



**НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
**им. Н.И. Лобачевского**  
**- Национальный исследовательский университет -**





Нижегородский государственный университет им. Н.И. Лобачевского  
– Национальный исследовательский университет –

# Параллельное программирование с использованием OpenMP

**Гергель В.П.**

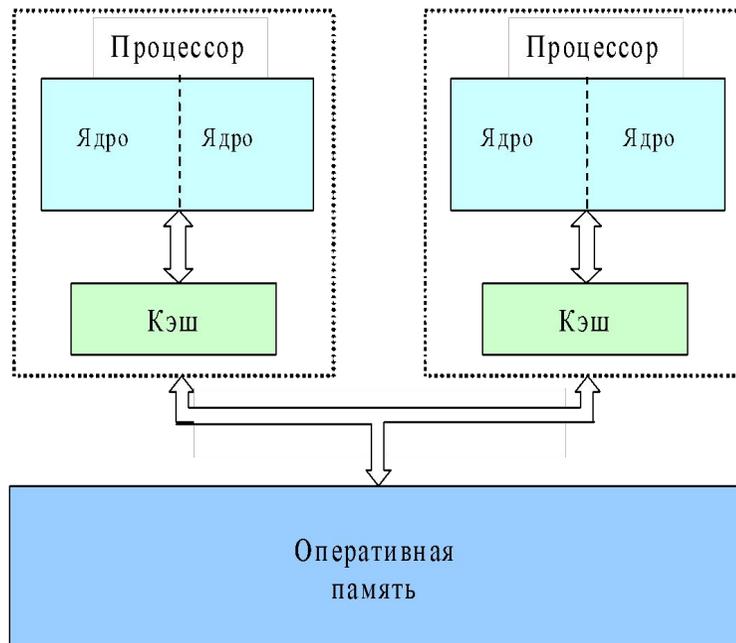
д.т.н, проф., декан факультета ВМК,  
руководитель Центра  
суперкомпьютерных технологий ННГУ

# ОСНОВЫ ПОДХОДА



# Технология OpenMP для систем с общей памятью

- Технология OpenMP задумана как стандарт параллельного программирования для многопроцессорных систем с общей памятью (SMP, ccNUMA, ...).



Системы с общей памятью описываются в виде модели параллельного компьютера с произвольным доступом к памяти (parallel random-access machine – PRAM).

# Способы разработки программ для параллельных вычислений...

- **Способ 1:** автоматическое распараллеливание последовательных программ.
  - возможности автоматического построения параллельных программ ограничены.
- **Способ 2:** расширение существующих алгоритмических языков средствами параллельного программирования.
- **Способ 3:** использование новых алгоритмических языков параллельного программирования.

*Примечание:* Способы 2-3 приводят к необходимости значительной переработки существующего программного обеспечения.

# Способы разработки программ для параллельных вычислений...

- **Способ 4:** использование тех или иных **внеязыковых средств языка программирования**.
  - Примеры средств – директивы или комментарии, которые обрабатываются специальным препроцессором до начала компиляции программы.
  - Директивы дают **указания** на возможные способы распараллеливания программы, при этом **исходный текст программы остается неизменным**.
  - Препроцессор заменяет **директивы** параллелизма **на дополнительный программный код** (как правило, в виде обращений к процедурам параллельной библиотеки).
  - При отсутствии препроцессора компилятор построит исходный последовательный программный код.

# Способы разработки программ для параллельных вычислений

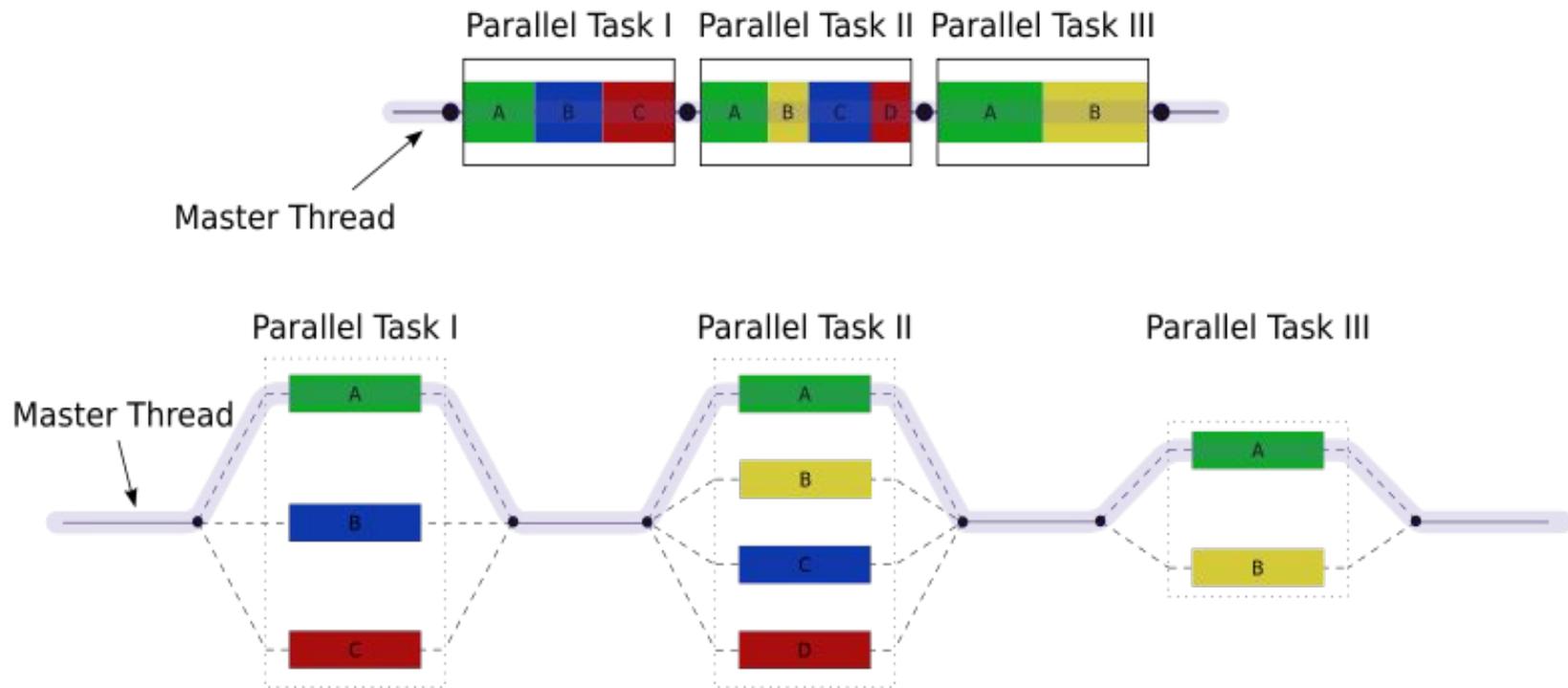
- **Способ 4: Положительные стороны:**
  - **Снижает потребность переработки существующего программного кода.**
  - **Обеспечивает единство программного кода для последовательных и параллельных вычислений,** что снижает сложность развития и сопровождения программ.
  - **Позволяет осуществлять поэтапную (инкрементную) разработку параллельных программ.**

***Именно этот подход и является основой технологии OpenMP***



# Принципы организации параллелизма...

- ❑ Использование потоков (общее адресное пространство).
- ❑ «Пульсирующий» (fork-join) параллелизм.



\*Источник: <http://en.wikipedia.org/wiki/OpenMP>

# Принципы организации параллелизма

- ❑ При выполнении обычного кода (вне параллельных областей) программа исполняется одним потоком (*master thread*).
- ❑ При появлении директивы **parallel** происходит создание “команды” (*team*) потоков для параллельного выполнения вычислений.
- ❑ После выхода из области действия директивы **parallel** происходит синхронизация, все потоки, кроме *master*, уничтожаются (или приостанавливаются).
- ❑ Продолжается последовательное выполнение кода (до очередного появления директивы **parallel**).



# Структура OpenMP

---

- Компоненты:
  - Набор директив компилятора.
  - Библиотека функций.
  - Набор переменных окружения.
  
- Изложение материала будет проводиться на примере языка C/C++.

# ДИРЕКТИВЫ OPENMP

Формат, области видимости, типы.

Директива определения параллельной области.



# Формат записи директив...

- **Формат:**

```
#pragma omp имя_директивы [параметр,...]
```

- **Пример:**

```
#pragma omp parallel default(shared) private(beta,pi)
```

**Примечание:** На английском языке для термина «параметр» используется обозначение «clause».

# Области видимости директив

root.cpp

```
#pragma omp parallel  
{  
    TypeThreadNum();  
}
```

node.cpp

```
void TypeThreadNum() {  
    int num;  
    num = omp_get_thread_num();  
    #pragma omp critical  
    printf("Hello from %d\n", num);  
}
```

**Статический (лексический) контекст**  
параллельной области – блок, следующий за директивой `parallel`.

**Динамический контекст**  
параллельной области (включает статический контекст) – все функции из блока `parallel`.

**Отделяемые (orphaned) директивы** – директивы синхронизации и распределения работы. Могут появляться вне параллельной области.



# Типы директив

---

- ❑ Определение параллельной области.
- ❑ Разделение работы.
- ❑ Синхронизация.

# Определение параллельной области

- Директива **parallel** (основная директива OpenMP):
  - Когда основной поток выполнения достигает директиву **parallel**, создается набор (*team*) потоков.
  - Входной поток является основным потоком (*master thread*) этого набора и имеет номер 0.
  - Код области дублируется или разделяется между потоками для параллельного выполнения.
  - В конце области обеспечивается синхронизация потоков – выполняется ожидание завершения вычислений всех потоков.
  - Далее все потоки завершаются, и последующие вычисления продолжает выполнять только основной поток.



# Формат директивы

- Формат директивы **parallel** :

```
#pragma omp parallel [clause ...]  
    structured_block
```

- Возможные параметры (**clauses**):

```
if (scalar_expression)
```

```
private (list)
```

```
firstprivate (list)
```

```
default (shared | none)
```

```
shared (list)
```

```
copyin (list)
```

```
reduction (operator: list)
```

```
num_threads (integer-expression)
```



# Пример использования директивы...

```
#include <omp.h>
void main() {
    int nthreads, tid;
    // Создание параллельной области
    #pragma omp parallel private(tid)
    {
        // печать номера потока
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);
        // Печать количества потоков - только master
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    } // Завершение параллельной области
}
```



# Пример использования директивы

- Пример результатов выполнения программы для четырех потоков

```
Hello World from thread = 1  
Hello World from thread = 3  
Hello World from thread = 0  
Hello World from thread = 2
```

**Примечание:** Порядок печати потоков может меняться от запуска к запуску!!!

# Установка количества потоков

- Способы задания (по убыванию старшинства)
  - Параметр директивы:  
**num\_threads (N)**
  
  - Функция установки числа потоков:  
**omp\_set\_num\_threads (N)**
  
  - Переменная окружения:  
**OMP\_NUM\_THREADS**
  
  - Число, равное количеству процессоров, которое “видит” операционная система.



# Определение времени выполнения параллельной программы

```
double t1, t2, dt;  
t1 = omp_get_wtime ();  
...  
t2 = omp_get_wtime ();  
dt = t2 - t1;
```

# ДИРЕКТИВЫ OPENMP

Управление областью видимости данных.



# Директивы управления областью видимости

- Управление областью видимости обеспечивается при помощи параметров (**clauses**) директив:
  - **shared, default**
  - **private**
  - **firstprivate**
  - **lastprivate**
  - **reduction, copyin.**
  
- Параметры директив определяют, какие соотношения существуют между переменными последовательных и параллельных фрагментов выполняемой программы.



# Параметр `shared`

- Параметр `shared` определяет список переменных, которые будут общими для всех потоков параллельной области.

```
#pragma omp parallel shared(list)
```

**Примечание:** правильность использования таких переменных должна обеспечиваться программистом.

# Параметр `private`

- ❑ Параметр `private` определяет список переменных, которые будут локальными для каждого потока.

```
#pragma omp parallel private(list)
```

- ❑ Переменные создаются в момент формирования потоков параллельной области.
- ❑ Начальное значение переменных является неопределенным.

# Пример использования директивы private

```
#include <omp.h>
void main () {
    int nthreads, tid;
    // Создание параллельной области
    #pragma omp parallel private(tid)
    {
        // печать номера потока
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);
        // Печать количества потоков - только master
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    } // Завершение параллельной области
}
```



# Параметр `firstprivate`

---

- Параметр `firstprivate` позволяет создать локальные переменные потоков, которые перед использованием инициализируются значениями исходных переменных.

```
#pragma omp parallel firstprivate(list)
```

# Параметр `lastprivate`

- Параметр `lastprivate` позволяет создать локальные переменные потоков, значения которых запоминаются в исходных переменных после завершения параллельной области (используются значения потока, выполнившего последнюю итерацию цикла или последнюю секцию).

```
#pragma omp parallel lastprivate(list)
```

# ДИРЕКТИВЫ OPENMP

Распределение вычислений между потоками



# Директивы распределения вычислений между потоками

- ❑ Существует 3 директивы для распределения вычислений в параллельной области:
  - **for** – распараллеливание циклов.
  - **sections** – распараллеливание отдельных фрагментов кода (функциональное распараллеливание).
  - **single** – директива для указания последовательного выполнения кода.
- ❑ Начало выполнения директив по умолчанию не синхронизируется.
- ❑ Завершение директив по умолчанию является синхронным.



# Директива `for`

- Формат директивы `for`:

```
#pragma omp for [clause ...]  
for loop
```

- Возможные параметры (`clause`):

- `private(list)`
- `firstprivate(list)`
- `lastprivate(list)`
- `schedule(kind[, chunk_size])`
- `reduction(operator: list)`
- `ordered`
- `nowait`



# Пример использования директивы for

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
void main() {
    int i, n, chunk;
    float a[NMAX], b[NMAX], c[NMAX];
    for (i=0; i < NMAX; i++)
        a[i] = b[i] = i * 1.0;
    n = NMAX; chunk = CHUNK;
    #pragma omp parallel shared(a,b,c,n,chunk) private(i)
    {
        #pragma omp for
        for (i=0; i < n; i++)
            c[i] = a[i] + b[i];
    } // end of parallel section
}
```



# Директива for. Параметр schedule

- ❑ **static** – итерации делятся на блоки по **chunk** итераций и статически разделяются между потоками.
  - Если параметр **chunk** не определен, итерации делятся между потоками равномерно и непрерывно.
- ❑ **dynamic** – распределение итерационных блоков осуществляется динамически (по умолчанию **chunk=1**).
- ❑ **guided** – размер итерационного блока уменьшается экспоненциально при каждом распределении.
  - **chunk** определяет минимальный размер блока (по умолчанию **chunk=1**).
- ❑ **runtime** – правило распределения определяется переменной **OMP\_SCHEDULE** (при использовании **runtime** параметр **chunk** задаваться не должен).



# Пример использования директивы for

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
void main() {
    int i, n, chunk;
    float a[NMAX], b[NMAX], c[NMAX];
    for (i=0; i < NMAX; i++)
        a[i] = b[i] = i * 1.0;
    n = NMAX; chunk = CHUNK;
    #pragma omp parallel shared(a,b,c,n,chunk) private(i)
    {
        #pragma omp for schedule(dynamic,chunk)
        for (i=0; i < n; i++)
            c[i] = a[i] + b[i];
    } // end of parallel section
}
```



# Директива `sections`...

- Формат директивы `sections`:

```
#pragma omp sections [clause ...]  
{  
    #pragma omp section // несколько секций  
    structured_block  
}
```

- Возможные параметры (`clause`):

- `private(list)`,
- `firstprivate(list)`,
- `lastprivate(list)`,
- `reduction(operator: list)`,
- `nowait`



# Директива `sections`

- Директива `sections` – распределение вычислений для отдельных фрагментов кода.
- Фрагменты кода:
  - Выделяются при помощи директивы `section`.
  - Каждый фрагмент выполняется однократно.
  - Разные фрагменты выполняются разными потоками.
  - Завершение директивы по умолчанию синхронизируется
  - Директивы `section` должны использоваться только в статическом контексте.

# Пример использования директивы sections...

```
#include <omp.h>
#define NMAX 1000
void main() {
    int i, n;
    float a[NMAX], b[NMAX], c[NMAX];
    for (i=0; i < NMAX; i++)
        a[i] = b[i] = i * 1.0;
    n = NMAX;
    #pragma omp parallel shared(a,b,c,n) private(i)
    {
        // продолжение на следующем слайде
    }
}
```

# Пример использования директивы sections

```
#pragma omp sections nowait
{
    #pragma omp section
        for (i=0; i < n/2; i++)
            c[i] = a[i] + b[i];
    #pragma omp section
        for (i=n/2; i < n; i++)
            c[i] = a[i] + b[i];
} // end of sections
} // end of parallel section
}
```

# Объединение директив `parallel` и `for/sections`

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
void main () {
    int i, n, chunk;
    float a[NMAX], b[NMAX], c[NMAX];
    for (i=0; i < NMAX; i++)
        a[i] = b[i] = i * 1.0;
    n = NMAX;
    chunk = CHUNK;
    #pragma omp parallel for shared(a,b,c,n) \
        schedule(static,chunk)
        for (i=0; i < n; i++)
            c[i] = a[i] + b[i];
}
```



# ДИРЕКТИВЫ OPENMP

Операция редукции



# Параметр `reduction`

- ❑ Параметр `reduction` определяет список переменных, для которых выполняется операция редукции (объединения).

`reduction` (operator: list)

- ❑ Перед выполнением параллельной области для каждого потока создаются копии этих переменных.
- ❑ Потоки формируют значения в своих локальных переменных.
- ❑ При завершении параллельной области на всеми локальными значениями выполняются необходимые операции редукции, результаты которых запоминаются в исходных (глобальных) переменных.



# Пример использования параметра reduction

```
#include <omp.h>
void main() { // vector dot product
    int i, n, chunk;
    float a[100], b[100], result;
    n = 100; chunk = 10;
    result = 0.0;
    for (i=0; i < n; i++) {
        a[i] = i * 1.0; b[i] = i * 2.0;
    }
    #pragma omp parallel for default(shared) \
        schedule(static,chunk) reduction(+:result)
    for (i=0; i < n; i++)
        result = result + (a[i] * b[i]);
    printf("Final result= %f\n",result);
}
```



# Правила записи параметра `reduction`

- Возможный формат записи выражения:
  - `x = x op expr`
  - `x = expr op x`
  - `x binop = expr`
  - `x++`, `++x`, `x--`, `--x`
- `x` должна быть скалярной переменной.
- `expr` не должно ссылаться на `x`.
- `op` (operator) должна быть неперегруженной операцией вида:
  - `+`, `-`, `*`, `/`, `&`, `^`, `|`, `&&`, `||`
- `binop` должна быть неперегруженной операцией вида:
  - `+`, `-`, `*`, `/`, `&`, `^`, `|`



# Заключение

- ❑ Данная лекция посвящена рассмотрению методов параллельного программирования для вычислительных систем с общей памятью с использованием технологии OpenMP.
- ❑ В лекции проводится обзор технологии OpenMP.
- ❑ Рассматриваются директивы OpenMP, позволяющие
  - определять параллельные области,
  - управлять областью видимости данных,
  - распределять вычислений между потоками,
  - выполнять операции редукции.
- ❑ Дальнейшее изучение технологии OpenMP будет продолжено в одной из следующих лекций.



# Литература

**Гергель В.П.** Высокопроизводительные вычисления для многопроцессорных многоядерных систем. - М.: Изд-во Московского университета, 2010. – 544 с.

□ **Дополнительная литература:**

- **Воеводин В.В., Воеводин Вл.В.** Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
- **Гергель В.П.** Теория и практика параллельных вычислений. - М.: Интернет-Университет, БИНОМ. Лаборатория знаний, 2007.
- **Гергель В.П.** Новые языки и технологии параллельного программирования. - М.: Издательство Московского университета, 2012. – 434 с.
- **Гергель В.П., Баркалов К.А., Мееров И.Б., Сысоев А.В. и др.** Параллельные вычисления. Технологии и численные методы. Учебное пособие в 4 томах. – Нижний Новгород: Изд-во Нижегородского госуниверситета, 2013. – 1394 с.

