



Казанский федеральный
УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Синхронизация в OpenMP

Эдуард Храмченков

Синхронизация

- ▶ Выполнение кода потоками недетерминированно – в один и тот же момент времени потоки могут находиться на разных стадиях
- ▶ Синхронизация гарантирует, что к определенной точке (барьеру синхронизации) все потоки будут пребывать в одинаковом состоянии



Синхронизация

- ▶ Иногда синхронизация необходима для корректного выполнения алгоритма, работы с разделяемыми объектами
- ▶ Неявная синхронизация – в конце регионов параллельного кода
- ▶ Существуют ситуации, когда барьер синхронизации требуется указать явным образом



Директива `barrier`

- ▶ `#pragma omp barrier`
- ▶ Создает точку синхронизации, у которой потоки ждут до тех пор, пока последний из них не завершит выполнение
- ▶ К барьеру должны подходить все потоки или ни один из них
- ▶ Последовательность параллельных регионов и барьеров должна быть одинакова для всех потоков



Пример 1

- ▶ Барьер синхронизации часто используется, когда надо иметь гарантированное значение разделяемой переменной, после ее обработки несколькими потоками
- ▶ Если значение переменной будет использоваться в дальнейшем без синхронизации, это может привести к неопределенному поведению



Директива `ordered`

- ▶ `#pragma omp ordered`
- ▶ Позволяет выполнить последовательный код внутри параллельного цикла
- ▶ Все остальное тело цикла, не включенное в блок `ordered` выполняется параллельно
- ▶ Помогает определить наличие «условия гонки», а также структурировать вывод данных потоками



Директива critical

- ▶ `#pragma omp critical [(имя)]`
- ▶ Создает критическую область или критическую секцию – регион кода, который в каждый момент времени может выполняться только одним потоком
- ▶ Имя критической секции должно быть уникальным в области видимости всей программы



Директива critical

- ▶ Как правило, критическая секция служит для корректного доступа к разделяемому ресурсу
- ▶ Критическая секция не гарантирует конкретной последовательности выполнения кода потоками
- ▶ Критическая секция очень дорогая в плане производительности конструкция, применять лишь при необходимости



Пример 2

- ▶ Вывод в стандартный поток без коллизий
- ▶ Суммирование «в лоб» без критической секции дает некорректный результат
- ▶ Суммирование с критической секцией – медленное
- ▶ Выход – считать локальные суммы, потом складывать их
- ▶ Если потоков много локальные суммы тоже нужно считать в критической секции



Директива `atomic`

- ▶ `#pragma omp atomic`
- ▶ Позволяет нескольким потокам работать с разделяемыми данными корректным образом
- ▶ Может служить более эффективной альтернативой критическим секциям
- ▶ Применяется только к выражению с присваиванием идущему сразу за директивой



Директива `atomic`

- ▶ Платформа на которой компилируется код, должна поддерживать атомарные операции
- ▶ Если поток выполняет атомарную операцию над общими данными, ни один другой поток не может вмешаться
- ▶ Корректность работы гарантируется для общей переменной стоящей слева от `=`



Директива `atomic`

- ▶ Выражение должно быть вида
 - `++x` или `x++` или `--x` или `x--`
 - `x binop = expr;`
 - `x = x binop expr;`
 - `x = expr binop x;`
- ▶ `binop` есть `+`, `*`, `-`, `/`, `&`, `^`, `|`, `<<`, `>>`
- ▶ `x` есть l-value скалярного типа
- ▶ `expr` возвращает скалярный тип и не зависит от `x`



Пример 3

- ▶ Применение `atomic` для суммирования массивов
- ▶ Сравнение эффективности суммирования массивов с использованием `critical` и `atomic`



Редукция в OpenMP

- ▶ Для эффективного суммирования или инкрементации общей переменной следует использовать операцию редукции
- ▶ Используется с директивой `for`
- ▶ Не требует использования критических секций и атомарных операций
- ▶ В начале редукции каждый поток получает частную копию переменной, в конце все копии автоматически аккумулируются



Редукция в OpenMP

- ▶ `#pragma omp parallel for reduction(op:list)`
- ▶ `op` – один из допустимых операторов
- ▶ `list` – список переменных, где будет происходить аккумуляция

Оператор	Начальное значение переменной
<code>+, -, , ^, , &</code>	0
<code>*, &&</code>	1



Пример 4

- ▶ Применение редукции для суммирования массива



Директива master

- ▶ `#pragma omp master`
- ▶ Гарантирует исполнение следующего за ней блока кода только мастер-поток
- ▶ В случае если директива используется для задания начальных значений, после нее рекомендуется явно указывать барьер синхронизации
- ▶ Во многих случаях заменяема `#pragma omp single`



Директива flush

- ▶ `#pragma omp flush [(переменная, ...)]`
- ▶ Принудительно обновляет значения указанных общих переменных для потока
- ▶ Служит для гарантии того, что у всех потоков будут одинаковые значения разделяемых переменных
- ▶ Если переменные не указаны явно операция будет выполнена для всех переменных доступных потоку



Замки в OpenMP

- ▶ OpenMP предоставляет возможность использовать замки, что позволяет создавать более гибкие конструкции, нежели `critical` и `atomic`
- ▶ Работа с замками осуществляется при помощи собственных функций OpenMP
- ▶ Семантика использования соответствует логике семафора



Замки в OpenMP

- ▶ `omp_init_lock()` – инициализация замка; после вызова замок не установлен
- ▶ `omp_destroy_lock()` – уничтожение замка; замок должен быть отключен перед этим
- ▶ `omp_set_lock()` – пытается установить замок, если другой поток уже установил замок, ждет пока он освободится и потом устанавливает его



Замки в OpenMP

- ▶ `omp_unset_lock()` – снятие замка; эту функцию должен вызывать тот же самый поток, который устанавливал замок, иначе возникает неопределенность
- ▶ `omp_test_lock()` – пытается установить замок; если замок установлен другим потоком, возвращает 0, иначе устанавливает замок и возвращает 1



Пример 5

- ▶ Использование замков в OpenMP
- ▶ Ситуация deadlock – каждый поток ждет освобождения замка, установленного другим потоком





Казанский федеральный
УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru