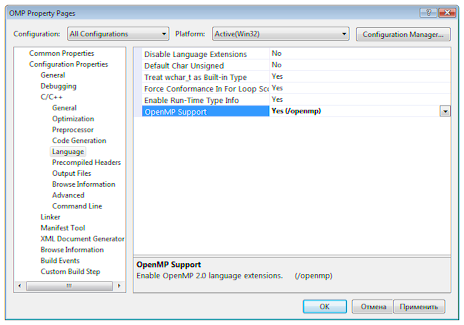
**Задание параллелизма и некоторые другие сведения OpenMP**

OpenMP – это расширения обычных языков программирования (С, C++, Фортран). Распараллеливание в OpenMP выполняется явно при помощи вставки в текст программы специальных директив.

**Как включить OpenMP:**

Visual Studio 2005/2008/2010 **Professional:**

Для вклю­че­ния OpenMP на­жми­те Project → OMP Properties (OMP — имя про­ек­та из мо­их при­ме­ров). Сле­ва ввер­ху по­явив­ше­го­ся ок­на вы­бе­ри­те «All Configurations» и в раз­де­ле Configuration Properties → C/C++ → Language вклю­чи­те «OpenMP Support»:

[](http://iproc.ru/materials/OMPVS/property1.png)

GNU Compiler:

“g++ main.cpp -O2 –o run.exe –fopenmp”

Программа начинается с последовательной области – сначала работает один процесс (нить), при входе в параллельную область порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область.

Для написания эффективной параллельной программы необходимо, чтобы все нити, участвующие в обработке программы, были равномерно загружены полезной работой. Существенным моментом является также необходимость синхронизации доступа к общим данным. Само наличие данных, общих для нескольких нитей, приводит к конфликтам при одновременном несогласованном доступе.

Директивы OpenMP в программах на языке С++ – указаниями препроцессору, начинающимися с **#pragma omp**.

После получения выполняемого файла необходимо запустить его на требуемом количестве процессоров. Для этого обычно нужно задать количество нитей, выполняющих параллельные области программы, определив значение переменной среды OMP\_NUM\_THREADS (или вызвать метод “**void** **omp\_set\_num\_threads(num\_threads)**”).

Функция **omp\_get\_wtime()** возвращает в вызвавшей нити астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом. С++: **double omp\_get\_wtime().** Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка. Гарантируется, что момент времени, используемый в качестве точки отсчета, не будет изменён за время существования процесса. Таймеры разных нитей могут быть не синхронизированы и выдавать различные значения. Функция **omp\_get\_wtick()** возвращает в вызвавшей нити разрешение таймера в секундах. Это время можно рассматривать как меру точности таймера. С++: **double omp\_get\_wtick().**

Параллельная область задаётся при помощи директивы **omp** **parallel**. С++: **#pragma omp parallel [**опция**[[,]** опция**]...].**

Возможные опции:

* **if(**условие**)** – выполнение параллельной области по условию. Вхождение в параллельную область осуществляется только при выполнении некоторого условия. Если условие не выполнено, то директива не срабатывает и продолжается обработка программы в прежнем режиме;
* **num\_threads(**целочисленное выражение**)** – явное задание количества нитей, которые будут выполнять параллельную область; по умолчанию выбирается последнее значение, установленное с помощью функции **omp\_set\_num\_threads()**, или значение переменной **OMP\_NUM\_THREADS**;
* **default(private|firstprivate|shared|none)** – всем переменным в параллельной области, которым явно не назначен класс, будет назначен класс **private**, **firstprivate** или **shared** соответственно; **none** означает, что всем переменным в параллельной области класс должен быть назначен явно; в языке С++ задаются только варианты **shared** или **none**;
* **private(**список**)** – задаёт список переменных, для которых порождается локальная копия в каждой нити; начальное значение локальных копий переменных из списка не определено;
* **firstprivate(**список**)** – задаёт список переменных, для которых порождается локальная копия в каждой нити; локальные копии переменных инициализируются значениями этих переменных в нити-мастере;
* **shared(**список**)** – задаёт список переменных, общих для всех нитей;
* **copyin(**список**)** – задаёт список переменных, объявленных как **threadprivate**, которые при входе в параллельную область инициализируются значениями соответствующих переменных в нити-мастере;
* **reduction(**оператор**:**список**)** – задаёт оператор и список общих переменных; для каждой переменной создаются локальные копии в каждой нити; локальные копии инициализируются соответственно типу оператора (для аддитивных операций – **0** или его аналоги, для мультипликативных операций – **1** или её аналоги); над локальными копиями переменных после выполнения всех операторов параллельной области выполняется заданный оператор; оператор это: для языка С++ – **+**, **\***, **-**,**&**, **|**, **^**, **&&**, **||**; порядок выполнения операторов не определён, поэтому результат может отличаться от запуска к запуску.

При входе в параллельную область порождаются новые **OMP\_NUM\_THREADS-1** нитей, каждая нить получает свой уникальный номер, причём порождающая нить получает номер **0** и становится основной нитью группы («мастером»). Остальные нити получают в качестве номера целые числа с **1** до **OMP\_NUM\_THREADS-1**. Количество нитей, выполняющих данную параллельную область, остаётся неизменным до момента выхода из области. При выходе из параллельной области производится неявная синхронизация и уничтожаются все нити, кроме породившей.

Все порождённые нити исполняют один и тот же код, соответствующий параллельной области. Предполагается, что в SMP-системе нити будут распределены по различным процессорам (однако это, как правило, находится в ведении операционной системы).

Сокращённая запись: если внутри параллельной области содержится только один параллельный цикл, то можно использовать укороченную запись: **parallel for.**

**Параллельные циклы.** Если в параллельной области встретился оператор цикла, то он будет выполнен всеми нитями текущей группы, то есть каждая нить выполнит все итерации данного цикла. Для распределения итераций цикла между различными нитями можно использовать директиву **for** (**do ... [end do]**). С++: **#pragma omp for [**опция **[[,]** опция**]...].** Эта директива относится к идущему следом за данной директивой блоку, включающему операторы **for** (**do**).

Возможные опции:

* **private(**список**)** – задаёт список переменных, для которых порождается локальная копия в каждой нити; начальное значение локальных копий переменных из списка не определено;
* **firstprivate(**список**)** – задаёт список переменных, для которых порождается локальная копия в каждой нити; локальные копии переменных инициализируются значениями этих переменных в нити-мастере;
* **lastprivate(**список**)** – переменным, перечисленным в списке, присваивается результат с последнего витка цикла;
* **reduction(**оператор**:**список**)** – задаёт оператор и список общих переменных; для каждой переменной создаются локальные копии в каждой нити; локальные копии инициализируются соответственно типу оператора (для аддитивных операций – **0** или его аналоги, для мультипликативных операций – **1** или её аналоги); над локальными копиями переменных после завершения всех итераций цикла выполняется заданный оператор; оператор это: для языка Си – **+**, **\***, **-**, **&**, **|**, **^**, **&&**, **||**; порядок выполнения операторов не определён, поэтому результат может отличаться от запуска к запуску;
* **schedule(type[, chunk\_size])** – опция задаёт, каким образом итерации цикла распределяются между нитями;
* **collapse(n)** – опция указывает, что **n** последовательных тесно вложенных циклов ассоциируется с данной директивой; для циклов образуется общее пространство итераций, которое делится между нитями; если опция **collapse** не задана, то директива относится только к одному непосредственно следующему за ней циклу;
* **ordered** – опция, говорящая о том, что в цикле могут встречаться директивы **ordered**; в этом случае определяется блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле;
* **nowait** – в конце параллельного цикла происходит неявная барьерная синхронизация параллельно работающих нитей: их дальнейшее выполнение происходит только тогда, когда все они достигнут данной точки; если в подобной задержке нет необходимости, опция **nowait** позволяет нитям, уже дошедшим до конца цикла, продолжить выполнение без синхронизации с остальными. Если директива **end do** в явном виде не указана, то в конце параллельного цикла синхронизация все равно будет выполнена.

На вид параллельных циклов накладываются достаточно жёсткие ограничения. В частности, предполагается, что корректная программа не должна зависеть от того, какая именно нить какую итерацию параллельного цикла выполнит. Нельзя использовать побочный выход из параллельного цикла. Размер блока итераций, указанный в опции **schedule**, не должен изменяться в рамках цикла.

В опции **schedule** параметр **type** задаёт следующий тип распределения итераций:

* **static** – блочно-циклическое распределение итераций цикла; размер блока – **chunk\_size**. Первый блок из **chunk\_size** итераций выполняет нулевая нить (мастер), второй блок – следующая и т.д. до последней нити, затем распределение снова начинается с нулевой нити. Если значение **chunk\_size** не указано, то всё множество итераций делится на непрерывные куски примерно одинакового размера (конкретный способ зависит от реализации), и полученные порции итераций распределяются между нитями;
* **dynamic** – динамическое распределение итераций с фиксированным размером блока: сначала каждая нить получает **chunk** итераций (по умолчанию **chunk=1**), та нить, которая заканчивает выполнение своей порции итераций, получает первую свободную порцию из **chunk** итераций. Освободившиеся нити получают новые порции итераций до тех пор, пока все порции не будут исчерпаны. Последняя порция может содержать меньше итераций, чем все остальные;
* **guided** – динамическое распределение итераций, при котором размер порции уменьшается с некоторого начального значения до величины **chunk** (по умолчанию **chunk=1**) пропорционально количеству ещё не распределённых итераций, делённому на количество нитей, выполняющих цикл. Размер первоначально выделяемого блока зависит от реализации. В ряде случаев такое распределение позволяет аккуратнее разделить работу и сбалансировать загрузку нитей. Количество итераций в последней порции может оказаться меньше значения **chunk;**
* **auto** – способ распределения итераций выбирается компилятором и/или системой выполнения. Параметр **chunk** при этом не задаётся;
* **runtime** – способ распределения итераций выбирается во время работы программы по значению переменной окружения **OMP\_SCHEDULE**. Параметр **chunk** при этом не задаётся.

Использованная литература:

Антонов А.С. Параллельное программирование с использованием технологии MPI. М.: Изд-во МГУ. 2004. 71 с. [http://parallel.ru/info/parallel/ antonov/](http://parallel.ru/info/parallel/%20antonov/)

<http://www.openmp.org/mp-documents/spec30.pdf>

<http://iproc.ru/programming/openmp-visual-studio/>

<https://ru.wikipedia.org/wiki/OpenMP>