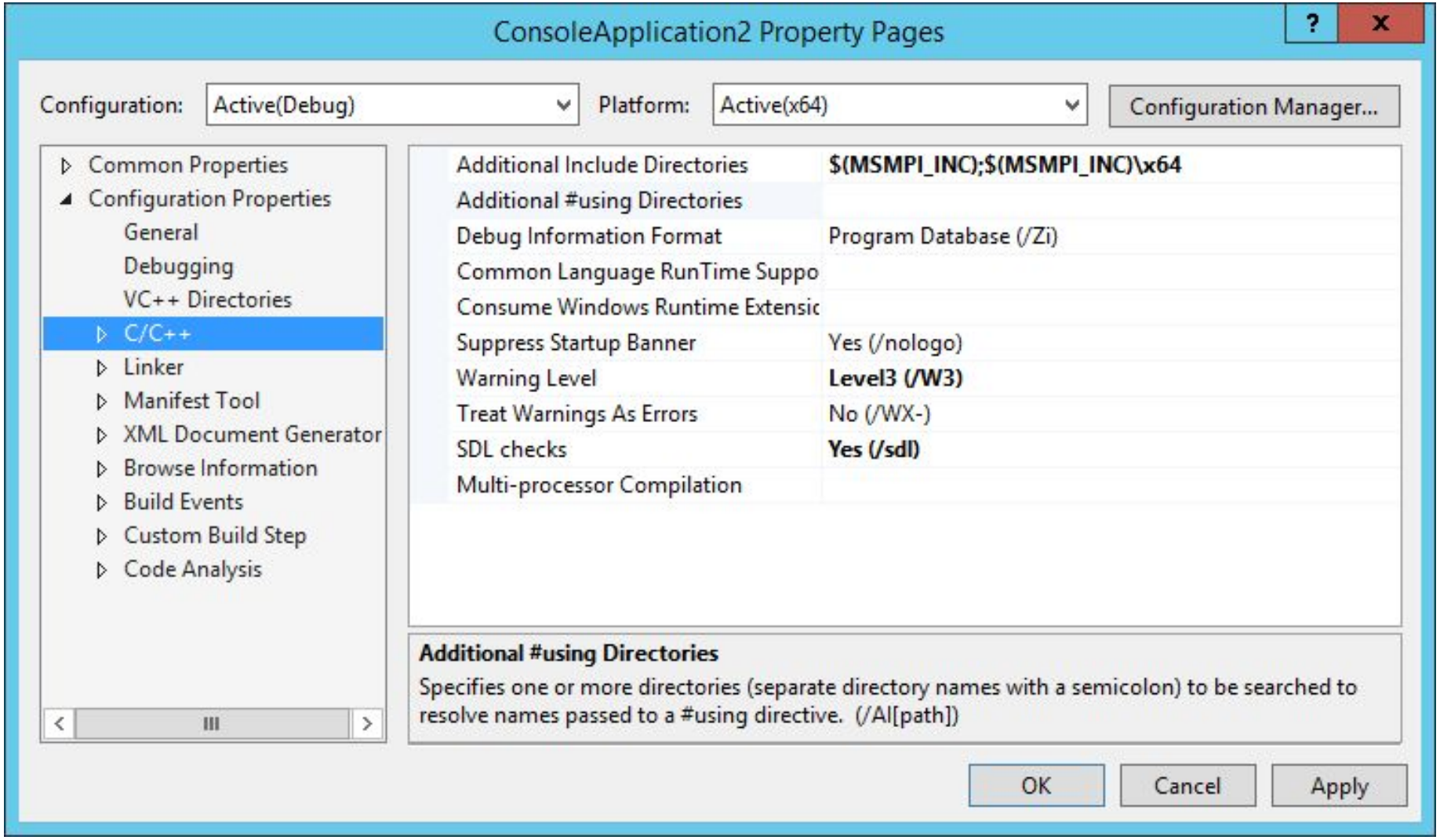


В настройках проекта установить:

- Дополнительные каталоги включаемых файлов:
  - $(MSMPI_INC);$(MSMPI_INC)\x86

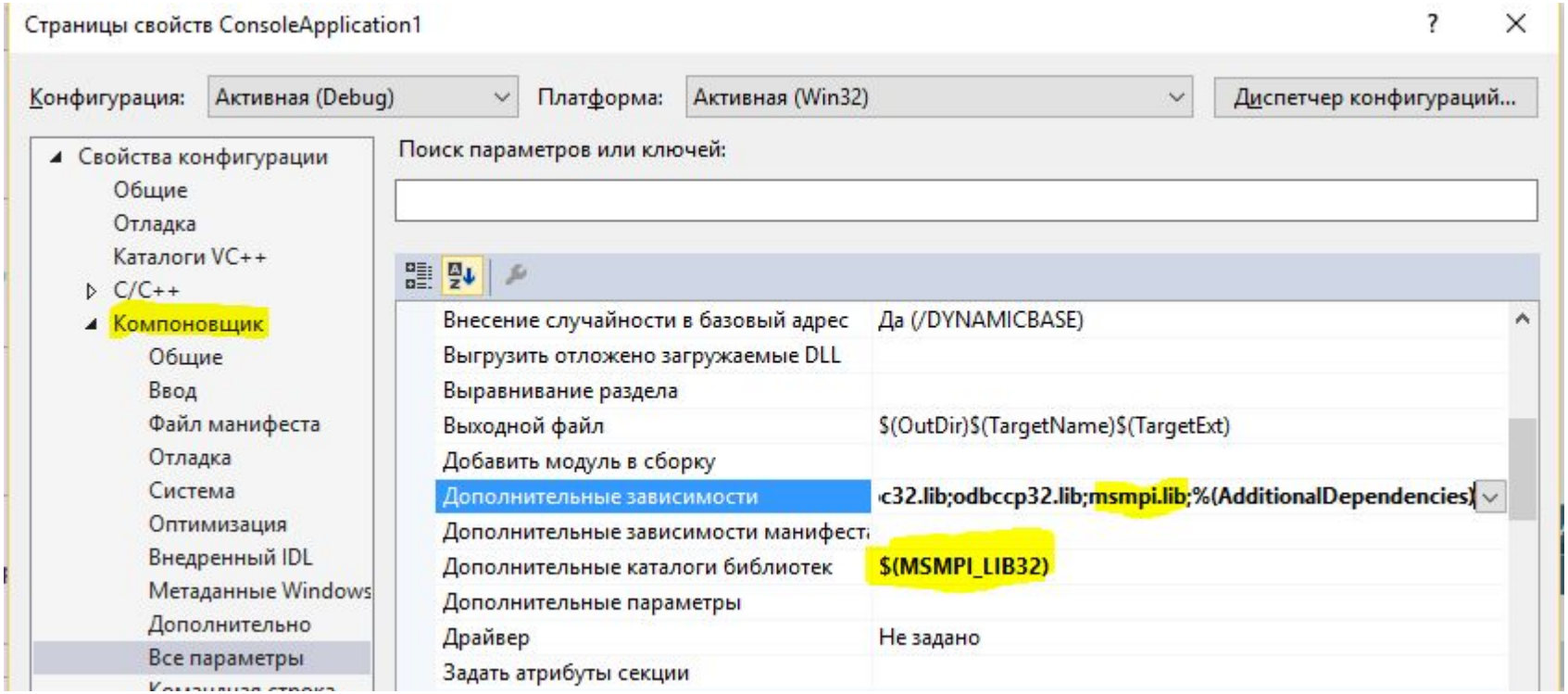
  или
  - $(MSMPI_INC);$(MSMPI_INC)\x64

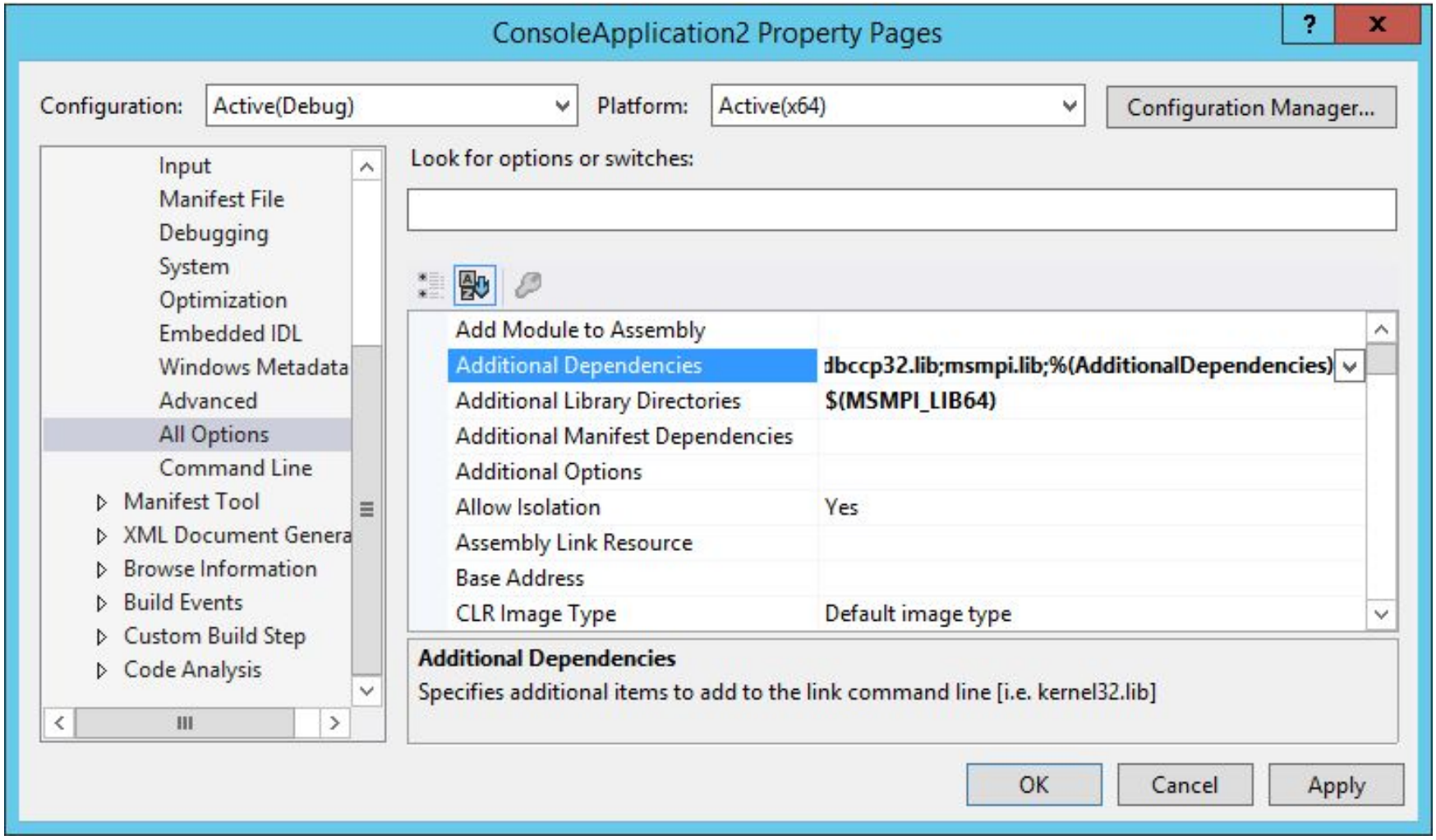



ConsoleApplication2 Property Pages
Configuration: Active(Debug)
Platform: Active(x64)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
Additional Include Directories
$(MSMPI_INC);$(MSMPI_INC)\x64
Additional #using Directories
Debug Information Format
Program Database (/Zi)
Common Language RunTime Suppo
Consume Windows Runtime Extensic
Suppress Startup Banner
Yes (/nologo)
Warning Level
Level3 (/W3)
Treat Warnings As Errors
No (/WX-)
SDL checks
Yes (/sdl)
Multi-processor Compilation
Additional #using Directories
Specifies one or more directories (separate directory names with a semicolon) to be searched to resolve names passed to a #using directive. (/AI[path])
OK
Cancel
Apply

- В меню «Компоновщик»:
  - Добавить в «Дополнительные зависимости» файл **msmpi.lib**
  - Добавить в «Дополнительные каталоги библиотек» ссылку на папку:
    - **$(MSMPI_LIB32)**

    Или
    - **$(MSMPI_LIB64)**




ConsoleApplication2 Property Pages
Configuration:
Active(Debug)
Platform:
Active(x64)
Configuration Manager...
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Manifest Tool
XML Document Genera
Browse Information
Build Events
Custom Build Step
Code Analysis
Look for options or switches:
Add Module to Assembly
Additional Dependencies
dbccp32.lib;msmpi.lib;%(AdditionalDependencies)
Additional Library Directories
$(MSMPI_LIB64)
Additional Manifest Dependencies
Additional Options
Allow Isolation
Yes
Assembly Link Resource
Base Address
CLR Image Type
Default image type
Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]
OK
Cancel
Apply

- **Если разрядность программы не соответствует разрядности подключенных библиотек, то возникнут такие ошибки:**

LNK1120: 5 unresolved externals
LNK2019: unresolved external symbol _MPI_Comm_rank@8 referenced in function _main
LNK2019: unresolved external symbol _MPI_Finalize@0 referenced in function _main
LNK2019: unresolved external symbol _MPI_Init@8 referenced in function _main
LNK2019: unresolved external symbol _MPI_Recv@28 referenced in function _main

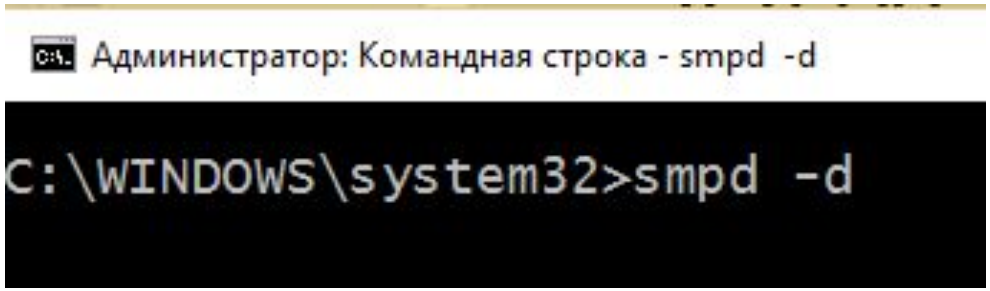
Первое приложение

```
#include "stdafx.h"

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d"
        " out of %d processors\n",
        processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Запуск приложения

- На каждой машине запустить демон:

smpd -d



- На управляющем узле выполнить команду запуска программы:

mpiexec -hosts КоличХостов IP1 КолПроц1 IP2 КолПроц2 … IPN КолПроцN -wdir ПутьКПапкеСПрограммой ИмяФайла.exe

Запуск приложения

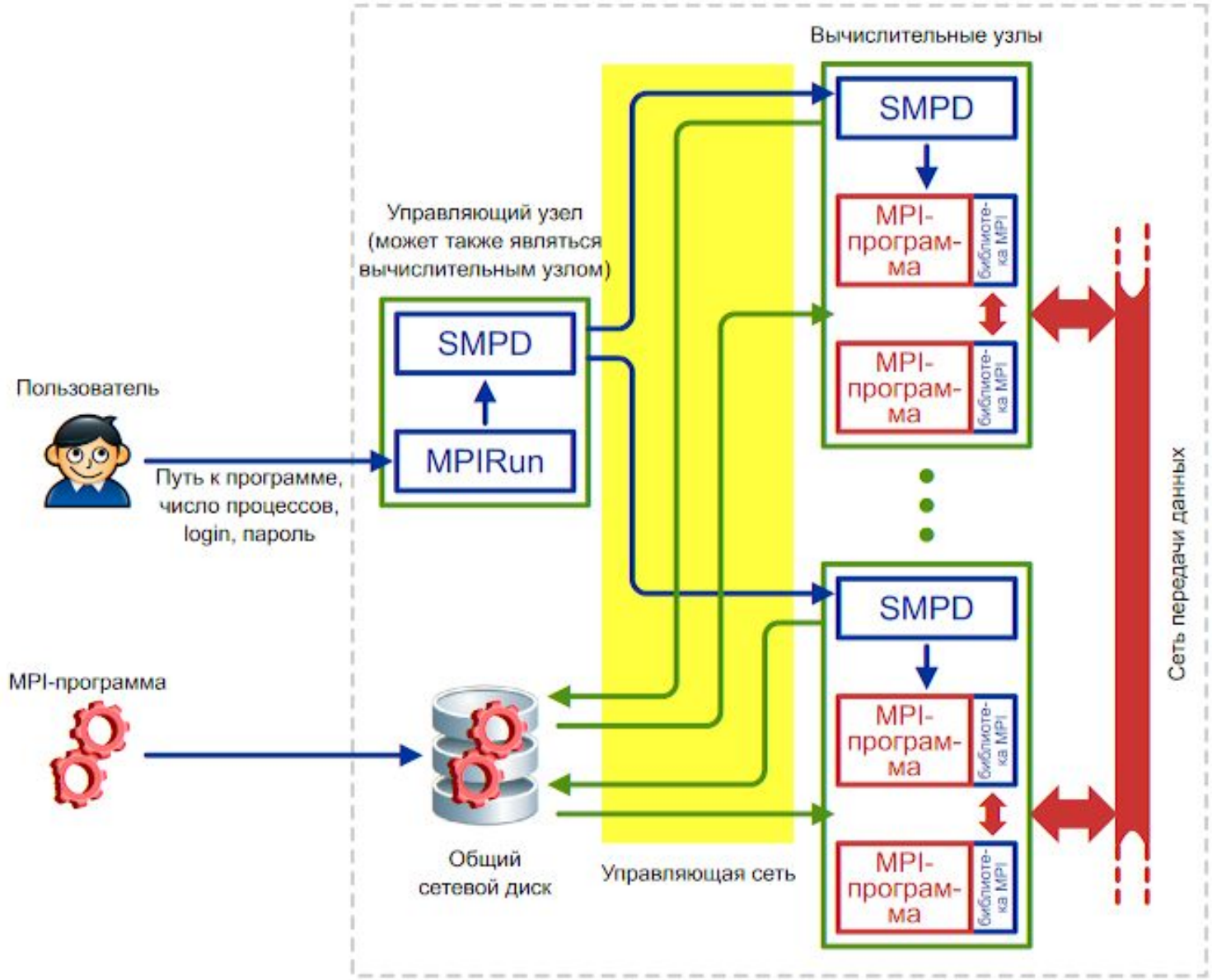
- Пример строки запуска программы на 3-х хостах:

```
mpiexec -hosts 3 192.168.0.2 2 192.168.0.3 2
127.0.0.1 3 -wdir \\192.168.0.1\MPIProgram
Example.exe
```

- Результат:

Hello world from processor one, rank 4 out of 7 processors
Hello world from processor one, rank 3 out of 7 processors
Hello world from processor two, rank 6 out of 7 processors
Hello world from processor two, rank 5 out of 7 processors
Hello world from processor quad, rank 2 out of 7 processors
Hello world from processor quad, rank 0 out of 7 processors
Hello world from processor quad, rank 1 out of 7 processors

Общая схема работы MPICH на кластере



Первое приложение

```
#include "stdafx.h"

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d"
        " out of %d processors\n",
        processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Константы

Тип MPI	Тип Си
MPI_CHAR	char
MPI_BYTE	unsigned char
MPI_SHORT	short
MPI_INT	int
MPI_LONG	long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_LONG_DOUBLE	long double

Функции и константы

- **MPI_COMM_WORLD** – все процессы (константа)
- **int MPI_Init(int* argc, char** argv)**
- **int MPI_Finalize()**
- **int MPI_Comm_size(MPI_Comm comm, int* size)** – определить количество запущенных процессов:

```
int size;
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

- **int MPI_Comm_rank(MPI_comm comm, int* rank)** – определение номера процесса в группе

```
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

- **int MPI_Abort(MPI_Comm comm, int errorcode)** – аварийное завершение работы процессов

```
MPI_Abort(MPI_COMM_WORLD,MPI_ERR_OTHER);
```

Функции

- **`int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm)`** – передача сообщения
  - **buf** - адрес начала буфера посылки сообщения
  - **count** - число передаваемых элементов в сообщении
  - **datatype** - тип передаваемых элементов
  - **dest** - номер процесса-получателя
  - **msgtag** - метка сообщения
  - **comm** - идентификатор группы

```
#define N 10
…
int rank, buf[N];
…
MPI_Send(buf, N, MPI_INT, 1, 10, MPI_COMM_WORLD);
```

Функции

- **`int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_comm comm, MPI_Status *status)`** – прием сообщения (блокирующая функция)
- *выходной параметр* **buf** - адрес начала буфера приема сообщения
- **count** - максимальное число элементов в принимаемом сообщении
- **datatype** - тип элементов принимаемого сообщения
- **source** - номер процесса-отправителя
- **msgtag** - метка принимаемого сообщения
- **comm** - идентификатор группы
- *выходной параметр* **status** - параметры принятого сообщения

```
#define N 10
…
int rank, buf[N];
MPI_Status status;
…
MPI_Recv(buf, N, MPI_INT, 1, 10, MPI_COMM_WORLD, &status);
…
```

Функции

- **`int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status status)`** – проверка приемного буфера
  - **source** - номер процесса-отправителя
  - **tag** - метка сообщения
  - **comm** - идентификатор группы
  - **выходной параметр status** - параметры принятого сообщения

```
MPI_Status status;
…
MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
&status);
```

Структура MPI_Status

Содержит поля:

- MPI_SOURCE (источник),
- MPI_TAG (метка),
- MPI_ERROR (ошибка).

```
MPI_Status status;
…
MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
&status);
```

-

Функции

- **`int MPI_Get_count(MPI_Status status, MPI_Datatype datatype, int *count )`** – определение размера сообщения
  - **status** - информация о сообщении
  - **datatype -** тип принимаемых элементов
  - *выходной параметр* **count** - число элементов сообщения

```
MPI_Status status;
int count;
…
MPI_Get_count(&status, MPI_INT, &count);
```

Простой пример 1

```
const double a = 0.0;//Нижний предел
const double b = 100.0;//Верхний предел
const double h = 0.000001;//Шаг интегрирования

double fnc(double x)//Интегрируемая функция
{
    return x*x;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int myrank, ranksize, i;
    clock_t start, finish;
    MPI_Status status;
    MPI_Init(NULL, NULL);//Инициализация MPI
                         //Определяем свой номер в группе:
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    //Определяем размер группы:
    MPI_Comm_size(MPI_COMM_WORLD, &ranksize);
    double cur_a, cur_b, d_ba, cur_h;
    if (!myrank)
    {//Это процесс-мастер
     //Определяем размер диапазона для каждого процесса:
        d_ba = (b - a) / ranksize;
        //Оставляем первый диапазон для мастера:
        cur_a = a + d_ba;
        cur_h = h;
        start = clock();
```

```
    //Рассылаем исходные данные подчиненным процессам:
    for (i = 1; i<ranksize; i++)
    {
        cur_b = cur_a + d_ba - h;
        MPI_Send(&cur_a, 1, MPI_DOUBLE, i, 98,
            MPI_COMM_WORLD);
        MPI_Send(&cur_b, 1, MPI_DOUBLE, i, 99,
            MPI_COMM_WORLD);
        MPI_Send(&cur_h, 1, MPI_DOUBLE, i, 100,
            MPI_COMM_WORLD);
        cur_a += d_ba;
    }
    cur_a = a;
    cur_b = a + d_ba - h;
}
else
{//Это один из подчиненных процессов
 //Получаем исходные данные:
    MPI_Recv(&cur_a, 1, MPI_DOUBLE, 0, 98,
        MPI_COMM_WORLD, &status);
    MPI_Recv(&cur_b, 1, MPI_DOUBLE, 0, 99,
        MPI_COMM_WORLD, &status);
    MPI_Recv(&cur_h, 1, MPI_DOUBLE, 0, 100,
        MPI_COMM_WORLD, &status);
}
```

```
    //Расчет интеграла в своем диапазоне, выполняют все
    //процессы:
    double s = 0, s1;
    printf("Process %d. A=%.4f B=%.4f h=%.10f\n",
        myrank, cur_a, cur_b, cur_h);

    for (cur_a += cur_h; cur_a <= cur_b; cur_a += cur_h)
        s += cur_h*fnc(cur_a);

    if (!myrank)
    {//Это процесс-мастер
     //Собираем результаты расчетов:
        for (i = 1; i<ranksize; i++)
        {
            MPI_Recv(&s1, 1, MPI_DOUBLE, i, 101,
                MPI_COMM_WORLD, &status);
            s += s1;
        }
        finish = clock();
        //Печать результата:
        printf("Integral value: %.4f\n", s);
        printf("Time: %.4f\n", (double)(finish - start) / CLOCKS_PER_SEC);
    }
    else
        //Это подчиненный процесс, отправляем результаты
        //мастеру:
        MPI_Send(&s, 1, MPI_DOUBLE, 0, 101,
            MPI_COMM_WORLD);
    MPI_Finalize();//Завершение работы с MPI
    return 0;
}
```

Простой пример 2

```
const double a = 0.0;//Нижний предел
const double b = 100.0;//Верхний предел
const double h = 0.000001;//Шаг интегрирования

double fnc(double x)//Интегрируемая функция
{
    return x*x;
}


int _tmain(int argc, _TCHAR* argv[])
{
    int myrank, ranksize, i;
    clock_t start, finish;

    MPI_Init(&argc, &argv);//Инициализация MPI
                           //Определяем свой номер в группе:
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    //Определяем размер группы:
    MPI_Comm_size(MPI_COMM_WORLD, &ranksize);
    double cur_a, cur_b, d_ba, cur_h;
    double *sbuf = NULL;
    if (!myrank)
    {//Это процесс-мастер
     //Определяем размер диапазона для каждого процесса:
        d_ba = (b - a) / ranksize;
        sbuf = new double[ranksize * 3];
        cur_a = a;
        cur_h = h;
```

```
    for (i = 0; i<ranksize; i++)
    {
        cur_b = cur_a + d_ba - h;
        sbuf[i * 3] = cur_a;
        sbuf[i * 3 + 1] = cur_b;
        sbuf[i * 3 + 2] = h;
        cur_a += d_ba;
    }
}
double rbuf[3];

start = clock();
//Рассылка всем процессам, включая процесс-мастер
//начальных данных для расчета:
MPI_Scatter(sbuf, 3, MPI_DOUBLE, rbuf, 3, MPI_DOUBLE, 0,
    MPI_COMM_WORLD);
if (sbuf) delete[]sbuf;
cur_a = rbuf[0]; cur_b = rbuf[1]; cur_h = rbuf[2];
//Расчет интеграла в своем диапазоне, выполняют все
//процессы:
double s = 0;
printf("Process %d. A=%.4f B=%.4f h=%.10f\n",
    myrank, cur_a, cur_b, cur_h);
for (cur_a += cur_h; cur_a <= cur_b; cur_a += cur_h)
    s += cur_h*fnc(cur_a);
rbuf[0] = s;
if (!myrank) sbuf = new double[ranksize];
//Собираем значения интегралов от процессов:
MPI_Gather(rbuf, 1, MPI_DOUBLE, sbuf, 1, MPI_DOUBLE, 0,
    MPI_COMM_WORLD);
```

```
    if (!myrank)
    {//Это процесс-мастер
     //Суммирование интегралов, полученных каждым
     //процессом:
        for (i = 0, s = 0; i<ranksize; i++) s += sbuf[i];
        finish = clock();
        //Печать результата:
        printf("Integral value: %.4f\n", s);
        printf("Time: %.4f\n", (double)(finish - start) / CLOCKS_PER_SEC);

        delete[]sbuf;
    }
    MPI_Finalize();//Завершение работы с MPI

    return 0;
}
```