Параллельное программирование

(Численные методы, алгоритмы и программы. Введение в распараллеливание)

Лисицын Сергей
ФРКТ МФТИ 2018 г.

Программа 2 семестра

15 занятий:

Сентябрь (4) – OpenMP

Октябрь (4) – Распараллеливание циклов

Ноябрь (5) – Вычматы

Декабрь (2) – Долги

Книга: Карпов, Лобанов

Четверг	9^{00} - 10^{25}	Машин.обуч. 319 ЛК	Импульсные цифр.устр. /Доцент Поурцев В.П./430 ГК	Введ. в распарал. алг.и программ 801 КПМ		Импульсные цифр.устр. /Доцент Поурцев В.П./430 ГК	Квант.мех. 507а ГК	Введ. в распарал. алг.и программ 804 КПМ		Импульсные цифр.устр./Доцент Поурцев В.П./430 ГК
	10^{45} - 12^{10}					Квант.мех. 507а ГК				Лаборатория ИЦУ
	12^{20} - 13^{45}			Квант.мех. 507а ГК	Ин.яз.		Ин.яз.	Ин.яз.	Введ. в распарал. алг.и программ 801 КПМ	
	13^{55} - 15^{20}	Квант.мех. 507а ГК				Ин.яз.	Введ. в распарал. алг.и программ 801 КПМ		Ин.яз.	
	15^{30} - 16^{55}	Военная подготовка								
	17^{05} - 18^{30}									
	18^{35} - 20^{00}									
Пятница	9^{00} - 10^{25}	Квантовая механика/ Доцент Гец А.В./ 202 НК								
	10^{45} - 12^{10}	Ин.яз.	Ин.яз.		Введ. в распарал. алг.и программ 801 КПМ			Квант.мех. 507а ГК		Ин.яз.
	12^{20} - 13^{45}		Квант.мех. 507а ГК							
	13^{55} - 15^{20}	Введ. в распарал. алг.и программ /Доцент Карпов В.Е./ 202 НК		Введ. в распарал. алг.и программ /Доцент Карпов В.Е./ 202 НК			Введ. в распарал. алг.и программ /Доцент Карпов В.Е./202 НК			
	15^{30} - 16^{55}	Введ. в распарал. алг.и программ 802 КПМ	Лаборатория ИЦУ	Ин.яз.	Квант.мех. 507а ГК	Лаборатория ИЦУ				Ин. яз.
	17^{05} - 18^{30}									
	18^{35} - 20^{00}									

Программа 2 семестра

Программы:

- Обязательные задачи (9)

  2 по каждой теме для зачёта (6)
- Бонусные задачи (3)

Оценка за семестр:

+3: Лекционная контрольная

+3: Посещения/5

+3: Мгновенные обязательные задачи/3

+3: Бонусные задачи

Введение

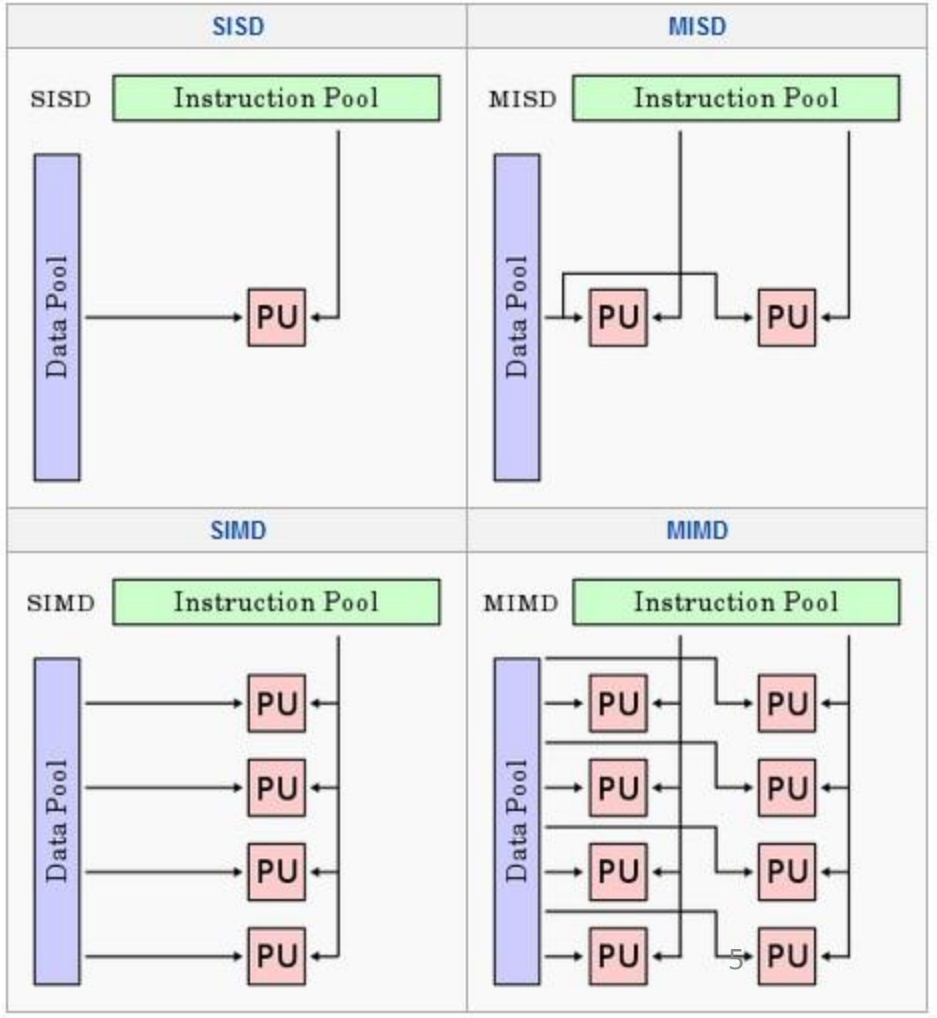
Perf = Freq*IPC/IC

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

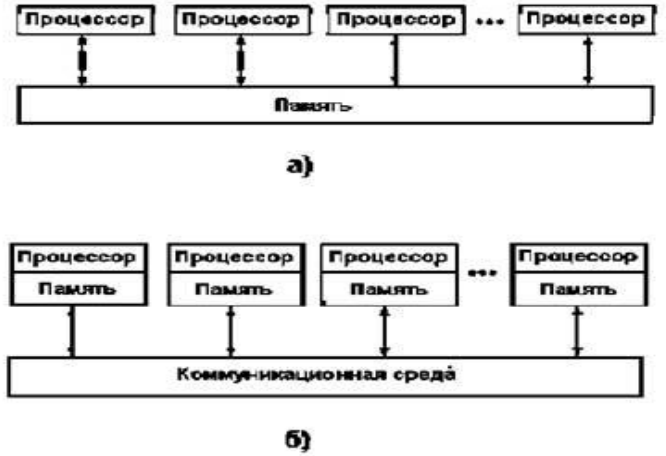
Введение

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)

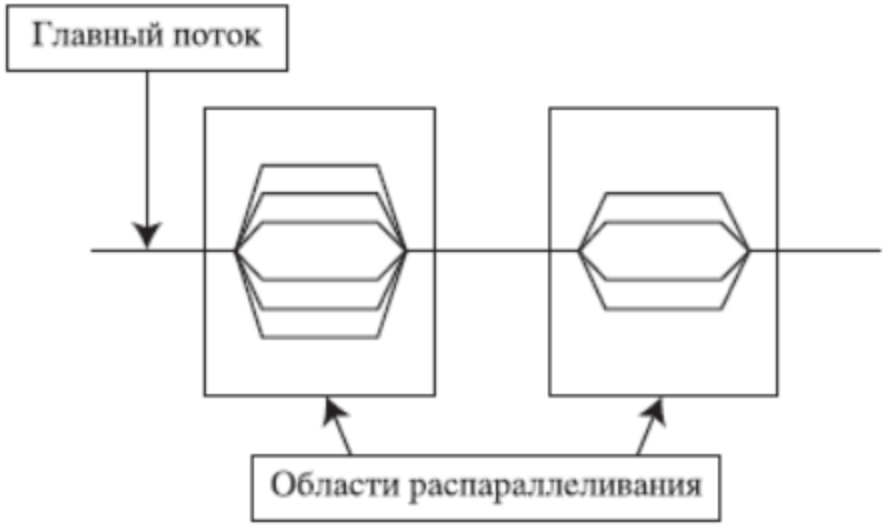
Авторизация на сервере

ssh <username>@calcnoqueue.vdi.mipt.ru

ssh –p 60001 <username>@remote.vdi.mipt.ru

<username> = s<group><id> (s51316)

OpenMP



Open Multi-Processing

Директивы компилятора, библиотечные процедуры и переменные окружения

Системы с общей памятью

Языки: Си, Си++ и Фортран

#pragma omp конструкция [предложение [предложение] …]

OpenMP

```
#include <omp.h> // Open Multi-Processing


int main(int argc, char **argv) {
	omp_set_num_threads(N); // установить число потоков в N
#pragma omp parallel // директива компилятору
{
	// параллельное исполнение
}
	return 0x00;
}
```

OpenMP

Компиляция:

```
gcc my_openMP_prog.c -o my_openMP_prog -fopenmp
```

OpenMP

```
#pragma omp parallel
{
    // parallel указывает, что данный блок кода должен быть исполнен
    // параллельно в несколько потоков
}

#pragma omp parallel // сокращенная запись
    … // блок кода, исполняющийся параллельно
```

OpenMP

```
#pragma omp parallel
{
    #pragma omp for
    for (int  i = 0; i < K; i++) // параллельное суммирование чисел 0..K
        res += i; // с теоретическим ускорением N, где N - число потоков
}

#pragma omp parallel for // сокращенная запись
    … // цикл for, исполняющийся параллельно
```

OpenMP

```
int i;

#pragma omp parallel for private(i)
        for (i = 0; i < K; i++) // параллельная печать чисел 0..K
                printf("i: %i\n", i); // с теоретическим ускорением N, где N - число потоков

int i;

#pragma omp parallel for shared(K) private(i)
        for (i = 0; i < K; i++) // параллельная печать чисел 0..K
                printf("i: %i\n", i); // с теоретическим ускорением N, где N - число потоков
```

OpenMP

```
#pragma omp parallel shared(a) private(myid, x)
{
    myid = omp_get_thread_num();
    x = work(myid);
    if(x < 1.0)
        a[myid] = x;
}
#pragma omp parallel default(private) shared(a)
{
    myid = omp_get_thread_num();
    x = work(myid);
    if(x < 1.0)
        a[myid] = x;
}
```

OpenMP

```
///Устанавливает количество потоков, которое может быть запрошено для
///параллельного блока.
void omp_set_num_threads(int num_threads)

///Возвращает количество потоков в текущей команде параллельных потоков
int omp_get_num_threads()

///Возвращает максимальное количество потоков, которое может быть установлено
int omp_get_max_threads()

///Возвращает номер потока в команде (целое число от 0 до N – 1)
int omp_get_thread_num()

///Возвращает количество физических процессоров доступных программе
int omp_get_num_procs()

///Возвращает не нулевое значение, если вызвана внутри параллельного блока
///В противном случае возвращается 0
int omp_in_parallel()
```

OpenMP

- firstprivate(var1, var2, ...) private + указанные переменные инициализируются значением до входа в параллельную секцию.
- lastprivate(var1, var2, ...) Приватные переменные сохраняют свое значение, которое они получили при достижении конца параллельного участка кода.
- reduction(оператор:var1, var2, ...) гарантирует безопасное выполнение операций редукции, например, вычисление глобальной суммы.
- if(выражение) параллельное выполнение необходимо только если выражение истинно.

OpenMP

- sections / section – разделение задач между потоками
- single – при необходимости сделать действие одним потоком в параллельном участке
- ordered – в параллельных циклах говорит о исполнении в строго фиксированной последовательности

OpenMP

```
int myid;
int a = 10;
#pragma omp parallel default(private) \
 firstprivate(a)
{
    myid = omp_get_thread_num();
    printf("Thread%d: a = %d\n", myid, a);
    a = myid;
    printf("Thread%d: a = %d\n", myid, a);
}
```

OpenMP

```
#pragma omp parallel
{
    #pragma omp for private(i) lastprivate(k)
    for(i=0; i<10; i++)
        k = i*i;
}
printf("k = %d\n", k);
```

OpenMP

```
#pragma omp parallel sections ///nowait
{
    #pragma omp section
    {
        printf("T%d: task1\n", omp_get_thread_num());
    }
    #pragma omp section
    {
        printf("T%d: task1\n", omp_get_thread_num());
    }
}
```

OpenMP

```
#pragma omp parallel
{
    #pragma for shared(x) private(i) reduction(+:sum)
    for(i=0; i<10000; i++)
        sum += x[i];
}
#pragma omp parallel
{
    #pragma for shared(x) private(i) reduction(min:gsum)
    for(i=0; i<10000; i++)
        gmin = min(gmin, x[i]);
}
```

+, - , *, &, ^, |, &&, ||, min, max

OpenMP

```
#pragma omp parallel
{
    #pragma omp for if(n>2000)
    {
        for(i=0; i<n; i++)
            a[i] = work(i);
    }
}
```

OpenMP

schedule(тип [, размер блока])

static – итерации равномерно распределяются по потокам, нудным размером блока.

dynamic – работа распределяется пакетами заданного размера между потоками. При завершении текущего блока берёт следующий.

guided – dynamic + Размер блока постепенно уменьшается вплоть до указанного значения.

OpenMP

critical – критическая секция

atomic – атомарность операции

barrier – точка синхронизации

master – блок, который будет выполнен только основным потоком

OpenMP