ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Волгоградский государственный технический университет

Кафедра электронно-вычислительных машин и систем

Методические указания

по выполнению лабораторной работы «Основы MPI. Запуск задач на кластере»

по курсу «Параллельное программирование»

Составители: доц. Андреев А.Е.,
Шаповалов О.В.

Волгоград, 2010

Введение

В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга. Для организации параллельных вычислений в таких условиях необходимо иметь возможность распределять вычислительную нагрузку и организовать информационное взаимодействие (передачу данных) между процессорами.

Решение всех перечисленных вопросов и обеспечивает интерфейс передачи данных (message passing interface - MPI).

В общем плане, для распределения вычислений между процессорами необходимо проанализировать алгоритм решения задачи, выделить информационно независимые фрагменты вычислений, провести их программную реализацию и затем разместить полученные части программы на разных процессорах. В рамках MPI принят более простой подход - для решения поставленной задачи разрабатывается одна программа и эта единственная программа запускается одновременно на выполнение на всех имеющихся процессорах. При этом для того, чтобы избежать идентичности вычислений на разных процессорах, можно, во-первых, подставлять разные данные для программы на разных процессорах, а во-вторых, в MPI имеются средства для идентификации процессора, на котором выполняется программа (и тем самым, предоставляется возможность организовать различия в вычислениях в зависимости от используемого программой процессора).

Подобный способ организации параллельных вычислений получил наименование модели "одна программа множество процессов" (single program multiple processes or SPMP).

Основы MPI

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться на разных процессорах, но на одном процессоре могут располагаться и несколько процессов (в этом случае их исполнение осуществляется в режиме разделения времени). В предельном случае для выполнения параллельной программы может использоваться один процессор - как правило, такой способ применяется для начальной проверки правильности параллельной программы.

Каждый процесс параллельной программы порождается на основе копии одного и того же программного кода (модель SPMP). Данный программный код, представленный в виде исполняемой программы, должен быть доступен в момент запуска параллельной программы на всех

используемых процессорах. Исходный программный код для исполняемой программы разрабатывается на алгоритмических языках C или Fortran с использованием той или иной реализации библиотеки MPI.

Количество процессов и число используемых процессоров определяется в момент запуска параллельной программы средствами среды исполнения MPI-программ и в ходе вычислений меняться не может. Все процессы программы последовательно перенумерованы от 0 до np-1, где np есть общее количество процессов. Номер процесса именуется рангом процесса.

Операции передачи данных

Основу MPI составляют операции передачи сообщений. Среди предусмотренных в составе MPI функций различаются парные (point-to-point) операции между двумя процессами и коллективные (collective) коммуникационные действия для одновременного взаимодействия нескольких процессов.

Для выполнения парных операций могут использоваться разные режимы передачи, среди которых синхронный, блокирующий и др. - полное рассмотрение возможных режимов передачи будет выполнено в лабораторной работе №4.

Понятие коммуникаторов

Процессы параллельной программы объединяются в группы. Под коммуникатором в MPI понимается специально создаваемый служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи данных.

Как правило, парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору. Коллективные операции применяются одновременно для всех процессов коммуникатора. Указание используемого коммуникатора является обязательным для операций передачи данных в MPI.

В ходе вычислений могут создаваться новые и удаляться существующие группы процессов и коммуникаторы. Один и тот же процесс может принадлежать разным группам и коммуникаторам. Все имеющиеся в

параллельной программе процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI_COMM_WORLD.

При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (intercommunicator).

Типы данных

При выполнении операций передачи сообщений для указания передаваемых или получаемых данных в функциях MPI необходимо указывать тип пересылаемых данных. MPI содержит большой набор базовых типов данных, во многом совпадающих с типами данных в алгоритмических языках C и Fortran. Кроме того, в MPI имеются возможности для создания новых производных типов данных для более точного и краткого описания содержимого пересылаемых сообщений.

Подробное рассмотрение возможностей MPI для работы с производными типами данных будет выполнено в лабораторной работе №4.

Введение в разработку параллельных программ с использованием MPI

Первой вызываемой функцией MPI должна быть функция:

```
int MPI_Init ( int *agrc, char ***argv )
```

для инициализации среды выполнения MPI-программы. Параметрами функции являются количество аргументов в командной строке и текст самой командной строки.

Последней вызываемой функцией MPI обязательно должна являться функция:

```
int MPI_Finalize (void)
```

Можно отметить, что структура параллельной программы, разработанная с использованием MPI, должна иметь следующий вид:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  <программный код без использования MPI функций>
  MPI_Init ( &agrc, &argv );
    <программный код с использованием MPI функций >
  MPI_Finalize();
  <программный код без использования MPI функций >
```

```
  return 0;
}
```

Следует отметить:

1. Файл mpi.h содержит определения именованных констант, прототипов функций и типов данных библиотеки MPI;
2. Функции MPI_Init и MPI_Finalize являются обязательными и должны быть выполнены (и только один раз) каждым процессом параллельной программы.

Определение количества и ранга процессов

Определение количества процессов в выполняемой параллельной программе осуществляется при помощи функции:

```
int MPI_Comm_size ( MPI_Comm comm, int *size ).
```

Для определения ранга (номера) процесса используется функция:

```
int MPI_Comm_rank ( MPI_Comm comm, int *rank ).
```

Как правило, вызов функций MPI_Comm_size и MPI_Comm_rank выполняется сразу после MPI_Init:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  int ProcNum, ProcRank;
  <программный код без использования MPI функций>
  MPI_Init ( &agrc, &argv );
  MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
    <программный код с использованием MPI функций >
  MPI_Finalize();
  <программный код без использования MPI функций >
  return 0;
}
```

Следует отметить:

1. Коммуникатор MPI_COMM_WORLD, как отмечалось ранее, создается по умолчанию и представляет все процессы выполняемой параллельной программы;
2. Ранг, получаемый при помощи функции MPI_Comm_rank, является рангом процесса, выполнившего вызов этой функции, т.е. переменная ProcRank будет принимать различные значения в разных процессах.

Передача сообщений

Для передачи сообщения процесс-отправитель должен выполнить функцию:

```
int MPI_Send(void *buf, int count, MPI_Datatype type, int dest,
  int tag, MPI_Comm comm), где
```

buf - адрес буфера памяти, в котором располагаются данные отправляемого сообщения;

count - количество элементов данных в сообщении;

type - тип элементов данных пересылаемого сообщения;

dest - ранг процесса, которому отправляется сообщение;

tag - значение-тег, используемое для идентификации сообщений;

comm - коммуникатор, в рамках которого выполняется передача данных.

Для указания типа пересылаемых данных в MPI имеется ряд базовых типов, полный список которых приведен в таблице 1.

Таблица 1. Базовые (предопределенные) типы данных MPI для алгоритмического языка C

MPI_Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short

MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Следует отметить:

1. Процессы, между которыми выполняется передача данных, в обязательном порядке должны принадлежать коммуникатору, указываемому в функции MPI_Send;
2. Параметр tag используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное целое число (см. также описание функции MPI_Recv);
3. завершение функции MPI_Send означает лишь, что операция передачи начала выполняться и пересылка сообщения будет рано или поздно будет выполнена.

Прием сообщений

Для приема сообщения процесс-получатель должен выполнить функцию:

```
int MPI_Recv(void *buf, int count, MPI_Datatype type, int source, int tag, MPI_Comm comm, MPI_Status *status),
```
где

buf - адрес буфера памяти, в который принимается сообщение;

count - количество элементов данных в сообщении;

type - тип элементов данных принимаемого сообщения;

source - ранг процесса, от которого должен быть выполнен прием сообщения;

tag - тег сообщения, которое должно быть принято для процесса;

comm - коммуникатор, в рамках которого выполняется передача данных;

status - указатель на структуру данных с информацией о результате выполнения операции приема данных.

Следует отметить:

1. Буфер памяти должен быть достаточным для приема сообщения, а тип элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения,
2. При необходимости приема сообщения от любого процесса-отправителя для параметра source может быть указано значение MPI_ANY_SOURCE,
3. При необходимости приема сообщения с любым тегом для параметра tag может быть указано значение MPI_ANY_TAG,
4. Параметр status позволяет определить ряд характеристик принятого сообщения:
    - status.MPI_SOURCE - ранг процесса-отправителя принятого сообщения;
    - status.MPI_TAG - тег принятого сообщения.

По завершении функции MPI_Recv в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция MPI_Recv является блокирующей для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если по каким-то причинам ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.

Первая параллельная программа с использованием MPI

Рассмотренный набор функций оказывается достаточным для разработки параллельных программ. Приводимая ниже программа является стандартным начальным примером для алгоритмического языка С.

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[]){
  int ProcNum, ProcRank, RecvRank;
  MPI_Status Status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  if ( ProcRank == 0 ){
    // Действия, выполняемые только процессом с рангом 0
    printf ("\n Hello from process %3d", ProcRank);
    for ( int i=1; i<ProcNum; i++ ) {
      MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
        MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
      printf("\n Hello from process %3d", RecvRank);
    }
  }
  else // Сообщение, отправляемое всеми процессами,
```

```
        // кроме процесса с рангом 0
      MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
  }
```

Пример 1. Первая параллельная программа с использованием MPI

Как следует из текста программы, каждый процесс определяет свой ранг, после чего действия в программе разделяются. Все процессы, кроме процесса с рангом 0, передают значение своего ранга нулевому процессу. Процесс с рангом 0 сначала печатает значение своего ранга, а далее последовательно принимает сообщения с рангами процессов и также печатает их значения. При этом важно отметить, что порядок приема сообщений заранее не определен и зависит от условий выполнения параллельной программы (более того, этот порядок может изменяться от запуска к запуску). Так, возможный вариант результатов печати процесса 0 может состоять в следующем (для параллельной программы из четырех процессов):

```
Hello from process 0
Hello from process 2
Hello from process 1
Hello from process 3
```

Такой "плавающий" вид получаемых результатов существенным образом усложняет разработку, тестирование и отладку параллельных программ, т.к. в этом случае исчезает один из основных принципов программирования - повторяемость выполняемых вычислительных экспериментов. Как правило, если это не приводит к потере эффективности, следует обеспечивать однозначность расчетов и при использовании параллельных вычислений. Так, для рассматриваемого простого примера можно восстановить постоянство получаемых результатов при помощи задания ранга процесса-отправителя в операции приема сообщения:

```
  MPI_Recv(&RecvRank, 1, MPI_INT, i, MPI_ANY_TAG,
MPI_COMM_WORLD, &Status).
```

Указание ранга процесса-отправителя регламентирует порядок приема сообщений, и, как результат, строки печати будут появляться строго в порядке возрастания рангов процессов (такая регламентация в отдельных ситуациях может приводить к замедлению выполняемых параллельных вычислений).

Следует отметить еще один важный момент - разрабатываемая с использованием MPI программа, как в данном частном варианте, так и в самом общем случае используется для порождения всех процессов параллельной программы и, как результат, должна определять вычисления,

выполняемые во всех этих процессах. Можно сказать, что MPI-программа является некоторым "макро-кодом", различные части которого используются разными процессами.

Для разделения фрагментов кода между процессами обычно используется подход, примененный в только что рассмотренной программе - при помощи функции MPI_Comm_rank определяется ранг процесса, а затем в соответствии с рангом выделяются необходимые для процесса участки программного кода. Наличие в одной и той же программе фрагментов кода разных процессов также значительно усложняет понимание и, в целом, разработку MPI-программы. Как результат, можно рекомендовать при увеличении объема разрабатываемых программ выносить программный код разных процессов в отдельные программные модули (функции). Общая схема MPI программы в этом случае будет иметь вид:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoProcess0();
else if ( ProcRank == 1 ) DoProcess1();
else if ( ProcRank == 2 ) DoProcess2();
```

Во многих случаях, как и в рассмотренном примере, выполняемые действия являются отличающимися только для процесса с рангом 0. В этом случае общая схема MPI программы принимает более простой вид:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoManagerProcess();
else DoWorkerProcesses();
```

Все функции MPI возвращают в качестве своего значения код завершения. При успешном выполнении функции возвращаемый код равен MPI_SUCCESS. Другие значения кода завершения свидетельствуют об обнаружении тех или иных ошибочных ситуаций в ходе выполнения функций. Для выяснения типа обнаруженной ошибки используются предопределенные именованные константы, среди которых:

- MPI_ERR_BUFFER - неправильный указатель на буфер;
- MPI_ERR_COMM - неправильный коммуникатор;
- MPI_ERR_RANK - неправильный ранг процесса

и др. - полный список констант для проверки кода завершения содержится в файле mpi.h.

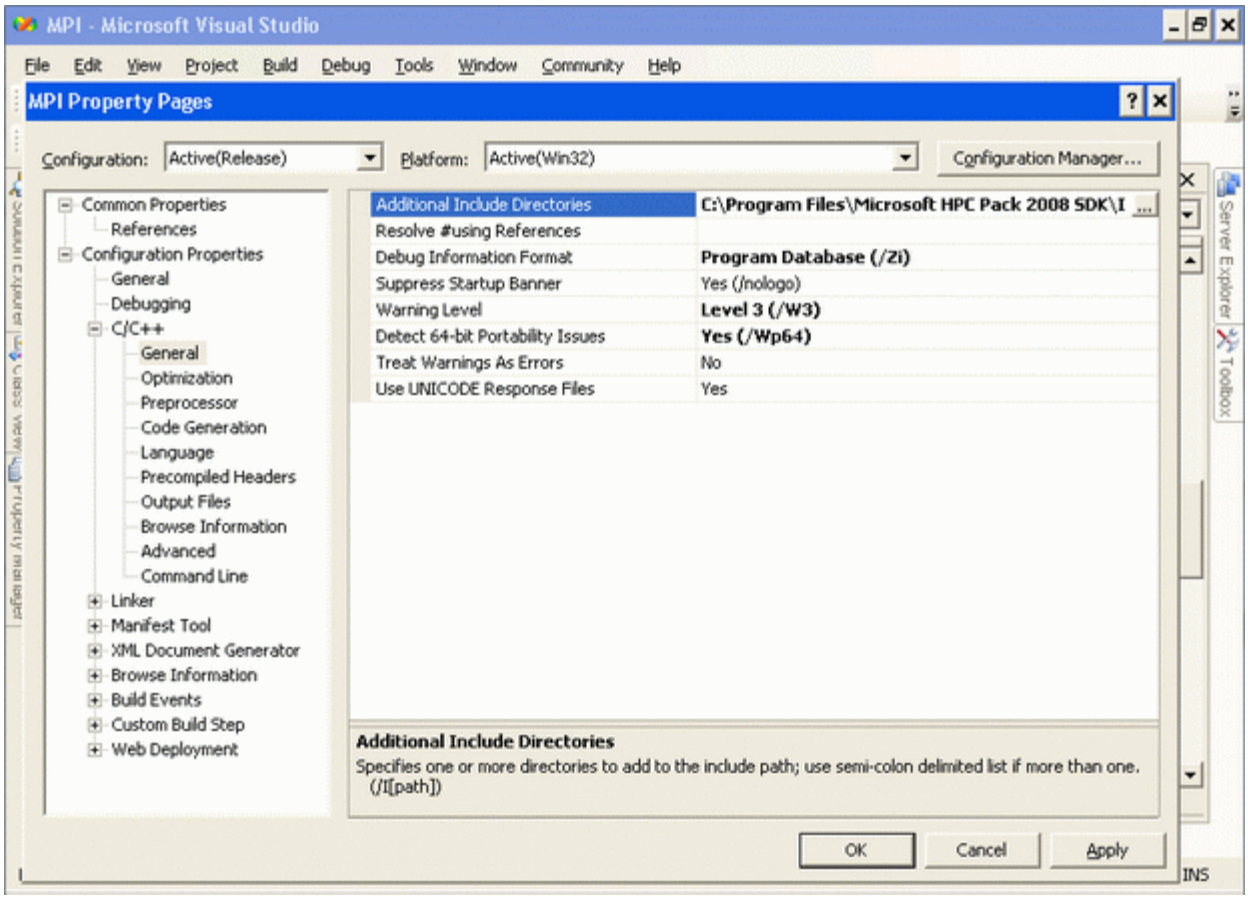
Настройка проекта в Visual Studio 2008 для использования MS MPI

В данном задании будут рассмотрены вопросы использования Visual Studio 2008 для компиляции параллельной MPI программы в среде MS MPI:

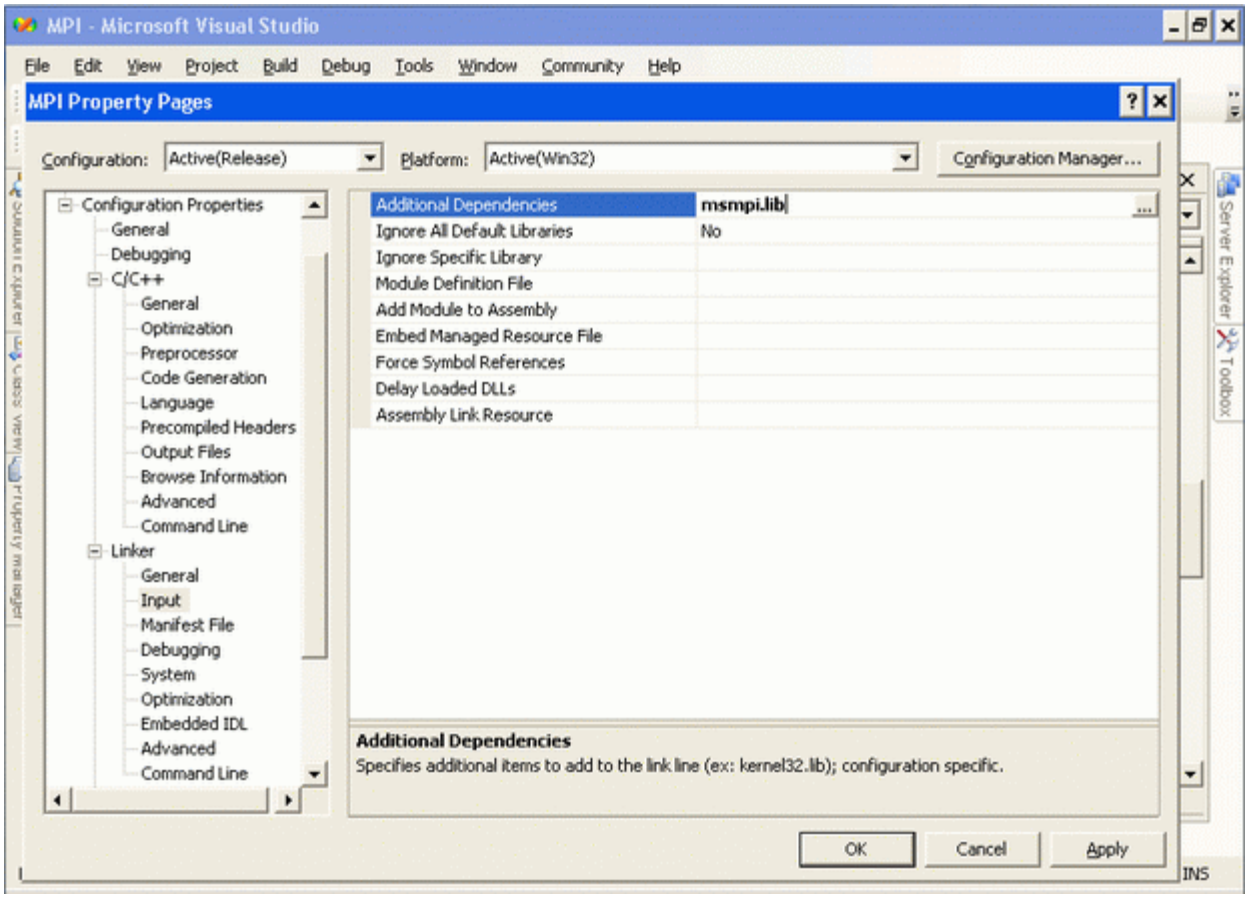
- Запустите Microsoft Visual Studio 2008
- Создайте новый проект: выберите пункт меню File->New->Project. В окне выбора нового проекта выберите консольное Win32 приложение (Visual C++->Win32->Win32 Console Application), введите имя проекта в поле "Name" (например, "mpi") и убедитесь, что путь до проекта выбран правильно (поле "Location"). Нажмите кнопку "OK" для выбора остальных настроек создаваемого проекта.
- В открывшемся окне нажмите кнопку "Next".
- В открывшемся окне выберите настройки проекта (можно оставить все настройки по умолчанию). Нажмите кнопку "Finish".

Для того, чтобы скомпилировать Вашу программу для использования в среде MS MPI, необходимо изменить следующие настройки проекта Microsoft Visual Studio 2008:

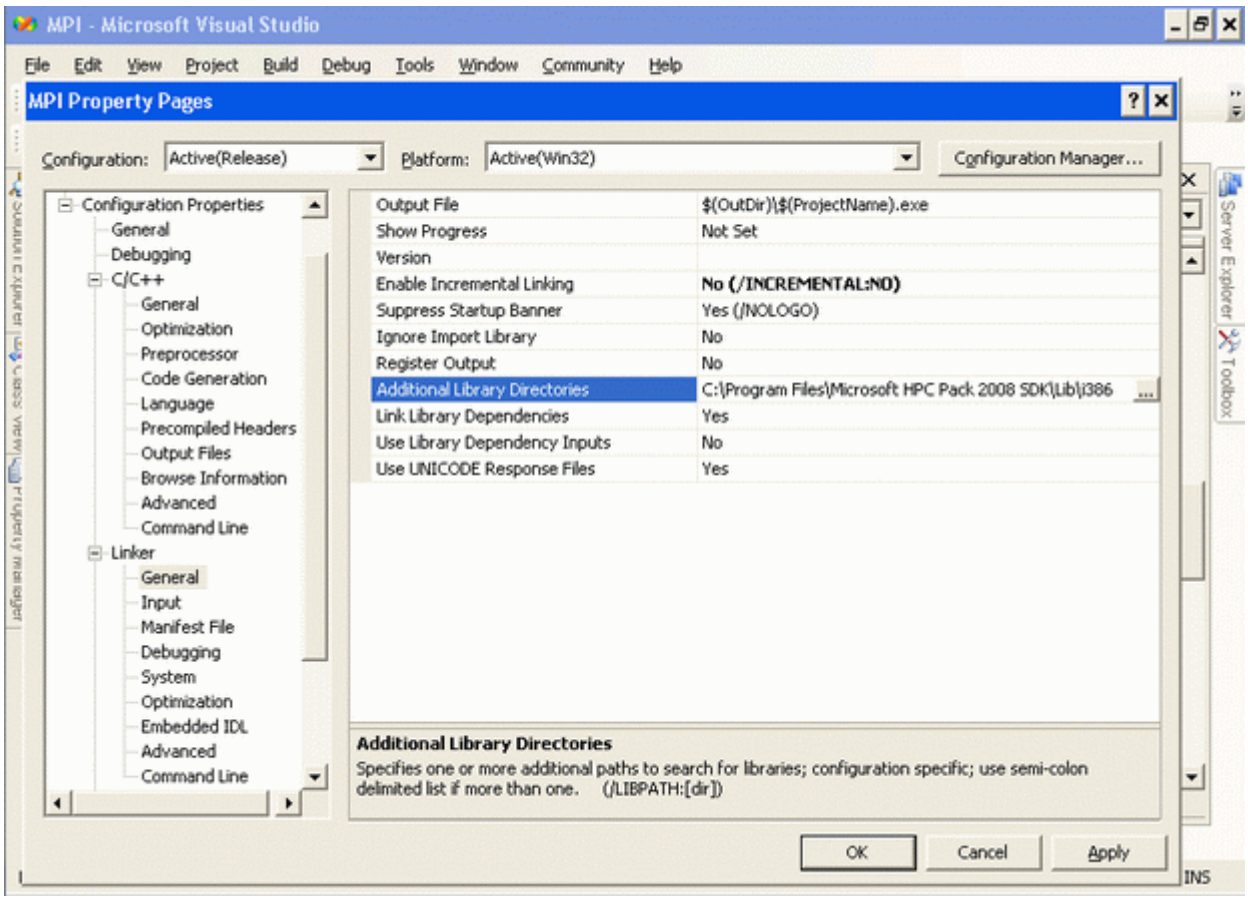
- Путь до заголовочных файлов объявлений MPI. Выберите пункт меню Project->Project Properties. В пункте Configuration Properties->C++->General->Additional Include Directories введите путь до заголовочных файлов MS MPI: <Директория установки HPC Pack 2008 SDK>\Include.



- Библиотечный файл с реализацией функций MPI. Выберите пункт меню Project->Project Properties. В пункте Configuration Properties->C++->Linker->Input->Additional Dependencies введите название библиотечного файла msmpi.lib.



- Путь до библиотечного файла msmpi.lib. Выберите пункт меню Project->Project Properties. В пункте Configuration Properties->C++->Linker->General->Additional Library Directories введите путь до библиотечного файла msmpi.lib: < Директория установки HPC Pack 2008 SDK >\Lib\i386 или < Директория установки HPC Pack 2008 SDK >\Lib\AMD64 в зависимости от используемой Вами архитектуры процессоров.



Задание 1.

При выполнении задания следует считать, что только главный процесс генерирует начальные данные для обработки, и результаты работы программы должны быть собраны, опять же, на главном процессе. То есть программа должна строиться по следующему шаблону:

- главный процесс генерирует начальные данные и рассылает их всем процессам (каждому процессу – только необходимую ему часть);

- каждый процесс обрабатывает свою часть данных;

- главный процесс собирает данные-результат со всех процессов.

Ниже представлена программа для сложения двух векторов целых чисел:

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[]){
  int ProcNum, ProcRank, RecvRank;
  MPI_Status Status;
```

```
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    int N=1000;
    int *a = new int[N];
    int *b = new int[N];
    int *c = new int[N];

    if ( ProcRank == 0 )
    {
      for(int i=0;i<N;i++)
       a[i]=b[i]=i;
    }

// рассылка данных по процессам
    if ( ProcRank == 0 )
    {
      for ( int i=1; i<ProcNum; i++ )
        {
          MPI_Send(&a[N*i/ProcNum],N/ProcNum,
                          MPI_INT,i,0,MPI_COMM_WORLD);
          MPI_Send(&b[N*i/ProcNum],N/ProcNum,
                          MPI_INT,i,0,MPI_COMM_WORLD);
        }
    }
    else
      {
         MPI_Recv(&a[N*ProcRank/ProcNum], N/ProcNum, MPI_INT, 0,
          MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
         MPI_Recv(&a[N*ProcRank/ProcNum], N/ProcNum, MPI_INT, 0,
          MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
      }

//собственно расчет
     for(int i= N*ProcRank/ProcNum;i<N*(ProcRank+1)/ProcNum;i++)
       c[i]=a[i]+b[i];


// сбор данных на главном процессе
    if ( ProcRank == 0 )
    {
      for ( int i=1; i<ProcNum; i++ )
        {
         MPI_Recv(&c[N*i/ProcNum], N/ProcNum, MPI_INT, i,
          MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
        }
    }
    else
      {
          MPI_Send(&c[N*ProcRank/ProcNum],N/ProcNum,
                   MPI_INT,0,0,MPI_COMM_WORLD);
      }
```

```
    MPI_Finalize();
    return 0;
  }
```

Пример 2. Вторая параллельная программа с использованием MPI

Напишите параллельную программу (по вариантам).

1. для нахождения максимального элемента в одномерном массиве целых чисел;
2. для умножения двух квадратных матриц;
3. для нахождения минимального элемента в одномерном массиве целых чисел;
4. для умножения матрицы на вектор;
5. для нахождения максимального элемента в одномерном массиве вещественных чисел одинарной точности;
6. для нахождения суммы одномерного массива вещественных чисел одинарной точности;
7. для нахождения минимального элемента в одномерном массиве вещественных чисел двойной точности.

Работа с Microsoft High Performance Computing Server 2008

Для эффективного использования вычислительного ресурса кластера необходимо обеспечить не только непосредственный механизм запуска заданий на выполнение, но и предоставить среду управления ходом выполнения заданий, решающую, в том числе, задачу эффективного распределения ресурсов. Эти задачи эффективно решаются с использованием встроенных в HPC 2008 средств.

Дадим определение важнейшим понятиям, используемым в HPC 2008:

Задание (job) - запрос на выделение вычислительного ресурсов кластера для выполнения задач. Каждое задание может содержать одну или несколько задач.

Задача (task) - команда или программа (в том числе, параллельная), которая должна быть выполнена на кластере. Задача не может существовать вне некоторого задания, при этом задание может содержать как несколько задач, так и одну.

Планировщик заданий (job scheduler) - сервис, отвечающий за поддержание очереди заданий, выделение системных ресурсов, постанову задач на выполнение, отслеживание состояния запущенных задач.

Узел (node) - вычислительный компьютер, включенный в кластер под управлением HPC 2008.

Сокет (socket) - один из нескольких вычислительных устройств (процессоров) узла. Под количеством сокетов понимается количество устройств физически вставленных в материнскую плату.

Ядро (core) – одно ядро процессора.

Очередь (queue) - список заданий, отправленных планировщику для выполнения на кластере. Порядок выполнения заданий определяется принятой на кластере политикой планирования.

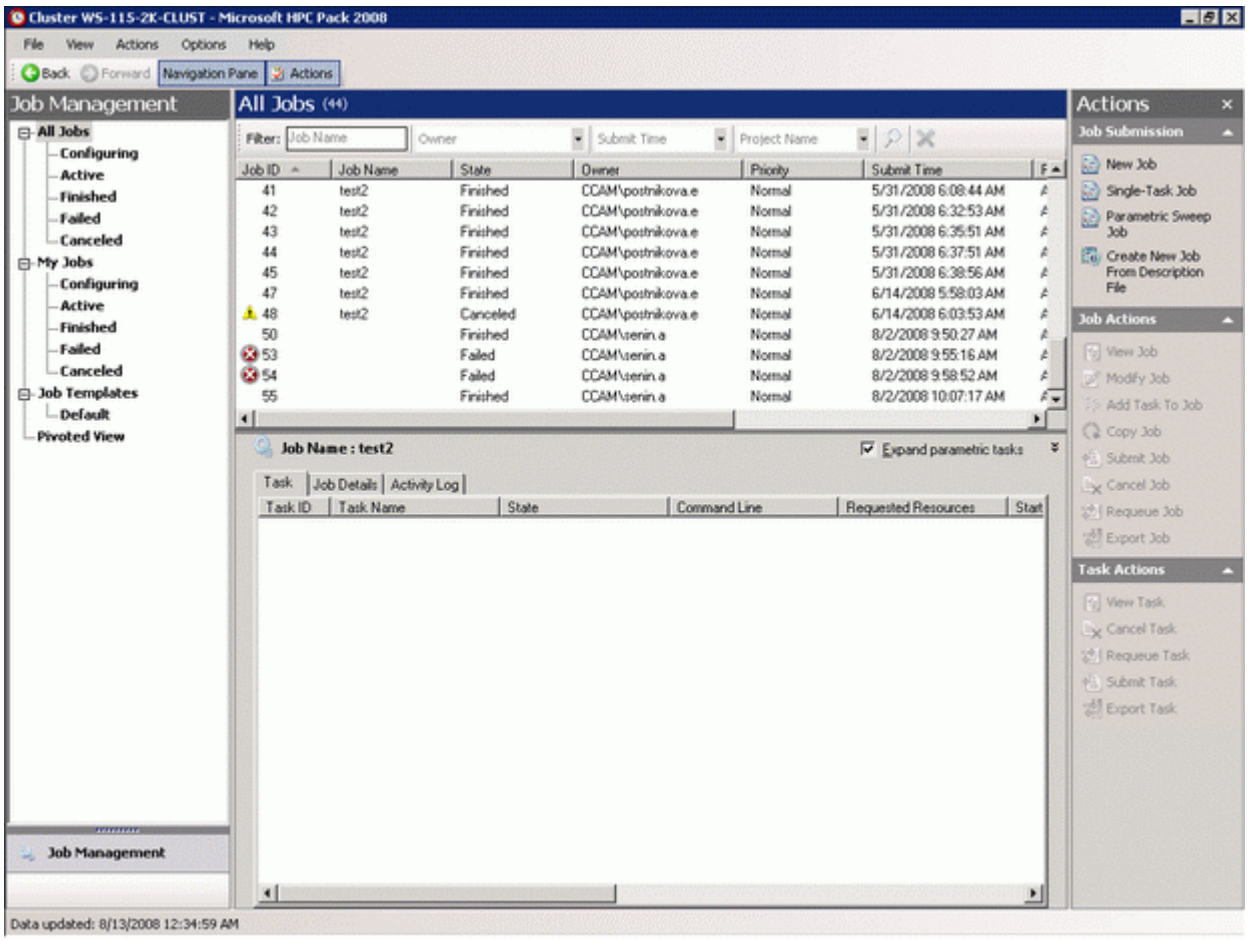
Список задач (task list) - эквивалент очереди заданий для задач каждого конкретного задания.

Для исполнения программы в HPC 2008 необходимо выполнить следующие действия:

- Создать задание с описанием вычислительных ресурсов, необходимых для его выполнения.
- Создать задачу. Параллельная задача описывается при помощи команды mpiexec.exe с соответствующими параметрами (список узлов для ее запуска, имя параллельной программы, аргументы командной строки программы и др.).
- Добавить задачу к созданному ранее заданию.

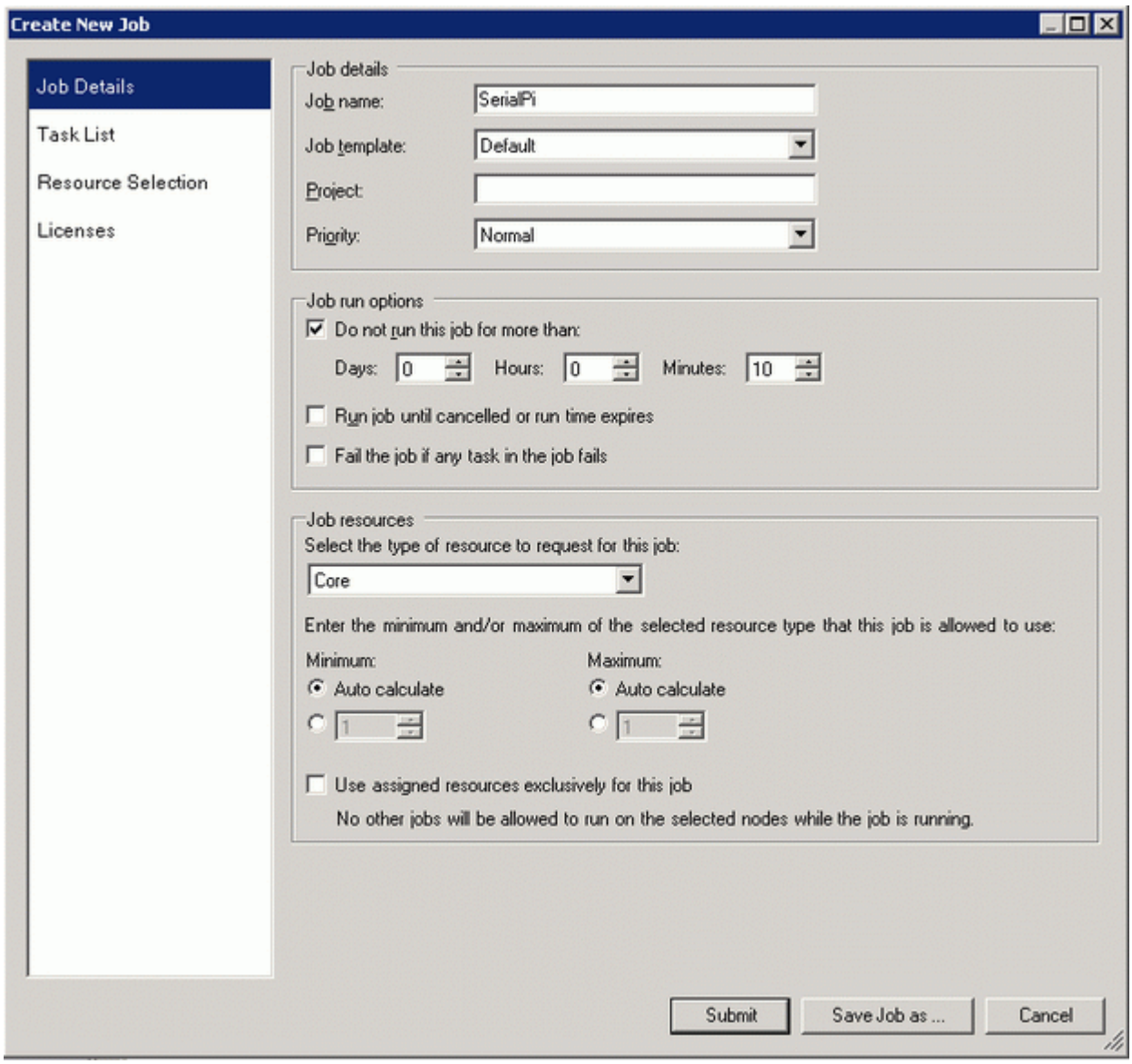
Запуск задачи в HPC 2008

- Откройте HPC Job Manager (Start->All Programs->Microsoft HPC Pack->HPC Job Manager) для запуска программы на кластере.



В открывшемся окне менеджера заданий выберите пункт меню Actions->Job Submission->New Job для постановки нового задания в очередь.

- После этого следует указать максимальные и минимальные требования задания к вычислительным ресурсам. Для этого следует указать тип ресурса (ядро, сокет или узел) и выбрать желаемые минимальное и максимальное значение. Выполнение задания не начнется, пока свободно меньше минимального числа соответствующих ресурсов.
    - Введите имя задания (на ваш выбор)
    - Укажите следующие параметры запуска:
        - тип ресурса – ядра
        - минимальное значение – 4
        - максимальное значение – 4



- На вкладке "Task List" необходимо добавить задачи в задание. Список задач представлен в верхней части экрана (сначала он пуст).
- Для добавления новой задачи к заданию следует нажать кнопку Add и ввести соответствующие настройки в открывшемся диалоге:
  - имя задачи (поле "Task Name") - на ваш выбор
  - команду, которую необходимо выполнить (поле "Command Line"). Запуск задач, разработанных для MS MPI, необходимо осуществлять с использованием специальной утилиты mpiexec.exe, которая принимает в качестве параметров имя параллельной программы и параметры запускаемой программы. Пример команды для запуска параллельной программы: "mpiexec.exe mpi.exe".
  - В поле "Working directory" укажите каталог, который будет использоваться как рабочий для запускаемой программы (\\node13\shared).
  - В полях "Standard input", "Standard output" и "Standard error" укажите файлы, которые будут использоваться как стандартный поток ввода и стандартный поток вывода соответственно.

    - Укажите в качестве минимального и максимального количества вычислительных ресурсов значение 4 (тип вычислительного ресурса выбирался при задании параметров задания).
    - Нажмите кнопку "Save" для добавления задачи в задание.
- При выделении задачи, в нижней части экрана появляются свойства задачи, доступные для редактирования:
    - Exclusive - опция определяет, может ли данная задача разделять вычислительные ресурсы с другими задачами.
    - Rerunnable - опция определяет, должен ли планировщик попытаться перезапустить задачу в случае ее падения.
    - Run Time - жесткий лимит времени исполнения задачи.
    - EnvironmentVariables - опция позволяет задать переменные окружения, которые будут доступны запускаемой задаче (помимо стандартных переменных окружения).
    - Required Nodes - данная опция позволяет отметить узлы, на которых должна быть запущена задача. Все из этих узлов должны быть выделены задаче для начала ее выполнения.
    - Number of Cores (Sockets, Nodes) - минимальное и максимальное число вычислительных ресурсов (ядро, сокет или узел), необходимых для задачи.
    - Task Name - имя задачи.
    - Command Line - команда, которая будет выполнена при старте задачи.
    - Working Directory - рабочая директория задачи.
    - Standard Input - путь до файла, который будет использоваться как стандартный порт ввода для задачи.
    - Standard Output - путь до файла, который будет использоваться как стандартный порт вывода для задачи.
    - Standard Error - путь до файла, который будет использоваться как стандартный порт вывода для ошибок.
- Перейдите на вкладку "Resource Selection" для выбора узлов, на которых будет запущено задание.
    - Поставьте флаг "Run this job only on nodes in the following list" и выберите из списка тот узел, на котором вы работаете.
- Нажмите кнопку "Submit" для добавления задания в очередь.
- Введите имя и пароль пользователя, имеющего право запуска задач на кластере, и нажмите "OK".
- Задание появится в очереди. По окончании выполнения его состояние изменится на "Finished".
- В файле, указанном в настройках задачи для перенаправления стандартного потока вывода, содержится результат работы программы.

Повторный запуск задачи

Если в будущем Вы захотите еще раз запустить задачу (например, с другими параметрами), то Вам будет полезна функция сохранения всех параметров задания, запущенного ранее, в xml файл с возможностью последующего быстрого создания копии:

- Откройте Microsoft HPC Job Manager (Start->All Programs->Microsoft HPC Pack->HPC Job Manager) и выделите задание, которое Вы хотите сохранить в xml-файл. В правой части окна станет активна кнопка Export Job. Нажмите ее для выбора xml-файла, куда произойдет сохранение.
- Откройте сохраненную ранее задачу и запустите на выполнение (<Главное меню>->Actions->Job Submision->Create New Job from Description File).

Контрольные вопросы

1. Дайте определение MPI.
2. Объясните понятие SPMP.
3. Какова базовая структура параллельной программы с использованием MPI?
4. Дайте определения терминам задание (job) и задача (task). В чем основные отличия?
5. Какие основные настройки Microsoft Visual Studio 2008 необходимо произвести при компиляции параллельной программы для использования в среде MS MPI?
6. В чем особенность запуска параллельных задач (скомпилированных для MS MPI) на кластере?