

Разработка параллельных программ для систем с распределенной памятью

Создание распределенных приложений

Основы технологии MPI

Особенности систем с распределенной памятью

СИСТЕМЫ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Распределенные системы (1)

- Мультикомпьютеры
 - Кластерные системы (Clusters)
 - Массивно-параллельные процессоры (MPP)

Распределенные системы (2)

- Своя оперативная память
- Своя операционная система
- Различные вычислительные мощности
- **Неограниченное масштабирование**

Распределенные системы (3)

- Возможно использовать только процессы
- **Высокая стоимость коммуникаций**

Способы создания распределенных программ

СОЗДАНИЕ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

Распределенное приложение

- Распределенное приложение (программа) - множество одновременно выполняемых взаимодействующих процессов, которые могут выполняться на одном или нескольких узлах вычислительной сети

Стоимость коммуникаций

- Выбор оптимальной топологии
- Минимизация количества связей
- Минимизация количества пересылок
- Минимизация размера сообщений
- Унификация и типизация сообщений
- И т.д.

Выбор технологии

- Специальные языки
 - Erlang, Go и т.п.
- Общая шина сообщений
 - RabbitMq и т.п.
- Высокоуровневые библиотеки
 - WCF (SOAP), WebAPI (REST), MPI и т.п.
- Низкоуровневые функции и объекты
 - Socket, Pipe и т.п.

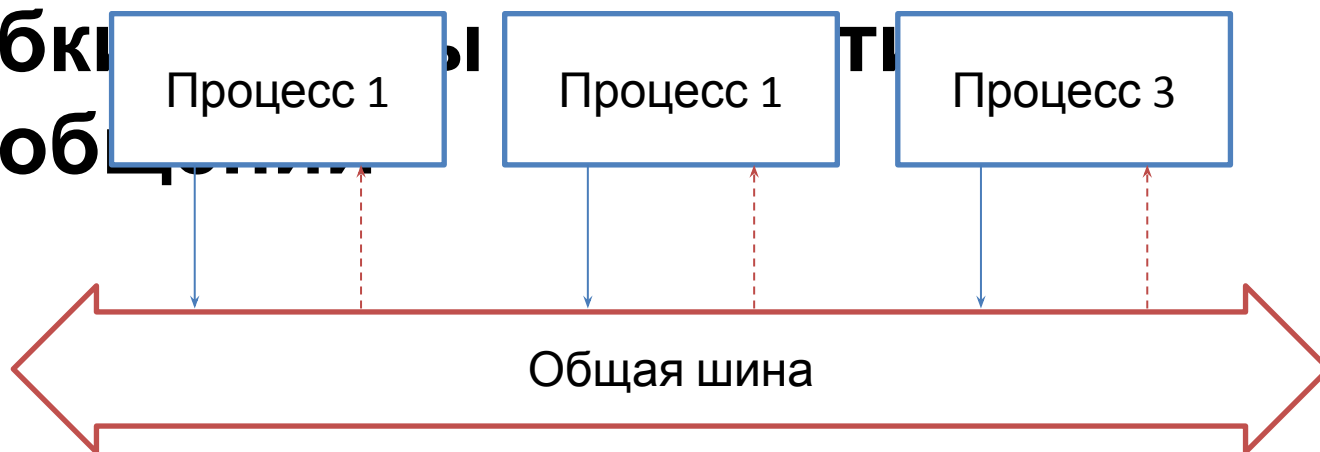
Специализированные языки

- Инкапсулируют сложность коммуникаций
- Основное внимание на прикладной задаче
- Простая запись математических алгоритмов
- **Встроенные средства распараллеливания**

Общая шина сообщений

- Инкапсулирует сложность коммуникаций
- Основное внимание на прикладной задаче
- Простая и очевидная схема взаимодействия

• Гибкие
сообщения



Высокоуровневые библиотеки

- Инкапсулируют сложность коммуникаций
- Основное внимание на прикладной задаче
- Асинхронность чаще реализуется вручную
- Произвольная схема взаимодействия

Низкоуровневые функции

- Сложность осуществления коммуникаций
- Гибкость осуществления коммуникаций
- Вероятная зависимость от платформы
- Произвольная схема взаимодействия

Combo?

- При разработке крупных систем зачастую используются разные подходы

Распараллеливание вычислительных алгоритмов с помощью
MPI

ОСНОВЫ ТЕХНОЛОГИИ MPI

Почему рассматриваем MPI?

- Позволяет раскрыть некоторые особенности реализации более высокоуровневых решений, а также дает хорошее представление о сложностях и особенностях разработки распределенных приложений

MPI

- *«The MPI standard includes point-to-point message-passing, collective communications, group and communicator concepts, process topologies, environmental management, process creation and management, one-sided communications, extended collective operations, external interfaces, I/O, some miscellaneous topics, and a profiling interface»*
 - Стандарт - <http://www.mpi-forum.org>
 - MPICH - <http://www.mpich.org>
 - [Microsoft MPI](#)

Концепция MPI

- Определяет API и протокол обмена сообщениями между процессами распределенного приложения
 - Базовые понятия касаются преимущественно вопросов коммуникации: функции обмена сообщениями, типы данных и формат сообщений, коммутаторы, топологии и способ идентификации процесса (ранг)

Коммуникатор

Приложение

Коммуникатор

Процесс 0

Процесс 1

Процесс 2

Процесс 3

Процесс 4

...

...

Виртуальные топологии

- Есть возможность задавать виртуальную топологию связей между процессами, которая будет отражать логическую взаимосвязь процессов приложения
- Виртуальная топология может быть использована при распределении процессов по узлам с целью уменьшения коммуникационной трудоемкости

Структура кода MPI-процесса

```
#include <mpi.h>
```

```
void main(int argc, char *argv[])
```

```
{
```

```
    <Программный код без использования функций MPI>
```

```
    MPI_Init(&argc, &argv);
```

```
    <Программный код с использованием функций MPI>
```

```
    MPI_Finalize();
```

```
    <Программный код без использования функций MPI>
```

```
}
```

Ранг и количество процессов

```
#include <mpi.h>
```

```
void main(int argc, char *argv[])
```

```
{
```

```
    <Программный код без использования функций MPI>
```

```
    MPI_Init(&argc, &argv);
```

```
    int rank, size;
```

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

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

```
    <Программный код с использованием функций MPI>
```

```
    MPI_Finalize();
```

```
    <Программный код без использования функций MPI>
```

```
}
```

Передача и прием сообщений

```
int MPI_Send(  
    void* buffer,  
    int count,  
    MPI_Datatype type,  
    int dest,  
    int tag,  
    MPI_Comm comm  
);
```

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

Корневой процесс

```
#include <mpi.h>

void main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0)
    {
        // Корневой процесс
    }
    else
    {
        // Дочерние процессы
    }
    MPI_Finalize();
}
```


Редукция

- Передача данных от всех процессов одному процессу (обычно корневному процессу)

```
int MPI_Reduce(  
    void* sendBuffer,  
    void* receiveBuffer,  
    MPI_Datatype type,  
    MPI_Op operation,  
    int root,  
    MPI_Comm comm  
);
```

Барьерная синхронизация

- Точка синхронизации, которую должны достигнуть все процессы, прежде чем продолжат свое выполнение

```
int MPI_Barrier(MPI_Comm comm);
```

Время выполнения

```
double startTime = MPI_Wtime();  
<Блок кода>  
double stopTime = MPI_Wtime();  
double elapsed = stopTime - startTime;
```

Пример вычислений

```
void CalculatePi(int rank, int size, int n)
{
    double pi;

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    double sum = 0.0;
    double coeff = 1.0 / (double)n;

    for (int i = rank + 1; i <= n; i += size)
    {
        double x = coeff * ((double)i + 0.5);
        sum += 4.0 / (1.0 + x * x);
    }

    sum *= coeff;

    MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0)
    {
        printf("PI = %.8f\n", pi);
    }
}
```

ВОПРОСЫ?