Введение в OpenMP

Содержание

Многопроцессорные системы

Что такое OpenMP?

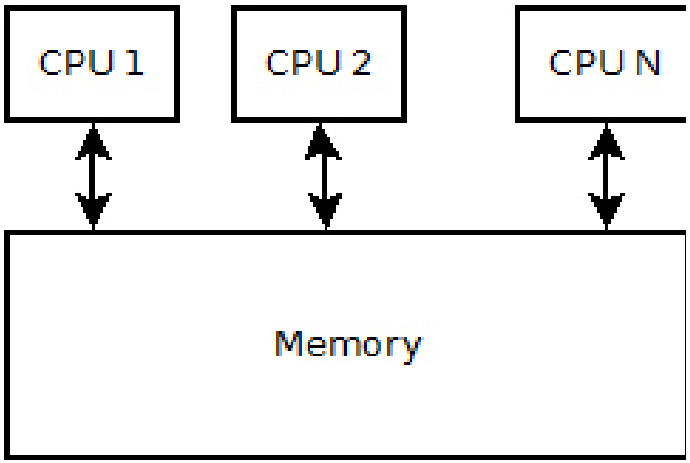
Основные компоненты

- Переменные окружения
- Функции
- Директивы

Виды памяти в многопроцессорных системах

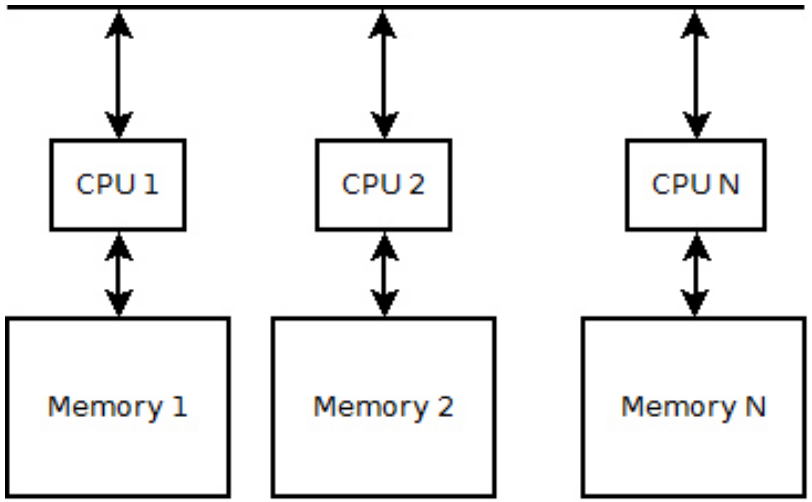
ОБЩАЯ / РАЗДЕЛЯЕМАЯ (SMP)

Symmetric Multiprocessing



РАСПРЕДЕЛЕННАЯ (MPP)

Massive Parallel Processing



Особенности программирования для SMP и MPP систем

SMP

- “+”
  - Хорошее межпроцессорное взаимодействие
- “-”
  - Плохая масштабируемость

MPP

- “+”
  - Хорошая масштабируемость
- “-”
  - Плохое межпроцессорное взаимодействие

Современные CPU

Intel Core

- ~4 ядра

Intel Xeon

- ~12 ядер

Intel Xeon Phi

- ~60 ядер

Hyper-Threading Technology (?)

Что такое OpenMP?

Промышленный стандарт разработки параллельных программ для SMP-систем

- OpenMP основан на модели разделяемой памяти

Сайт проекта

- www.openmp.org

Текущая версия

- 4.5 (ноябрь 2015)

Преимущества OpenMP

Простота использования

Переносимость

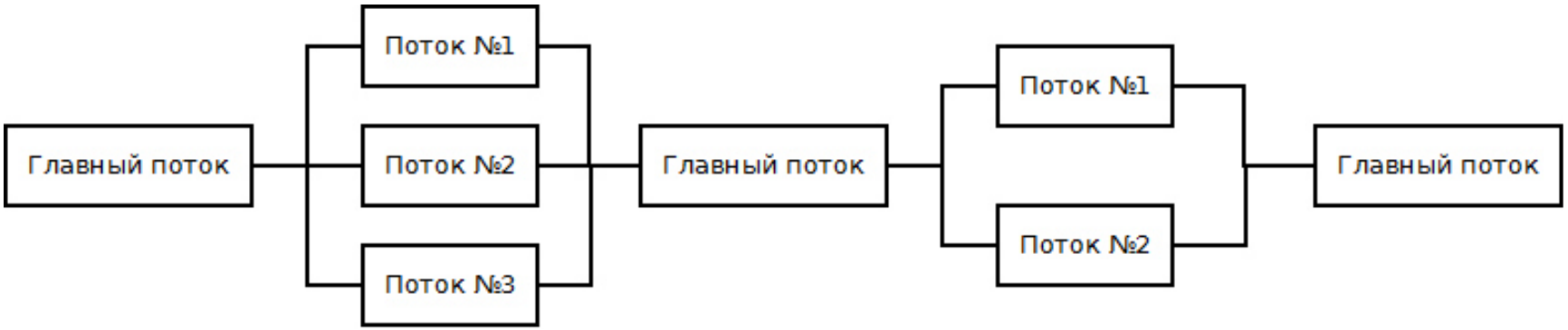
Низкие накладные расходы

Поддержка параллелизма по данным

- SIMD (Single Instruction Multiple Data)

Многопоточный параллелизм

Модель fork-join



Основные компоненты OpenMP

Набор директив

- `#pragma omp …`

Набор функций библиотеки

- `int omp_get_num_procs(void)`
- …

Переменные окружения

- `OMP_NUM_THREADS`

Как использовать OpenMP?

Подключить заголовочный файл omp.h

Использовать директивы препроцессора для распараллеливания выбранных участков кода

Компиляция и запуск

icc

- Linux: ключ `-openmp`
- Windows: ключ `/Qopenmp`

gcc

- Ключ `-fopenmp`

MS Visual Studio

- Project → Properties
- Configuration Properties → C/C++ → Language → OpenMP Support → Yes

Hello World

```
#include <stdio.h>
#include <omp.h>
int main() {
#pragma omp parallel
printf("Hello World!\n");
return 0;
}
```

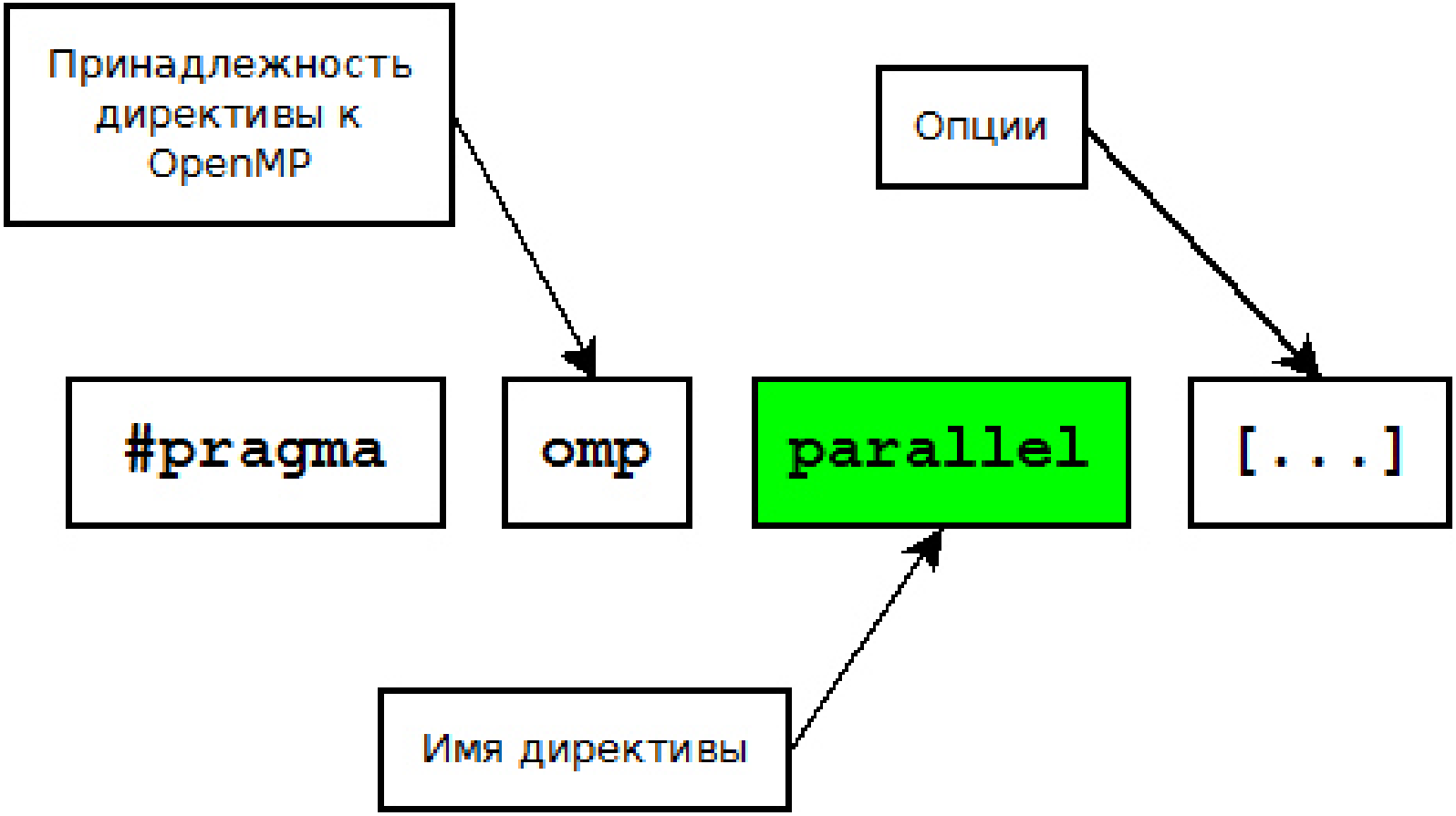
Hello World

```
[u1214@umt lambda]$ icc -openmp hello.c -o hello
[u1214@umt lambda]$ ./hello
Hello World!
Hello World!
Hello World!
Hello World!
```

Hello World

```
[u1214@umt lambda]$ icc hello.c -o hello
hello.c(4): warning #3180: unrecognized OpenMP
#pragma
#pragma omp parallel
        ^
[u1214@umt lambda]$ ./hello
Hello World!
```

Общий вид директив OpenMP



Директива `parallel`

При встрече директивы `#pragma omp parallel` создается команда потоков

Потоки нумеруются с 0

Каждый поток исполняет код в структурном блоке

В конце структурного блока неявный барьер для всех потоков

Поток оперирует с “внутренними” и “внешними” переменными

Создание необходимого количества потоков

С помощью переменной окружения

- `export OMP_NUM_THREADS=N`

С помощью опции

- `omp_set_num_threads(N);`

С помощью опции

- `#pragma omp parallel num_threads(N)`

Функции для работы с потоками

Получение количества процессоров, доступных для работы

- `int omp_get_num_procs(void)`

Получение количества потоков в группе

- `int omp_get_num_threads(void)`

Получение номера потока в группе

- `int omp_get_thread_num(void)`

Функции для измерения времени

double omp_get_wtime(**void**)

- Возвращает время в секундах с некоторого момента в прошлом

double omp_get_wtick(**void**)

- Возвращает количество секунд между тактами часов процессора

Распараллеливание циклов

```
#pragma omp parallel for
for(init; var rel b; incr)
```

Предполагается отсутствие информационных зависимостей между итерациями

`init` должно быть явно задано

`rel`

- `<, <=, >, >=`

`incr`

- `++, --, +=, -=` и т. п.

Распределение итераций цикла

```
#pragma omp parallel for schedule(...)
```

`schedule(static, chunk_size)`

- Статическое распределение порций цикла размером `chunk_size`

`schedule(dynamic, chunk_size)`

- Динамическое распределение порций цикла размером `chunk_size`

`schedule(guided, chunk_size)`

- Динамическое распределение порций цикла размером не меньше `chunk_size`

Опции для переменных

`private(...)`

- Задает список индивидуальных для потока переменных

`firstprivate(...)`

- Переменные в списке получают значение, равное значению переменной на главном потоке в момент входа в параллельный участок

`shared(...)`

- Задает список переменных, общих для всех потоков

Опция `reduction`

Определяет значение переменных, входящих в список ее аргументов, на главном потоке после завершения параллельного участка как результат выполнения редуктивной операции

- `reduction (operator:list)`

Пример

- `#pragma omp parallel reduction(+:X) num_threads(N)`

Инициализация

- `+ → 0`
- `* → 1`

Синхронизация в OpenMP

`#pragma omp atomic`

- Обеспечение атомарности операции, к которой применяется директива

`#pragma omp critical`

- Критическая секция (выполняется только одним потоком)

`#pragma omp master`

- Операторы выполняются только главным потоком

`#pragma omp barrier`

- Барьерная синхронизация

Практические задания

№1

- Ускорить выполнение цикла **`for`** в программе, вычисляющей покоординатную функцию от элементов массива **`a`**: **`a[i]=F(a[i]);`**

№2

- Элементы массива **`a`** инициализируются **`0`**, массива **`b[0]`**…**`b[n-1]`** – случайными числами от **`0`** до **`n-1`**. Распараллелить цикл

```
for (i=0; i<n; i++) a[b[i]]++;
```

№3

- Распараллелить цикл вычисления суммы

```
for (i=0; i<n; i++) sum+=F(i);
```

№4

- Написать программу, вычисляющую количество простых чисел в диапазоне от 2 до N. Ускорить ее с помощью OpenMP.