Вопросы по курсу ТПП

Технология программирования OpenMP

1. В чем состоят основы технологии OpenMP?

В составе технологии OpenMP можно выделить:
- Директивы,
- Библиотеку функций,
- Набор переменных окружения

2. В чем состоят основные преимущества и недостатки технологии OpenMP?

Понятие гонки потоков, когда результат операции зависит от порядка выполнения.

Необходимо обеспечить взаимное исключение потоков при работе с общими данными

Так же может быть необходимо выполнение синхронизации вычислений

3. Что понимается под параллельной программой в рамках технологии OpenMP?

Технология OpenMP (Open Multi-Processing) позволяет разделять код на несколько *потоков,* в специально указываемых при помощи директив местах кода – *параллельных фрагментах.*

В технологии OpenMP, потоки одной и той же параллельной программы выполняются в общем адресном пространстве, что обеспечивает возможность использования общих данных для параллельно выполняемых потоков.

4. Какие проблемы возникают при использовании общих данных в параллельно выполняемых потоках?

Понятие гонки потоков, когда результат операции зависит от порядка выполнения.

Необходимо обеспечить взаимное исключение потоков при работе с общими данными

Так же может быть необходимо выполнение синхронизации вычислений

5. Какой формат записи директив OpenMP?

#pragma omp <имя_директивы> [<параметр>[[,] <параметр>]...]

6. В чем состоит назначение директивы parallel?

Выделение параллельных участков

7. В чем состоят понятия фрагмента, области и секции параллельной программы?

- Параллельный фрагмент (parallel construct) – блок программы, управляемый директивой parallel.
- Параллельная область (parallel region) – параллельно выполняемые участки программного кода, динамически возникающие в результате вызова функций из параллельных фрагментов.
- Параллельная секция (parallel section)– часть параллельного фрагмента, выделяемая для параллельного выполнения при помощи директивы section

8. Как осуществляется распараллеливание циклов в OpenMP? Какие условия должны выполняться, чтобы циклы могли быть распараллелены?

Для распараллеливания циклов в OpenMP применяется директива for:
#pragma omp for [<параметр> ...] <цикл_for>
Между итерациями цикла не должно быть информационной зависимости
Также цикл должен иметь вид цикла со счётчиком

9. Какие возможности имеются в OpenMP для управления распределением итераций циклов между потоками?

Для этого существует параметр schedule директивы for:

- Static – статический способ распределения итераций, итерации цикла делятся на блоки размера chunk
- Dynamic – динамический способ распределения, первая итерация цикла раздаёт каждому потоку по chunk, дальнейшее выделение происходит при окончании процессом текущей итерации
- Guided – управляемый способ распределения итераций, начальный размер блока определяется параметром среды реализации OpenMP, а затем уменьшается экспоненциально, но не меньше chunk
- Runtime – выбор конкретного способа осуществляется в момент начала выполнения программы в соответствии со значением OMP_SCHEDULE. Это позволяет менять код программы без повторной компиляции и сборки

10. Как определяется порядок выполнения итераций в распараллеливаемых циклах?

По умолчанию потоки ожидают окончания выполнения итераций цикла, конец цикла представляет собой некоторый барьер, который можно отменить параметром nowait директивы for

11. Как определяются общие и локальные переменные потоков?

Задаются явно в параметрах директивы parallel, по умолчанию переменные общие

12. Что понимается под операцией редукции?

Редукция – коллективная операция над локальными переменными.
Reduction(operator: list) list – задаёт набор локальных переменных для которых должна быть выполнена коллективная операция

13. Какие возможности предоставляет технология OpenMP для управления распределенными вычислениями?

Распределение вычислительной нагрузки между потоками с помощью директивы sections (распараллеливание по задачам)
При помощи директивы sections выделяется программный код, который далее будет разделен на параллельно выполняемые секции

14. Какие директивы синхронизации поддерживаются в OpenMP?

Обеспечение атомарности операции (atomic)
Использование критических секций (critical)
Применение переменных семафорного типа
Определение одно потоковых участков (single и master)
Выполнение барьерной синхронизации (barrier)

Синхронизация состояния памяти (flush)

15. Назначение директив shared, private, default, firstprivate, lastprivate, threadprivate, copyin.

threadprivate – постоянно существующие локальные переменные

copyin – указание переменных в которые нужно сохранить результат параллельного фрагмента программы

Технология программирования Message Passing Interface

1. В чем состоят основы технологии MPI?

2. В чем состоят основные преимущества и недостатки технологии MPI?

3. Что понимается под параллельной программой в рамках технологии MPI?

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться как на разных процессорах, так и на одном. Каждый процесс параллельной программы порождается на основе копии одного и того же программного кода (модель SPMP). Все процессы программы последовательно перенумерованы от 0 до p-1, где p есть общее количество процессов. Номер процесса именуется рангом процесса.

4. Как происходит инициализация и завершение MPI программ?

```
int MPI_Init(int *agrc, char ***argv);
int MPI_Finalize(void);
```

5. Как происходит передача и прием сообщений MPI программе? Виды точечных взаимодействий.

Различают парные (point-to-point) и коллективные (collective):

- Блокирующие (MPI_Send/MPI_Recv)
- Передача с буферизацией (MPI_Bsend)
- Передача с синхронизацией (MPI_Ssend)
- Передача по готовности (MPI_Rsend)

6. Как происходит передача данных от одного процесса всем?

С помощью коллективных операций (например, MPI_Bcast (broadcast))

7. Как происходит передача данных от всем процессов одному?

С помощью коллективных операций (например, MPI_Gather)

Также с помощью операции редукции

8. Какие используются в MPI для синхронизации вычислений?

Для синхронизации используются барьеры с указанием коммуникатора MPI_Barrier(MPI_Comm comm)

9. Как организуется неблокирующий обмен данными между процессами?

Путем добавления префикса I (Immediate) MPI_Isend

10. Как организуется одновременное выполнение прием и передачи данных?

С помощью функции MPI_Sendrecv

11. Прием и передача по шаблону.

???

12. Коллективные взаимодействия процессов

???

13. Интер- и интра-коммуникаторы

Интер-коммуникаторы – глобальные
Интра-коммуникаторы – для передачи внутри группы

14. Основные функции работы с коммуникаторами

MPI_Comm_dup – дублирование уже существующего
MPI_Comm_create – создание нового коммуникатора

15. (создание, разбиение, дублирование, удаление)

16. Информационные функции для работы с группами

Определение количества процессов в группе и их рангов

Многопроцессорные системы

1. Для чего нужна распределенная обработка информации?

Распределенная обработка информации позволяет эффективно использовать ресурсы нескольких компьютеров или процессоров для решения сложных задач. Это увеличивает производительность и общую мощность системы.

2. В каких задачах применима распределенная обработка информации?

Распределенная обработка применяется в задачах, где требуется обработка больших объемов данных, выполнение сложных вычислений, симуляции, анализ больших наборов данных (например, в области научных исследований, финансов, обработки изображений и видео).

3. Типы параллелизма:

Параллелизм данных: Раздельная обработка независимых порций данных.

Параллелизм задач: Выполнение различных задач одновременно.

Параллелизм памяти: Работа с разными областями памяти одновременно.

4. Классификация по Флинну:

SISD (Single Instruction, Single Data): Одна инструкция обрабатывается на одном наборе данных.

SIMD (Single Instruction, Multiple Data): Одна инструкция обрабатывает несколько наборов данных.

MISD (Multiple Instruction, Single Data): Несколько инструкций обрабатывают одинаковый набор данных.

MIMD (Multiple Instruction, Multiple Data): Различные инструкции обрабатывают различные наборы данных.

5. Основные типы архитектуры систем параллельной обработки:

Многопроцессорные системы (Multiprocessor Systems): Несколько процессоров, работающих над общей задачей.

Кластерные системы (Cluster Systems): Несколько компьютеров, объединенных для совместной работы.

Системы с распределенной памятью (Distributed Memory Systems): Каждый процессор имеет свою собственную память.

Системы с общей памятью (Shared Memory Systems): Все процессоры имеют доступ к общей памяти.

6. Мультипроцессорные системы с общей и распределенной памятью:

С общей памятью: Процессоры имеют общий доступ к общей памяти.

С распределенной памятью: Каждый процессор имеет свою собственную память.

7. Средства параллельного программирования:

MPI (Message Passing Interface): Для обмена данными между процессорами.

OpenMP: Директивы для добавления параллельности в код.

CUDA и OpenCL: Для программирования графических процессоров.

8. Основные характеристики параллельных программ:

Уровень параллелизма: Количество задач, выполняемых одновременно.

Гранулярность: Размер каждой подзадачи.

Скорость выполнения: Время выполнения программы на параллельной системе.

9. Линейное и «сверхлинейное» ускорение:

Линейное ускорение: Ускорение, пропорциональное количеству процессоров.

Сверхлинейное ускорение: Ускорение больше, чем пропорционально количеству процессоров, обычно из-за оптимизации ресурсов.

10. Закон Амдала:

Иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей

11. Тесты LINPACK:

Тесты производительности LINPACK служат для измерения вычислительной производительности компьютеров при обработке чисел с плавающей запятой

12. Информационные зависимости:

Ситуации, когда порядок выполнения операций влияет на результат программы.

13. Лемма Брента:

Зная время выполнения на системе с достаточным числом процессоров, можно рассчитать время выполнения на системе с n процессорами.

$t + (q - t) / n$

Кластерные вычисления

1. Что такое кластер, параллельное приложение:

Кластер: Группа связанных компьютеров, работающих вместе как единая система, обычно для повышения производительности и обработки больших объемов данных.

Параллельное приложение: Программа, спроектированная для эффективного использования параллельной обработки, выполняющаяся на нескольких процессорах или компьютерах.

2. Основные задачи и проблемы в кластерных вычислениях:

Задачи: Увеличение производительности, обработка больших данных, распределенные вычисления.

Проблемы: Управление ресурсами, синхронизация, отказоустойчивость, эффективная коммуникация.

3. Ключевые даты развития кластерных вычислений:

Развитие кластерных вычислений началось в 1980-х годах, с появлением первых кластеров. Значительный рост был зафиксирован в 1990-х и начале 2000-х годов.

4. Архитектура типового кластера на примере Microsoft High Performance Computing Server 2008:

Этот сервер включает в себя управляющий узел, где работает управляющий программный продукт, и вычислительные узлы, выполняющие вычислительные задачи. Взаимодействие осуществляется через сетевые протоколы.

5. Сетевые топологии кластерных систем:

Шина (Bus): Все узлы подключены к общей шине.

Кольцо (Ring): Каждый узел подключен к двум соседним узлам, формируя кольцо.

Звезда (Star): Все узлы подключены к центральному узлу (порой называемому коммутатором).

Многомерная топология (Mesh, Torus): Комбинация различных соединений для обеспечения более сложной структуры.

6. Виды заданий в кластерных системах:

Разделенные задачи (Divisible Load): Задачи, которые могут быть разделены на подзадачи.

Неразделенные задачи (Indivisible Load): Задачи, которые не могут быть разделены и должны быть выполнены целиком на одном узле.

7. Особенности построения кластерных систем с общей и распределенной памятью:

С общей памятью: Узлы имеют доступ к общей памяти, требуется более сложная синхронизация.

С распределенной памятью: Узлы имеют свою собственную память, обмен данных происходит через сеть.

8. Характеристики производительности механизмов обмена:

Пропускная способность (Bandwidth): Количество данных, которые могут быть переданы за единицу времени.

Задержка (Latency): Время, требуемое для передачи данных от одного узла к другому.

9. Жизненный цикл кластерных систем:

Включает в себя проектирование, развертывание, масштабирование, обслуживание и, при необходимости, обновление и замену узлов.

10. Отладка параллельной программы:

Требует инструментов для анализа и отладки параллельного кода, таких как отслеживание потоков, профилирование и анализ зависимостей данных.

11. Оптимизация параллельной программы:

Включает в себя использование эффективных алгоритмов, балансировку нагрузки, минимизацию обмена данными и оптимизацию доступа к памяти.

12. Цикл разработки многопоточных приложений:

Включает в себя планирование, проектирование, реализацию, отладку, тестирование и оптимизацию.