Лабораторная работа №1

Введение в формальную верификацию ПО Проектирование моделей протокола синхронизации

Цель работы: изучение методов формальной верификации последовательных, многопоточных, многопроцессных и асинхронных программ предоставляет в распоряжение обучаемого набор теоретических и практических методов, позволяющих осуществлять проверку корректности различных программ на заданном множестве входных данных или на множестве возможных вычислений программы.

Продолжительность работы: 4 часа.

Теоретические сведения

Механизмы синхронизации

Для повышения производительности вычислительных систем и облегчения задачи программистов существуют специальные механизмы синхронизации: семафоры, мониторы Хора, очередей сообщений.

Семафоры

Одним из первых механизмов, предложенных для синхронизации поведения процессов, стали семафоры, концепцию которых описал Дейкстра (Dijkstra) в 1965 году.

Семафор — это объект синхронизации, который позволяет нескольким потокам одновременно получать доступ к одному и тому же ресурсу (например, к файлу или разделу жесткого диска). Он работает по принципу "разделяй и властвуй", позволяя эффективно управлять ресурсами в многопоточных приложениях.

В программировании семафоры используются для управления доступом к общим ресурсам, таким как файлы, процессы, каналы связи и т.д. Они могут быть использованы для обеспечения безопасности данных, предотвращения конфликтов при одновременном доступе нескольких потоков к одному ресурсу, а также для оптимизации производительности приложений.

Семафор представляет собой целую переменную, принимающую неотрицательные значения, доступ любого процесса к которой, за исключением момента ее инициализации, может осуществляться только через две атомарные операции: P (от датского слова proberen – проверять) и V (от verhogen – увеличивать). Классическое определение этих операций выглядит следующим образом:

```
P(S):   пока S == 0 процесс блокируется;
        S = S – 1;
V(S):   S = S + 1;
```

Эта запись означает следующее: при выполнении операции P над семафором S сначала проверяется его значение. Если оно больше 0, то из S вычитается 1. Если оно меньше или равно 0, то процесс блокируется до тех пор, пока S не станет больше 0, после чего из S вычитается 1. При выполнении операции V над семафором S к его значению просто прибавляется 1. В момент создания семафор может быть инициализирован любым неотрицательным значением.

Подобные переменные-семафоры могут с успехом применяться для решения различных задач организации взаимодействия процессов. В ряде языков программирования они были непосредственно введены в синтаксис языка (например, в ALGOL-68), в других случаях реализуются с помощью специальных системных вызовов. Соответствующая целая переменная располагается внутри адресного пространства ядра операционной системы. Операционная система обеспечивает атомарность операций P и V, используя, например, метод запрета прерываний на время выполнения соответствующих системных вызовов. Если при выполнении операции P заблокированными оказались несколько процессов, то порядок их разблокирования может быть произвольным, например, FIFO.

Барьеры

Барьер — это объект синхронизации, который используется для ограничения доступа к общему ресурсу или участку памяти. Он работает по принципу "разделяй и властвуй", позволяя эффективно управлять доступом к общему ресурсу в многопоточных приложениях.

В программировании барьеры используются для ограничения доступа к общим переменным, функциям, массивам и другим объектам, которые могут использоваться несколькими потоками одновременно. Они позволяют предотвратить конфликты при доступе к общему ресурсу, такие как доступ к общей памяти или разделяемому объекту.

Для использования барьера необходимо сначала определить его тип и параметры синхронизации, например, мьютекс или атомарную операцию. Затем можно создать экземпляр барьера с помощью оператора new и передать ему необходимые параметры синхронизации. После этого можно использовать барьер для ограничения доступа к объекту, вызывая методы объекта, такие как lock() или acquire() для блокировки доступа или release() для освобождения доступа.

Использование барьеров позволяет обеспечить безопасность данных, предотвратить утечки памяти и улучшить производительность приложений. Однако, использование барьеров может привести к проблемам с производительностью, если доступ к общему ресурсу осуществляется слишком часто или если ресурсы не освобождаются вовремя.

Мониторы

Хотя решение задачи producer-consumer с помощью семафоров выглядит достаточно изящно, программирование с их использованием требует повышенной осторожности и внимания, чем отчасти напоминает программирование на языке Ассемблера. Допустим, что в рассмотренном примере мы случайно поменяли местами операции P, сначала выполнив операцию для семафора mutex, а уже затем для семафоров full и empty. Допустим теперь, что потребитель, войдя в свой критический участок (mutex сброшен), обнаруживает, что буфер пуст. Он блокируется и начинает ждать появления сообщений. Но производитель не может войти в критический участок для передачи информации, так как тот заблокирован потребителем. Получаем тупиковую ситуацию.

В сложных программах произвести анализ правильности использования семафоров с карандашом в руках становится очень непросто. В то же время обычные способы отладки программ зачастую не дают результата, поскольку возникновение ошибок зависит от interleaving атомарных операций, и ошибки могут быть трудновоспроизводимы. Для того чтобы облегчить работу программистов, в 1974 году Хором (Hoare) был предложен механизм еще более высокого уровня, чем семафоры, получивший название мониторов. Мы с вами рассмотрим конструкцию, несколько отличающуюся от оригинальной.

Мониторы представляют собой тип данных, который может быть с успехом внедрен в объектно-ориентированные языки программирования. Монитор обладает собственными переменными, определяющими его состояние. Значения этих переменных извне могут быть изменены только с помощью вызова функций-методов, принадлежащих монитору. В свою очередь, эти функции-методы могут использовать в работе только данные, находящиеся внутри монитора, и свои параметры.

Сообщения

Для прямой и непрямой адресации достаточно двух примитивов, чтобы описать передачу сообщений по линии связи – send и receive. В случае прямой адресации мы будем обозначать их так:

send(P, message) – послать сообщение message процессу P ;
receive(Q, message) – получить сообщение message от процесса Q.

В случае непрямой адресации мы будем обозначать их так:

send(A, message) – послать сообщение message в почтовый ящик A ;
receive(A, message) – получить сообщение message из почтового ящика A.

Примитивы send и receive уже имеют скрытый от наших глаз механизм взаимоисключения. Более того, в большинстве систем они уже имеют и скрытый механизм блокировки при чтении из пустого буфера и при записи в полностью заполненный буфер. Реализация решения задачи producer-consumer для таких примитивов становится неприлично тривиальной. Надо отметить, что, несмотря на простоту использования, передача сообщений в пределах одного компьютера происходит существенно медленнее, чем работа с семафорами и мониторами.

Аппаратный ускоритель для нейронных сетей

Аппаратный ускоритель для нейронных сетей — это специализированное устройство, которое используется для ускорения работы нейронной сети путем повышения ее скорости обработки данных. Ускорители обычно состоят из нескольких компонентов, включая процессор, память и графический процессор (GPU), которые работают вместе для выполнения операций над данными.

Процессор отвечает за выполнение инструкций, которые требуют большого количества вычислительных ресурсов, таких как операции с плавающей запятой, целочисленное деление и умножение. Память отвечает за хранение данных, которые обрабатывает нейронная сеть. Графический процессор (GPU) отвечает за обработку изображений, видео и других типов визуализации данных.

Ускорители для нейронных сетей используются во многих областях, где требуется обработка больших объемов данных, таких как компьютерное зрение, распознавание речи, анализ текста и машинное обучение. Они позволяют ускорить работу нейронной сети, уменьшив время, необходимое для выполнения сложных операций над данными.

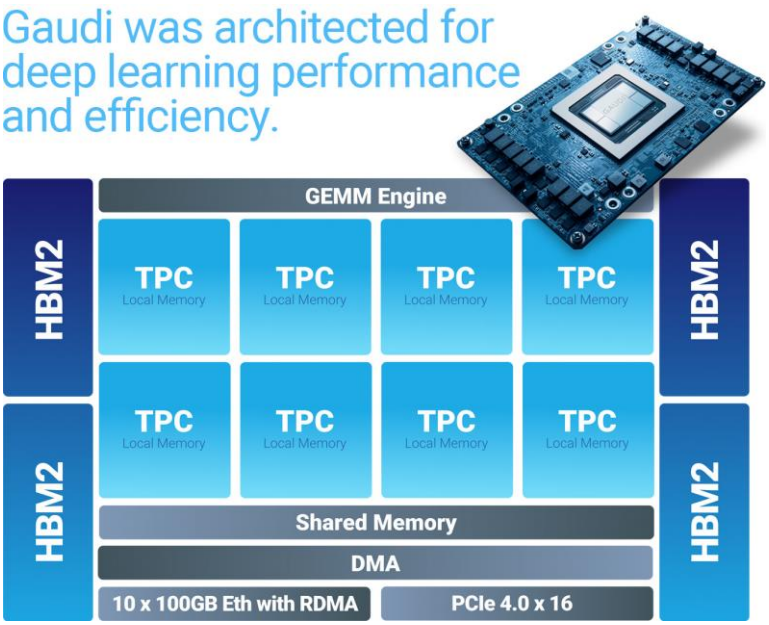

рис. 1. Архитектура ускорителя от Intel

Тензорный процессор TPC

Тензорный процессор — это специализированный процессор, который используется для обработки тензоров и векторных выражений. Тензорные процессоры являются одним из наиболее мощных инструментов для работы с тензорами и векторными выражениями в компьютерных системах.

Тензорные процессоры обладают высокой скоростью обработки и способны выполнять сложные математические операции над тензорами и векторными выражениями, такие как сложение, вычитание, умножение, возведение в степень и извлечение квадратного корня. Они также поддерживают работу с комплексными числами и матрицами, что делает их полезными инструментами для анализа и моделирования физических явлений, таких как электричество, магнетизм и теплопередача.

Тензорные процессоры широко применяются в научных исследованиях, инженерии, криптографии и других областях, где требуется обработка больших объемов данных и выполнение сложных вычислений. Они обеспечивают высокую скорость обработки и точность результатов, что делает их незаменимыми инструментами для инженеров, ученых и разработчиков программного обеспечения.

Векторное ядро VPU

Векторный процессор — это специализированная архитектура процессора, которая использует математические функции для представления и манипулирования данными в компьютере. Она была разработана в 1990-х годах и стала популярной в 2000-х годах благодаря своей способности эффективно работать с большими объемами данных и выполнять сложные вычисления.

Векторные процессоры были разработаны в ответ на растущую потребность в обработке больших объемов данных и выполнении сложных вычислений. Они предлагают возможность эффективно работать с данными, разбивая их на более мелкие части и выполняя операции над каждой частью по отдельности. Это позволяет значительно сократить время выполнения операций и повысить эффективность работы системы.

Технология векторных процессоров постоянно развивается и улучшается, и существуют различные реализации векторных процессоров, такие как MIPS и PowerPC, которые позволяют выполнять более сложные вычисления с использованием графических процессоров (GPU).

Матричный умножитель ME

Матричный умножитель — это устройство, которое используется для умножения двух матриц. Он состоит из нескольких строк и столбцов, каждая из которых содержит элементы, расположенные в определенном порядке. Когда матрица подается на умножитель, она обрабатывается последовательно, каждый элемент умножается на соответствующий элемент следующей строки, затем следующий элемент умножается на предыдущий элемент и так далее до тех пор, пока матрица не будет полностью обработана.

Матричные умножители широко используются в вычислительной технике, особенно в области машинного обучения и искусственного интеллекта. Они позволяют быстро и эффективно умножать большие объемы данных и выполнять сложные вычисления.

Существует несколько типов матричных умножителей, включая линейные, цилиндрические и сферические. Каждый тип имеет свои преимущества и недостатки, и выбор конкретного типа зависит от требований задачи и характеристик оборудования.

Задание

1. Спроектировать протокол синхронизации между устройствами TPC (семафоры, теговая система, барьеры).

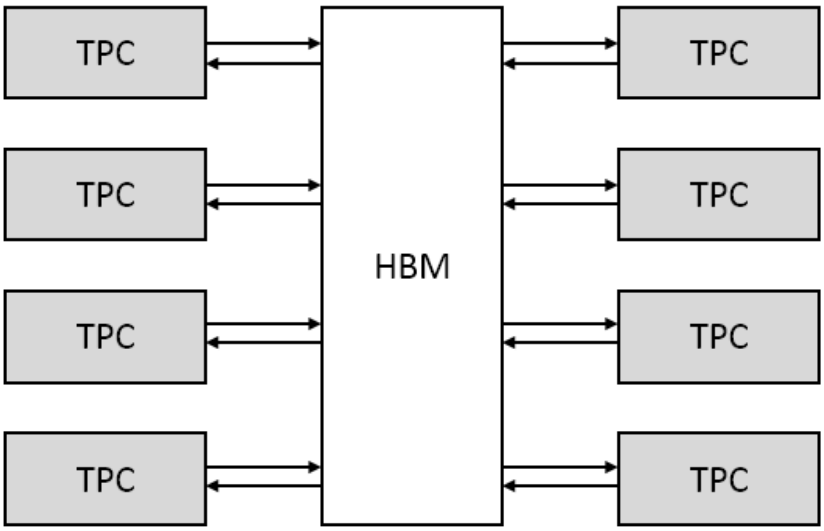

рис. 2. Функциональная схема устройства для задания №1

2. Спроектировать протокол синхронизации между 2 TPC на разных чиплетах через интерфейс D2D.

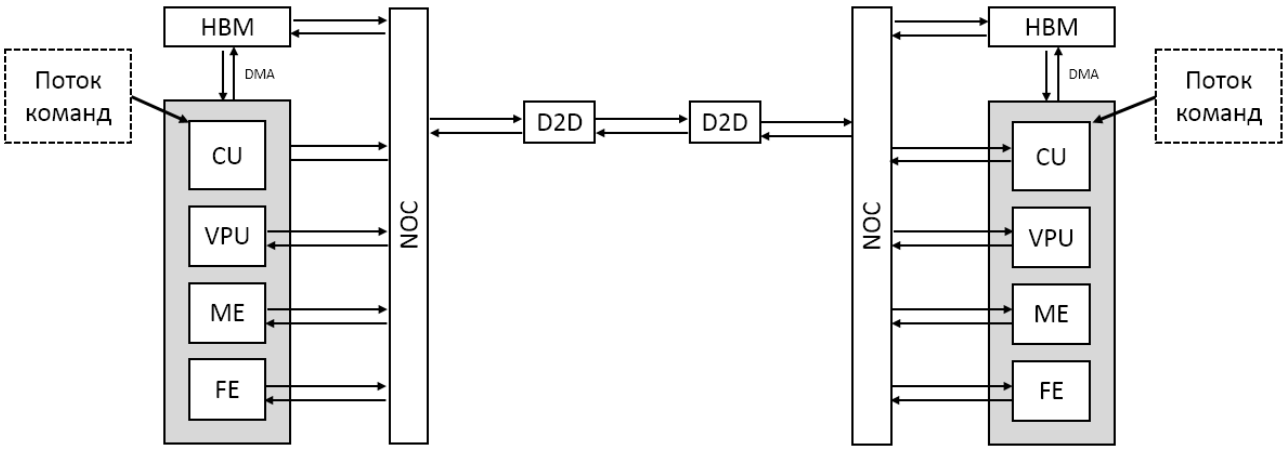

рис. 3. Функциональная схема устройства для задания №2

Программная модель

v(addr_start, addr_end) *# команда для векторного устройства VPU*
m(addr_start, addr_end) *# команда для матричного устройства ME*
f(addr_start, addr_end) *# команда для устройства расчета функций активации FE*
* addr_start – начальный адрес, addr_end – конечный адрес в байтах.

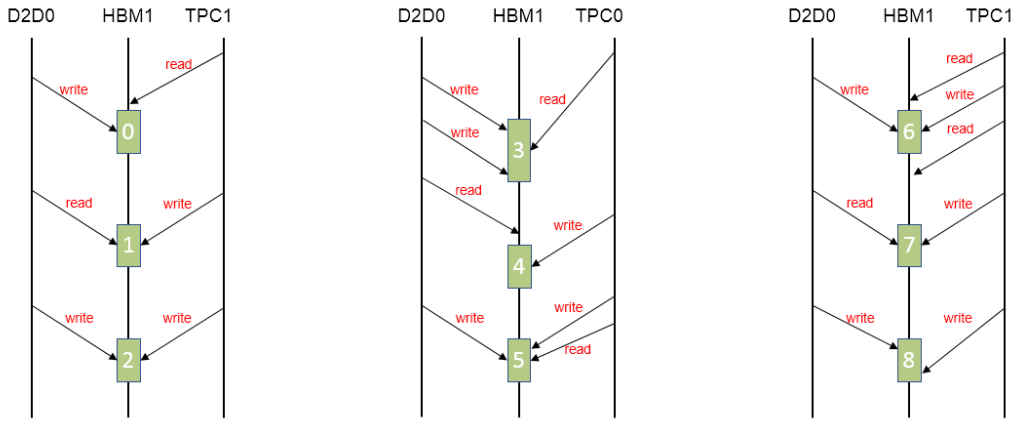

рис. 4. Возможные конфликты по данным

Возможен любой поток команд (с пересечениями по адресам) выше, например:

v(0, 10)
m(15, 50)
m(15, 50)
v(0, 50)
m(20, 70)
m(15, 50)
v(70, 80)
m(88, 90)
f(40, 50)

Входные данные

- Блок CU – планировщик (сервисное ядро для раздачи инструкций).
- Пропускная способность любого DMA порта 512 бит (любой запрос разбивается на n транзакций по 512 бит).
- Кол-во тактов работы VPU = 7.
- Кол-во тактов работы ME = 5.
- Кол-во тактов работы FE = 10.
- Вместимость всех устройств 64 байта (за одно вычисление, пока считаются предыдущие 64 байта следующие не запрашиваются).
- Последовательность обработки данных на любом устройстве: запросить 64 байта
- Доступ в свою память на TPC = 1.
- Доступ к HBM = [10, 30].
- Доступ к D2D = [100, 300].
- 4 устройства внутри TPC. Случайный поток команд.

Рекомендации к выполнению задания

В рамках решения задания возможно: добавление аппаратных блоков, например, глобального или локального планировщика, буфера; изменение программной модели, расширение системы команд.

Необходимо эмпирически подобрать задержку для добавленных устройств.

Защита работы

Отчет должен содержать:

1) функциональная схема устройства (при добавлении новых блоков или устройств) и алгоритм работы протокола синхронизации с точки зрения HW;

2) функциональная схема работы протокола и сценарий использования;
3) программная модель (при ее изменении или добавлении инструкций) и алгоритм работы протокола синхронизации с точки зрения SW.