Выделение заданного количества тредов для данного параллельного блока осуществляется через num\_threads(x):

#pragma omp parallel num\_threads(4)

{

#pragma omp for

for(…)

{

}

}

Если треды продолжительное время простаивают, например, из-за механизмов синхронизации, то стоит выделить тредов больше, чем исполнителей в системе.

Если мы хотим использовать для нашей задачи не все доступные вычислительные ресурсы, то стоит выделять наоборот меньше тредов, чем доступно.

**omp\_set\_num\_threads(x)** – изменяет переменную окружения, задавая определенное количество потоков.

Большинство современных ОС не гарантирует быстроту выделения тредов. ОС гарантирует только то, что тред будет выделен. Если такие гарантии нужны, например, в runtime-системах, то необходимо резервировать некоторое количество вычислительных ресурсов под гарантии.

**Ограничение OpenMP 2.0** – нельзя использовать беззнаковые переменные в качестве счетчиков цикла.

**Способ условного многопоточного выполнения блока** (если условие не выполняется, то блок выполняется в одном потоке):

#pragma omp parallel if (…)

{

}

**Секции позволяют выполнять разные блоки в разных потоках** (switch по номеру итерации делает то же самое):

#pragma omp parallel

{

#pragma omp sections

{

#pragma omp section

{

}

#pragma omp section

{

}

}

}

**single** позволяет выполнять блок кода одним любым удобным потоком:

#pragma omp parallel

{

#pragma omp single

{

}

}

**main** позволяет выполнять блок кода одним и тем же (главным) потоком:

#pragma omp parallel

{

#pragma omp main

{

}

}

**task** позволяет выделить под этот блок отдельный поток.Удобно, если код явным образом не нарезается на циклы. Ожидание окончания выполнения task неявным образом блокирует основной тред, но вложенные task’и не блокируют.Чтобы основной тред ожидал выполнения внутреннихtaskследует использовать **task wait:**

#pragma omp parallel

{

#pragma omp single

{

#pragma omp task

{

f(x);

}

#pragma omp task wait

}

}

Если необходимо, чтобы на вход task принимала переменную в конкретный момент времени и эта переменная не изменялась извне, то стоит использовать **task first private(x).**

**reduction** позволяет автоматизировать создание локальных переменных для каждого потока, а потом с использованием заданной операции слить все переменные в одну:

int x=0;

#pragma omp parallel for reduction(+:x)

{

for(int i = …)

{

if(…)

x++;

}

}