Распараллеливание циклов в OpenMP

Марчевский И.К., Попов А.Ю.

МГТУ им. Н.Э. Баумана

Циклы и требования к ним в OpenMP

Обработка данных с помощью циклов занимает большую часть времени в инженерных и научных приложениях. В OpenMP это — основной ресурс распараллеливания программы (в C/C++ — циклы `for`, в Fortran — циклы `do`). К ним предъявляются строгие требования, чтобы произвольная итерация могла быть выполнена некоторым потоком независимо от других, а также заранее было известно число выполняемых итераций, в том числе

- явно заданы границы цикла (инициализирующая часть, условие исполнения, изменение переменной цикла) либо используется цикл, привязанный к диапазону (range-based);
- шаг переменной цикла фиксирован, причем при условии вида `i != var` шаг равен только ± 1;
- переменная цикла не изменяется в теле цикла, отсутствуют операторы `continue` и `break`.

Управление распределением итераций I

Распараллеливание цикла с указанием способа распределения итераций по потокам — директива
`#pragma omp for schedule(вид[,block])`

Опция `schedule` допускает следующие виды распределения:

`static` — каждому потоку последовательно выделяется `block` итераций, затем процедура повторяется, пока не закончатся итерации цикла (последний блок может быть меньшего размера). Если не указано `block`, оно вычисляется так, чтобы каждый поток получил приблизительно одинаковое число итераций и чтобы каждый поток получил хотя бы одну;

`dynamic` — каждый поток исполняет `block` итераций (по умолчанию — 1), затем запрашивает новые `block` (кроме последнего блока, который может быть меньше);

Управление распределением итераций II

`guided` — каждый поток исполняет некоторый блок итераций, затем запрашивает новый, размер которого уменьшается по мере исполнения цикла. При этом наименьший размер выделяемого блока не может быть меньше `block` (по умолчанию 1);

`auto` — распределение зависит от конкретной реализации OpenMP;

`runtime` — распределение и его параметры определяются во время работы программы с использованием переменной среды `OMP_SCHEDULE`.

Начиная с версии OpenMP 4.5, можно управлять выделением итераций потокам: `#pragma omp for schedule(`*`modifier`*`: вид[,block])`, где `modifier` может принимать значение `monotonic` (в логическом порядке) либо `nonmonotonic` (в произвольном порядке).

Опции для распараллеливания циклов

- Директива `#pragma omp ordered` указывает, что заключенные в нее действия должны в ходе параллельного исполнения цикла выполняться в таком порядке, как если бы цикл был последовательным. Директива у цикла при этом должна иметь опцию `ordered`.
- Опция `collapse(n)` у цикла доступна, начиная с версии OpenMP 3.0, и определяет, сколько тесновложенных циклов будут объединены в общее пространство итераций. Объединенный набор итераций будет учитываться при распределении по потокам. В противном случае ($n = 1$) распараллеливается только внешний цикл, а итерации внутреннего исполняются в последовательном режиме каждым потоком.
- Опция `lastprivate(var)` позволяет записать в переменной в общей памяти значение, присвоенное на последней итерации цикла.
- Опция `nowait` отменяет барьерную синхронизацию в конце цикла. В этом случае при продолжении параллельной секции освободившиеся потоки перейдут к выполнению дальнейших операций.

Секции

В OpenMP имеется возможность параллельного исполнения логически не связанных блоков кода (не являющихся итерациями цикла) — директива `#pragma omp sections` в рамках параллельной области.

- Каждая секция задается с помощью `#pragma omp section` и выполняется одним потоком (перед первой секцией необязательна).
- Для директивы `sections` доступны опции по аналогии с `for` — `private`, `firstprivate`, `lastprivate`, `reduction`.
- Если секций меньше числа потоков, то некоторые потоки будут простаивать; если больше — некоторые потоки получат более одной секции.
- После выполнения всех секций — синхронизация, если только не задана опция `nowait`.

Задачи (tasks)

Задачи доступны в OpenMP, начиная с версии 3.0, и предоставляют возможность параллельной работы потоков, более свободную по сравнению с циклами — task-parallel-подход в противоположность data-parallel.

- Задача порождается потоком с помощью директивы `#pragma omp task` и исполняется им же (сразу либо исполнение откладывается), если не указана опция `untied`.
- По умолчанию переменные в задачах относятся к типу `firstprivate`, для совместного использования их надо указывать как `shared`.
- При необходимости ожидания окончания исполнения всех задач текущего потока используется директива `#pragma omp taskwait`.
- Опция `if(условие)` определяет, будет ли порождена задача либо блок кода будет исполнен немедленно текущей нитью.
- Механизм задач удобен, когда сложно прогнозировать объем операций, в частности, при обращении к рекурсии, обходе деревьев и т.д.

Директива threadprivate

Для независимой работы с глобальными и статическими переменными, которые изначально являются общими, используется директива `#pragma omp threadprivate(varname)`.

- С помощью директивы `threadprivate` в каждой нити создается свой экземпляр переменной, которым можно пользоваться в разных параллельных секциях (в отличие от private-переменных). На последние накладываются определенные ограничения — количество потоков должно быть одинаковым, отсутствие динамического параллелизма и пр.
- Можно инициализировать такие переменные значением из мастер-потока с помощью опции `copyin(varname)` директивы `parallel` либо из одного из потоков с помощью опции `copyprivate(varname)` директивы `single`.

Альтернативы OpenMP I

Следует учитывать, что в поздних версиях OpenMP доступна возможность переноса вычислительной нагрузки (offloading) на различные ускорители, в т.ч. GPU.

1. OpenCL (Open Computing Language) — свободный стандарт для написания параллельных многопоточных программ на языке C/C++ для широкого класса устройств — CPU, GPU, FPGA (ПЛИС), суперкомпьютеры, встраиваемые устройства и пр.
   Предполагает определенную иерархию исполнителей, которые включают в себя несколько вычислительных блоков (compute units), состоящих из большого числа обрабатывающих блоков (processing elements). Декларируется высокая степень переносимости кода.
   Изначально предложен в 2009 г. компанией Apple Inc., сейчас развивается группой Khronos Group, включающей также AMD, ARM, Google, NVIDIA, Qualcomm и другие компании.

Альтернативы OpenMP II

2. DVM-система (Distributed virtual machine, distributed virtual memory) — разработка ИМП им. М.В. Келдыша и МГУ им. М.В. Ломоносова, система параллельного программирования на языках C и Fortran, в т.ч. и с различными технологиями (OpenMP, MPI, CUDA) — версия DVM-H (heterogeneous). Включает в себя компилятор и библиотеку.
3. Cilk (а также Cilk++, Intel Cilk Plus) — технология многопоточного программирования, в которой программист указывает возможные места параллельного исполнения, а система исполнения сама управляет распараллеливанием, распределением и перераспределением (в т.ч. work-stealing) нагрузки и пр. Является расширением C/C++, причем в частном случае программа исполняется одним потоком. Предложена в 1994 г. в MIT, затем в 2008 г. появился Cilk++. В 2010 г. проект был поглощен Intel и развивался как Intel Cilk Plus до 2017 г., когда Intel отказалась от него. В настоящее время снова поддерживается MIT под названием OpenCilk.

Альтернативы OpenMP III

4. Intel TBB (Thread Building Blocks) — библиотека Intel для параллельного программирования на многоядерных процессорах. В отличие от Cilk и OpenMP, не использует расширений языка, а только функции и механизмы самой библиотеки, в связи с чем не требуется поддержка компилятором.
   Intel TBB предоставляет как низкоуровневые механизмы (управление потоками, мьютексами, atomic-операции и пр.), так и инструменты для работы с памятью (собственные контейнеры — `concurrent_vector`, `concurrent_map`, `concurrent_set` и др.), а также высокоуровневые параллельные алгоритмы, такие как параллельная сортировка и параллельный конвейер.
   Версия 1.0 предложена в 2006 г., сейчас TBB является частью пакета oneAPI. Распространяется свободно и с открытым исходным кодом.