Параллельное программирование (параллельные алгоритмы)

Лисицын Сергей
ФРКТ МФТИ 2020 г.

Программа 1 семестра

713	711
718	717

Среда	9^{00} - 10^{25}		Выч.матем. 319 ЛК	Квант.механика 527 ГК		УМФ 414 ГК		УМФ 422 ГК	Алгоритмы на графах/ Погибельский Д.А./ 210 ГК	Операц. системы/ Коньков К.А./ 802 КПМ
	10^{45} - 12^{10}	Выч.матем. 319 ЛК		Выч.матем. 321 ЛК	Физкультура		Физкультура		Физкультура	
	12^{20} - 13^{45}	Ин.яз.	Ин.яз.	Ин.яз.	Ин.яз.	Ин.яз.	Ин.яз.	Ин.яз.	Ин.яз.	Ин.яз.
	13^{55} - 15^{20}	Физкультура	УМФ 418 ГК		УМФ 529 ГК	Выч.матем. 321 ЛК	Квант.механика 413 ГК			
	15^{30} - 16^{55}								УМФ 522 ГК	УМФ 526ГК
	17^{05} - 18^{30}	Параллельное программирование/ Чл.-корр. РАН Якобовский М.В./ Акт.зал								
	18^{35} - 20^{00}			Паралл.програм. (неч.нед.) 806 КПМ			Паралл.програм. (чет.нед.) 801 КПМ		Паралл.прогр. (чет.нед.) 806 КПМ	Паралл.програм. (неч.нед.) 801 КПМ

Суббота	9^{00} - 10^{25}	Физкультура	Лаборатория нелинейных преобразований и р/т сигналов		Квант.механика 524 ГК				Сист.упр.базами данных/ Шумилин/ 321 ЛК
	10^{45} - 12^{10}	Паралл.програм. (неч.нед.) 705 КПМ		Физкультура	Физкультура		Паралл.програм. (чет.нед.) 705 КПМ	Квант.механика 516 ГК	
	12^{20} - 13^{45}	Уравнения математической физики/ доцент Боговский М.Е./ 239 НК							
	13^{55} - 15^{20}								
	15^{30} - 16^{55}		Физкультура			Квант.механика 516 ГК	УМФ 409 ГК	Физкультура	Физкультура	Физкультура
	17^{05} - 18^{30}								
	18^{35} - 20^{00}								

Февраль

№	ПН	ВТ	СР	ЧТ	ПТ	СБ	ВС
5						1	2
6	3	4	5	6	7	8	9
7	10	11	12	13	14	15	16
8	17	18	19	20	21	22	23
9	24	25	26	27	28	29	

Март

№	ПН	ВТ	СР	ЧТ	ПТ	СБ	ВС
9							1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29
14	30	31					

Апрель

№	ПН	ВТ	СР	ЧТ	ПТ	СБ	ВС
14			1	2	3	4	5
15	6	7	8	9	10	11	12
16	13	14	15	16	17	18	19
17	20	21	22	23	24	25	26
18	27	28	29	30			

Май

№	ПН	ВТ	СР	ЧТ	ПТ	СБ	ВС
18					1	2	3
19	4	5	6	7	8	9	10
20	11	12	13	14	15	16	17
21	18	19	20	21	22	23	24
22	25	26	27	28	29	30	31

Программа 1 семестра

6 семинаров*:

1. Вводная лекция
2. Программирование на MPI
3. Статическая балансировка
4. Динамическая балансировка
5. Стандарт POSIX Threads
6. Программирование на общей памяти

*план может быть изменён

Программа 1 семестра

Программы:

- Обязательные задачи (4)
- Бонусные задачи (3)

Оценка за семестр:

+3: Лекционная контрольная

+2: Посещения/3

+2: Мгновенные обязательные задачи/2

+3: Бонусные задачи

Введение

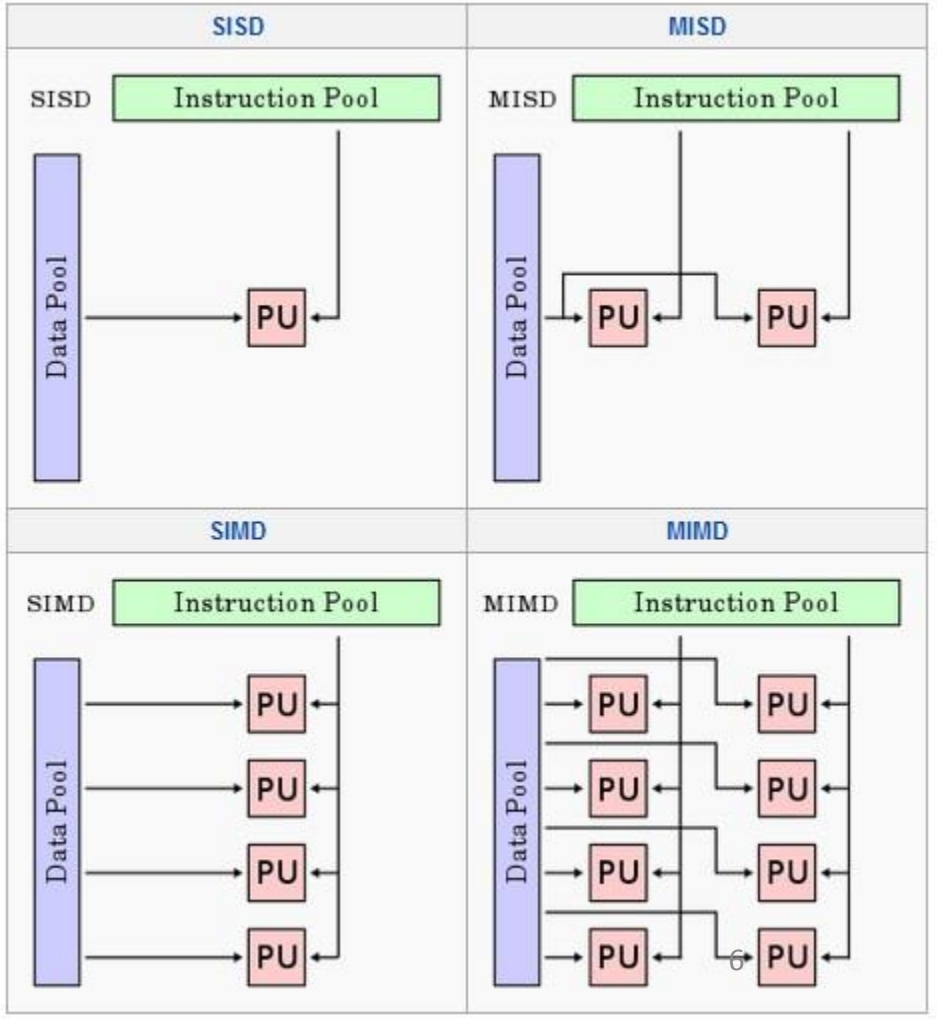
Perf = Freq*IPC/IC

- Конвейеризация вычислений (1970-е) – микроуровневый параллелизм
- Дублирование вычислителей (1980-е) – параллелизм уровня команд (векторизация, VLIW)
- Дублирование “конвейеров” (2000-е)- параллелизм уровня потоков/заданий

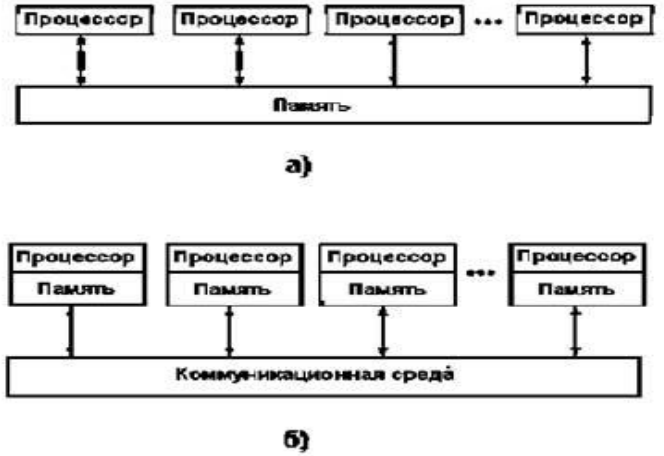
Введение

Таксономия (Классификация) Флинна (1966)

- SISD: компьютер фон-Неймановской архитектуры
- SIMD: векторные процессоры (MMX, SSE), матричные процессоры и процессоры с архитектурой VLIW.
- MISD: не используется
- MIMD:
  - Общая память - Symmetric Multiprocessor SMP
    Разделенная память - Massively Parallel Processing MPP -> Кластерные системы



Введение



Симметричное мультипроцессирование

1. Несколько однородных процессоров и массив общей памяти
2. Когерентность кэшей, урегулирование доступа к памяти
3. Ограниченная масштабируемость
4. Работает под единой ОС
5. Модель программирования: Потоки (pthread, OpenMP)

Массивно-параллельные системы

1. Вычислительные узлы и коммуникационная среда
2. Закрытая локальная память
3. Масштабируемость ~не ограниченная
4. Полная ОС на управляющей машине, на узлах урезанная версия
5. Модель программирования: Модель передачи сообщений (“fork”, MPI, PVM, BSPlib)

Введение

Метрики параллелизма

- Доля последовательных операций: $\alpha = \frac{IC_{|}}{(IC_{|} + IC_{||})}$
- Время на p потоках: $T_p = \alpha T_1 + \frac{(1-\alpha)T_1}{p}$
- Ускорение: $S = T_1/T_p$
- Эффективность: $E = S/p$

Закон Амдала

$$S = \frac{T_1}{T_p} = \frac{T_1}{\alpha T_1 + \frac{(1-\alpha)T_1}{p}} \leq \frac{1}{\alpha}$$

MPI

Message Passing Interface

Параллельная программа - множество одновременно выполняемых **процессов.**

Каждый процесс порождается на основе одного и того же программного кода (fork).

Количество процессов определяется в момент запуска программы.

Все процессы последовательно пронумерованы от 0 до **p-1** (ранг процесса), где **p** есть общее количество процессов.

MPI

- Подключение библиотеки:

```
#include "mpi.h"
```

- Начало работы MPI:

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

- Завершение работы:

```
MPI_Finalize();
```

```
#include "mpi.h"

int main ( int argc, char *argv[] )
{
        <программный код без MPI функций>
    MPI_Init ( &agrc, &argv );
        <программный код с MPI функциями>
    MPI_Finalize();
        <программный код без MPI функций>
    return 0;
}
```

Функции **MPI_Init** и **MPI_Finalize** являются обязательными и должны быть выполнены (только один раз) каждым процессом параллельной программы.

MPI

Компиляция программы:

mpicc -o <исполняемый файл> <исходный файл>.c

Запуск:

mpirun -n <число процессов>

<исполняемый файл> [аргументы]

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

int main(int argc, char* argv[])
{
    int ProcRank;
    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    printf("Hello from process %d\n", ProcRank);

    MPI_Finalize();
    return 0;
}

/// mpicc -o hello main.c
/// mpirun -n 4 hello
```

MPI

- Определение количества процессов

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

- Определения ранга процесса

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

```
#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;
}
```

MPI

Коммуникатор - служебный объект, объединяющий в своем составе группу процессов и контекст, используемый при передачи данных.

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

MPI_Comm comm = MPI_COMM_WORLD;

MPI

Операции передачи сообщений:

- Парные (point-to-point) – операции между двумя процессами

  MPI_Send, MPI_Recv…

- Коллективные (collective) – коммуникационные действия для **одновременного** взаимодействия нескольких процессов

  MPI_Bcast, MPI_Reduce…

MPI

Базовые типы

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

Производные типы

- Функции создания типа
  MPI_Type_contiguous
  MPI_Type_vector
  MPI_Type_hvector
  MPI_Type_indexed
  MPI_Type_hindexed
  MPI_Type_struct

- Функция регистрации типа
  `int` **`MPI_Type_commit`**
  `(MPI_Datatype *datatype);`

MPI

Передача сообщений:

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

Приём сообщений:

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

MPI_ANY_SOURCE

MPI_ANY_TAG

MPI_COMM_WORLD

buf – адрес буфера с данными отправляемого сообщения.
count – количество элементов данных в сообщении.
type - тип элементов данных пересылаемого сообщения.
dest - ранг процесса, которому отправляется сообщение.
source - ранг процесса, от которого должен быть выполнен прием сообщения.
tag - значение-тег, используемое для идентификации сообщений.
comm - коммуникатор, в рамках которого выполняется передача данных.
status – указатель на структуру данных с информацией о результате выполнения операции.

```c
#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 %d", 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 %d", 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;
}
```

MPI

Описание всех функций:
https://www.open-mpi.org/doc/current/

Подробное описание MPI на русском:
http://rsusu1.rnd.runnet.ru/tutor/method/m2/content.html