Стандарт OpenMP

https://www.openmp.org

Compilers & Tools

Опции директивы **parallel**

`num_threads` (*integer-expression*) — задает количество потоков.

`private` (*variable-list*) — список переменных, которые станут локальными.

`firstprivate` (*variable-list*) — аналогично с инициализацией значениями из основного потока.

`reduction` (*operator* `:` *variable-list*)

Редукция. В параллельной области переменная из списка превращается в набор локальных копий, к которым после завершения будет применена операция, а результат помещен в переменную основного потока.

Распределение работы

- `for` directive
- `sections` directive
- `single` directive

Директива **for**

относится к идущему следом оператору for.

for([***целочисленный тип***] *i* =…;…;…)

Если стоит перед вложенным циклом, то воздействует только на внешний.

Опции директивы **for**

- *private*
- *firstprivate*
- *reduction*
- *lastprivate* – задает список локальных переменных, которые после окончания цикла будут иметь значение с логически последней итерации.
- *nowait* – отказ от барьерной синхронизации.
- *schedule* – задает способ распределения итераций.

Директива **parallel for**

- сокращенный вариант.

Применимы любые опции директив *parallel* и *for* за исключением *nowait*.

Директива **sections / parallel sections**

Определяет набор независимых секций кода, каждая из которых выполняется отдельным потоком.

Секции отмечаются директивами

#pragma omp section

Опции:

- *private*
- *firstprivate*
- *lastprivate*
- *reduction*
- *nowait*

Директива **single**

определяет блок который будет выполняться только в одном потоке, если не указана опция **nowait** остальные будут ожидать завершения его работы.

Опции:

- *private*
- *firstprivate*
- *nowait*
- *copyprivate*(список переменных)

Опция **copyprivate** (список переменных)

После выполнения блока, помеченного директивой **single**, значения переменных будут скопированы в одноименные частные переменные, других потоков данной параллельной области.

Не может использоваться совместно с **nowait**;

Директива **master**

определяет блок, будет выполняться ***основным*** потоком.

Остальные данный блок пропускают, барьерной синхронизации не производится.

Средства синхронизации

Директива atomic

Гарантирует корректную работу с общей переменной.

```
#pragma omp atomic new-line
    expression-stmt
```

The expression statement must have one of the following forms:

```
x binop= expr
x++
++x
x--
--x
```

Директива **barrier**

Явная барьерная синхронизация
всех потоков параллельной области.

Директива **critical**

```
#pragma omp critical [(name)] new-line
        structured-block
```

определяет критическую секцию.

В критическую секцию с заданным именем могут входить различные блоки операторов.

Побочные выходы (`break; return`) ***из критической секции запрещены***.

Блокировки (**locks**)

Переменные типов
omp_lock_t и ***omp_nest_lock_t***

- аналоги мьютексов (обычного и рекурсивного).

MPI (Message Passing Interface) – интерфейс передачи сообщений

Программный интерфейс для передачи информации между процессами.

Существует большое количество бесплатных и коммерческих реализаций для различных платформ.

https://www.mpi-forum.org/

https://parallel.ru/tech/tech_dev/mpi.html

Используется при разработке программ для кластеров и суперкомпьютеров.

Литература

1. А.С. Антонов. Параллельное программирование с использованием технологии MPI.

2. Корнеев В.Д. Параллельное программирование кластеров: учеб. пособие–Новосибирск: Изд-во НГТУ, 2008.

стр. 7-143.

3. Параллельное и распределенное программирование с использованием C++. Камерон Хьюз, Трейси Хьюз, 2004

Глава 9 - стр. 312-336.

Практика

Для Microsoft Visual C++

Скачать и установить библиотеку Microsoft MPI;

подключить mpi.h и msmpi.lib.

Команда **для запуска** ***k*** процессов выглядит так:

```
mpiexec -n k MyApplication.exe
```

Инициализация и завершение

выполняется не более одного раза.

```
int MPI_Init(int* argc, char*** argv)
```

Обращение к функциям MPI между этими вызовами.

```
int MPI_Finalize(void)
```

Все функции MPI возвращают MPI_SUCCESS или код ошибки

(кроме функций замере времени:
`double MPI_Wtime(void)`
`double MPI_Wtick(void)`).

Коммуникатор

– параметр всех функций приёма/передачи.
Определяет группу процессов в которой осуществляется взаимодействие.

При старте программы считается, что все порожденные процессы работают в рамках глобального коммуникатора **MPI_COMM_WORLD**.

Номер процесса

– целое неотрицательное число уникальное ***в рамках*** каждой ***группы***, в которую процесс входит.

```
int MPI_Comm_rank( MPI_Comm comm, int* rank)
```

записывает номера процесса в коммуникаторе **`comm`** по адресу **`rank`**.

```
int MPI_Comm_size( MPI_Comm comm, int* size)
```

записывает число процессов в коммуникаторе `comm` по адресу `size`.

Взаимодействия MPI процессов.
Виды взаимодействий

1. Парные взаимодействия.
2. Коллективные взаимодействия.

Способы взаимодействий MPI и процесса

Блокирующий – ***управление*** вызывающему процессу ***возвращается*** только ***после того, как*** данные приняты / переданы / скопированы во временный буфер.

Неблокирующий – ***управление возвращается немедленно***, фактическая приемопередача происходит в фоне..

Парные взаимодействия.
Режимы передачи сообщения

1. Стандартный (асинхронный режим).
2. Синхронный.
3. Буферизированный.
4. По готовности.

Коллективные взаимодействия

- Барьерная синхронизация
- Широковещательный обмен
- Операции редукции
- Сборка данных
- Рассылка

Выполняются только в блокирующем режиме.