«АРХИТЕКТУРА СОВРЕМЕННОГО КОМПЬЮТЕРА: КОНЦЕПЦИИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ»

Основные модели параллельного выполнения программы на многопроцессорных компьютерах:

Модель передачи сообщений MPI

Модель параллелизма по данным HPF

Модель с общей памятью OpenMP.

Модель передачи сообщений. MPI

Обобщение и стандартизация различных библиотек передачи сообщений привели в 1993 году к разработке стандарта MPI (Message Passing Interface).

Взаимодействие процессов - обмен данными и синхронизация - осуществляется посредством передачи сообщений.

Параллельная программа представляет собой множество процессов, каждый из которых имеет собственное локальное адресное пространство

Его широкое внедрение в последующие годы обеспечило коренной перелом в решении проблемы переносимости параллельных программ, разрабатываемых в рамках разных подходов, использующих модель передачи сообщений в качестве модели выполнения.

Основные достоинства MPI

(по сравнению с интерфейсами других коммуникационных библиотек)

- Возможность использования в языках Фортран, C, C++;
- Предоставление возможностей для совмещения обменов сообщениями и вычислений;
- Предоставление режимов передачи сообщений, позволяющих избежать излишнего копирования информации для буферизации;
- Широкий набор коллективных операций (например, сбор информации с разных процессоров), допускающих эффективную реализацию
- Широкий набор редукционных операций (например, суммирование расположенных на разных процессорах данных, или нахождение их максимальных или минимальных значений), упрощающих работу программиста и допускающих эффективную реализацию
- Удобные средства именования адресатов сообщений, упрощающие разработку стандартных программ или разделение программы на функциональные блоки;
- Возможность задания типа передаваемой информации, что позволяет обеспечить ее автоматическое преобразование в случае различий в представлении данных на разных узлах системы.

Недостатки MPI

- слишком громоздкий и сложный интерфейс для прикладного программиста, а также для реализации

В настоящее время практически не существует реализаций MPI, в которых в полной мере обеспечивается совмещение обменов с вычислениями.

MPI-2

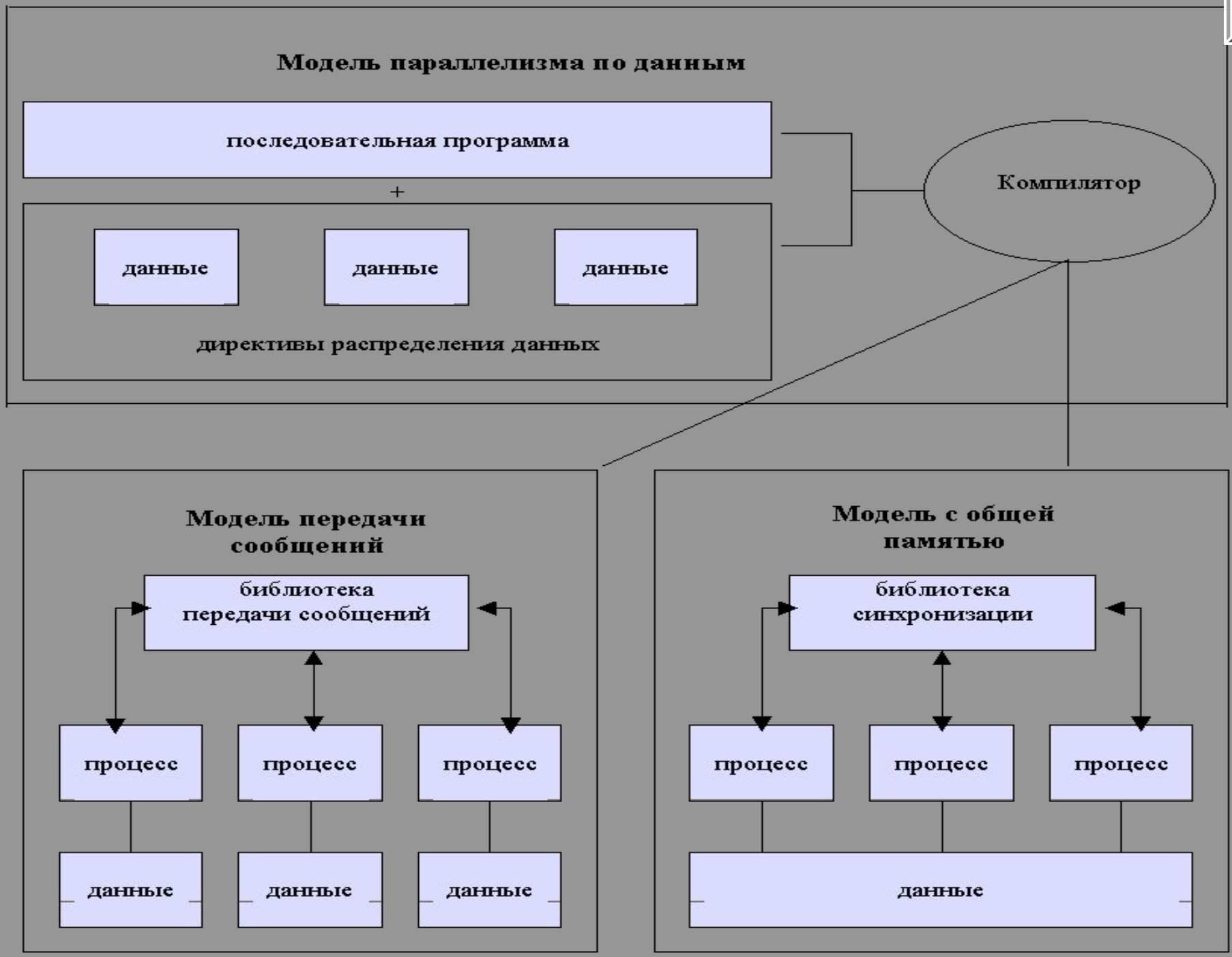
Появившийся в 1997 проект стандарта MPI-2 выглядит еще более громоздким и неподъемным для полной реализации.

Предусматривается развитие в следующих направлениях:

Односторонние коммуникации и средства синхронизации для организации взаимодействия процессов через общую память (для эффективной работы на системах с непосредственным доступом процессоров к памяти других процессоров)

Динамическое создание и уничтожение процессов

Параллельные операции ввода-вывода (для эффективного использования существующих возможностей параллельного доступа многих процессоров к различным дисковым устройствам)


Модель параллелизма по данным
последовательная программа
+
данные
данные
данные
директивы распределения данных
Компилятор
Модель передачи сообщений
библиотека передачи сообщений
процесс
процесс
процесс
данные
данные
данные
Модель с общей памятью
библиотека синхронизации
процесс
процесс
процесс
данные

Модель параллелизма по данным. HPF

- **Отсутствие понятия процесса и, как следствие, явная передача сообщений или явная синхронизация**

- **Данные последовательной программы распределяются программистом по процессорам параллельной машины.**
- **Последовательная программа преобразуется компилятором в параллельную программу, выполняющуюся либо в модели передачи сообщений, либо в модели с общей памятью.**
- **При этом вычисления распределяются по правилу собственных вычислений: каждый процессор выполняет только вычисления собственных данных, т.е. данных, распределенных на этот процессор.**

Достоинства модели параллелизма

Параллелизм по данным – естественный параллелизм вычислительных задач.

В модели параллелизма по данным сохраняется последовательный стиль программирования.

Распределение вычисляемых данных между процессорами – это не только самый компактный способ задать распределение работы между процессорами, но и способ повышения локализации данных. (лучше используется кэш-память, меньше подкачек с диска страниц виртуальной памяти, меньше пересылок данных с других процессоров).

Модель с общей памятью OpenMP

Процессы разделяют общее адресное пространство

Программист должен явно специфицировать общие данные и упорядочивать доступ к ним с помощью средств синхронизации

Появление большого количества языков программирования, реализующих данную концепцию, например Modula-2, ADA

Процессоры соединены с модулями памяти либо через общую шину, либо через коммутатор. Это позволяет обеспечить общее адресное пространство нескольким процессорам.

Интерфейс OpenMP задуман как стандарт для программирования на масштабируемых SMP-системах (SSMP,ccNUMA, etc.) в модели общей памяти (shared memory model).

Механизм выполнения

Одним из механизмов выполнения параллельных процессов стало многонитевое программирование «легковесных процессов», для которых не создаются полноценные процессы с отдельным адресным пространством, но которые все же могут распределяться по процессорам. Появился стандарт создания нитей POSIX thread, поддерживаемый всеми ведущими производителями.

Программа на языке OpenMP является последовательно-параллельной. Последовательные участки выполняются корневым процессом. Для выполнения параллельных участков образуются нити.

- Параллельная программа представляет собой систему нитей (threads), взаимодействующих посредством общих переменных и примитивов синхронизации.

- Нить — легковесный процесс, имеющий с другими нитями общие ресурсы, включая общую оперативную память.

- Эта модель может использоваться только на многопроцессорных системах с общей памятью или с DSM (Distributed Shared Memory — распределенная общая память)

- OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» поток создает набор подчиненных потоков и задача распределяется между ними.

- Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров не обязательно должно быть больше или равно количеству потоков).

Какие преимущества OpenMP дает разработчику?

Идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы

Достаточно гибкий механизм, предоставляющий разработчику большие возможности контроля над поведением параллельного приложения.

Программа на однопроцессорной платформе может быть использована в качестве последовательной программы, т.е. нет необходимости поддерживать последовательную и параллельную версии.

Недостатки OpenMP

- Ограниченность его области применения (мультипроцессоры и DSM-кластеры)
- Имеющиеся в нем средства распараллеливания циклов с зависимостями по данным являются слишком низкоуровневыми
- Программисту фактически позволяется использовать напрямую модель выполнения (программировать в терминах нитей), что может провоцировать создание плохо переносимых программ.

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ OPENMP

На примере простой программы умножения матриц можно увидеть, как использовать OpenMP для параллелизации программы. Рассмотрим следующий небольшой фрагмент кода умножения двух матриц.

```
for (i = 0; i < dim; i++)
{  for (j = 0; j < dim; j++)
   {  for (k = 0; k < dim; k++)
      {array[i][j] += array1[i][k] * array2[k][j];}
    }
}
```

На каждом уровне вложенности циклов тела циклов могут выполняться независимо друг от друга. Чтобы распараллелить приведенный код вставим прагму #pragma omp parallel for перед самым внешним циклом, так как это даст наибольший выигрыш в производительности. В распараллеленном цикле переменные array, array1, array2 и dim являются общими для потоков, а переменные i, j, k являются частными для каждого потока. Приведенный выше код теперь становится таким:

```
#pragma omp parallel for shared(array, array1, array2, dim) private(i, j, k)
for (i = 0; i < dim; i++)
 { for (j = 0; j < dim; j++)
  { for (k = 0; k < dim; k++)
    {array[ii[j] = array1[i][k] * array2[k][jj;}
  }
}
```

```
#include <stdio.h>
#include “mpi.h”
int main (int argc,char* argv[])
{   int rank, n, i, message;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank= =0)
   { printf (“\n  Hello from process %3d”, rank);
      for (i=1;i<n; i++)
      {  MPI_Recv(&message,1,MPI_INT,
MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,
&Status);
          printf (“\n  Hello from process %3d”, message);
            }
     }
else MPI_Send(&rank,1,MPI_INT, 0,0,MPI_COMM_WORLD);
     MPI_Finalize ();
Return 0;
}
```

```
#include “mpi.h”  //
# include <stdio.h>  //
#include <math.h> //
Int main(int argc,char**argv)
Int myrank,np,I,n,y,s1,s2,x(20),a(10),b(10),m=n/2;
File *dat,*res;
dat=fopen(“d:      ,”r”);
res=fopen(“d:      “,”w”);
MPI_Init(&argc,&argv);
MPI_Status status;
MPI_Comm_size(MPI_COMM_WORLD,&np);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
If (myrank==0)
{ //input array X from file
……..
//Divide array X into two arrays A and B
For (i=0; I<n/2;i++)
{A[i]=X[i];
B[i]=X[i+n/2];
}
m=n/2;
MPI_Send(A[0],m,MPI_INT,1,98,MPI_COMM_WORLD);
MPI_Send(B[0],m,MPI_INT,2,99,MPI_COMM_WORLD);
MPI_Recv(S1,1,MPI_INT,1,97,MPI_COMM_WORLD, MPI_Status *status);
```

```
MPI_Recv(s2,1,MPI_INT,2,96,MPI_COMM_WORLD, MPI_Status *status);
Y=s1+s2;
fprintf(res,"The sum of array is = 5d ",y);
fclose(dat);
fclose(res);
}
Else
{ {if (myrank ==1)
{ MPI_Recv(A[0],m,MPI_INT,0,98,MPI_COMM_WORLD, MPI_Status *status);
S1=0;
For (i=0; I<m;i++)
{s1=s1+A[i];
MPI_Send(s1,1,MPI_INT,0,97,MPI_COMM_WORLD);
} else //code for process 2
{ MPI_Recv(B[0],m,MPI_INT,0,99,MPI_COMM_WORLD, MPI_Status *status);
S2=0;
For (i=0; I<m;i++) s2=s2+B[i];
MPI_Send(s2,1,MPI_INT,0,96,MPI_COMM_WORLD);
}
} }
MPI_Finalize();
}
```