ПРИЛОЖЕНИЕ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

КАФЕДРА «ЭЛЕКТРОННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ»

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ
«Параллельное программирование»

Направления подготовки:
02.03.03 «Математическое обеспечение и администрирование информационных систем»

Профиль:
Программное обеспечение компьютерных технологий и систем искусственного интеллекта

Квалификация выпускника – бакалавр

Форма обучения – очная

Рязань, 2025 г.

Общие положения

Оценочные материалы – это совокупность учебно-методических материалов (практических заданий, описаний форм и процедур проверки), предназначенных для оценки качества освоения обучающимися данной дисциплины как части ОПОП.

Цель – оценить соответствие знаний, умений и владений, приобретенных обучающимся в процессе изучения дисциплины, целям и требованиям ОПОП в ходе проведения промежуточной аттестации.

Основная задача – обеспечить оценку уровня сформированности общепрофессиональных и профессиональных компетенций.

Контроль знаний обучающихся проводится в форме промежуточной аттестации. Промежуточная аттестация проводится в форме зачета. Форма проведения зачета – тестирование, письменный опрос по теоретическим вопросам и выполнение практических заданий.

Описание показателей и критериев оценивания компетенций

Сформированность каждой компетенции (или ее части) в рамках освоения данной дисциплины оценивается по трехуровневой шкале:

- пороговый уровень является обязательным для всех обучающихся по завершении освоения дисциплины;
- продвинутый уровень характеризуется превышением минимальных характеристик сформированности компетенций по завершении освоения дисциплины;
- эталонный уровень характеризуется максимально возможной выраженностью компетенций и является важным качественным ориентиром для самосовершенствования.

Уровень освоения компетенций, формируемых дисциплиной:

Описание критериев и шкалы оценивания тестирования:

Шкала оценивания	Критерий
3 балла (эталонный уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 85 до 100%
2 балла (продвинутый уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 70 до 84%
1 балл (пороговый уровень)	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 50 до 69%
0 баллов	уровень усвоения материала, предусмотренного программой: процент верных ответов на тестовые вопросы от 0 до 49%

Описание критериев и шкалы оценивания теоретического вопроса:

Шкала оценивания	Критерий
3 балла (эталонный уровень)	выставляется студенту, который дал полный ответ на вопрос, показал глубокие систематизированные знания, смог привести примеры, ответил на дополнительные вопросы преподавателя
2 балла (продвинутый уровень)	выставляется студенту, который дал полный ответ на вопрос, но на некоторые дополнительные вопросы преподавателя ответил только с помощью наводящих вопросов
1 балл	выставляется студенту, который дал неполный ответ на вопрос

(пороговый уровень)	в билете и смог ответить на дополнительные вопросы только с помощью преподавателя
0 баллов	выставляется студенту, который не смог ответить на вопрос

Описание критериев и шкалы оценивания практического задания:

Шкала оценивания	Критерий
3 балла (эталонный уровень)	Задача решена верно
2 балла (продвинутый уровень)	Задача решена верно, но имеются неточности в логике решения
1 балл (пороговый уровень)	Задача решена верно, с дополнительными наводящими вопросами преподавателя
0 баллов	Задача не решена

На промежуточную аттестацию (зачет) выносится тест, два теоретических вопроса и 1 задача. Максимально студент может набрать 12 баллов. Итоговый суммарный балл студента, полученный при прохождении промежуточной аттестации, переводится в традиционную форму по системе «зачтено» и «не зачтено».

Оценки «зачтено» заслуживает обучающийся, продемонстрировавший полное знание материала изученной дисциплины, усвоивший основную литературу, рекомендованную рабочей программой дисциплины; показавший систематический характер знаний по дисциплине, ответивший на все вопросы билета или допустивший погрешности в ответах на вопросы, но обладающий необходимыми знаниями для их устранения под руководством преподавателя.

Дополнительным условием получения оценки «зачтено» могут стать успехи при выполнении лабораторных работ, систематическая активная работа на лабораторных работах.

Оценка «зачтено» выставляется студенту, набравшему 4 и более баллов при промежуточной аттестации.

Оценки «не зачтено» заслуживает обучающийся, продемонстрировавший серьезные пробелы в знаниях основного материала изученной дисциплины, не ответивший на все вопросы билета и дополнительные вопросы. Как правило, оценка «не зачтено» ставится обучающимся, которые не могут продолжить обучение по образовательной программе без дополнительных занятий по соответствующей дисциплине (формирования и развития компетенций, закрепленных за данной дисциплиной).

Оценка «не зачтено» выставляется студенту, набравшему менее 4 баллов при промежуточной аттестации.

Код компетенции	Результаты освоения ОПОП Содержание компетенций
ПК-1	Способен проектировать программное обеспечение с использованием современных инструментальных средств

ПК-1.1. Проектирует и разрабатывает программное обеспечение
ПК-1.2. Применяет современные инструментальные средства при разработке программного обеспечения

ПК-9	Способен применять языки программирования C/C++ для решения задач в области ИИ

ПК-9.1: Разрабатывает и отлаживает эффективные многопоточные решения на C++, тестирует, испытывает и оценивает качество таких решений

ПК-9.2: Разрабатывает и отлаживает системы ИИ на C++ под конкретные аппаратные платформы с ограничениями по вычислительной мощности, в том числе для встроенных систем

Паспорт оценочных материалов по дисциплине

Контролируемые разделы (темы) дисциплины	Код контролируемой компетенции (или её части)	Вид, метод, форма оценочного мероприятия
Тема 1. Общая характеристика параллельных вычислительных систем. Тенденции развития современных процессоров	ПК-1.1	Зачет
Тема 2. Параллельные вычисления. Метрики и закономерности параллельных вычислений	ПК-1.1	Зачет
Тема 3. Введение в SIMD-инструкции в C++	ПК-9.1	Зачет
Тема 4. Технология параллельного программирования OpenMP	ПК-9.1	Зачет
Тема 5. Технология параллельного программирования для графических процессоров CUDA	ПК-9.2	Зачет
Тема 6. Технология параллельного программирования для графических процессоров OpenCL	ПК-1.2	Зачет
Тема 7. Технология параллельного программирования MPI	ПК-1.2	Зачет

ТИПОВЫЕ КОНТРОЛЬНЫЕ ЗАДАНИЯ ИЛИ ИНЫЕ МАТЕРИАЛЫ

Промежуточная аттестация в форме зачета

Код компетенции	Результаты освоения ОПОП Содержание компетенций
ПК-1	Способен проектировать программное обеспечение с использованием современных инструментальных средств

ПК-1.1. Проектирует и разрабатывает программное обеспечение

Типовые тестовые вопросы:

1. _____ вычисления - это одновременное выполнение нескольких вычислительных процессов для решения одной задачи.
***Ответ**: Параллельные*

2. Классификация Флинна выделяет четыре архитектуры: SISD, _____, MISD и _____.
***Ответ**: SIMD, MIMD*

3. SISD означает: _____ Instruction, _____ Data.
***Ответ**: Single, Single*

4. SIMD означает: _____ Instruction, _____ Data.
***Ответ**: Single, Multiple*

5. Основные уровни параллелизма: микроуровень, _____, уровень потоков, уровень задач.
***Ответ**: уровень команд*

6. _____ – это мера отношения объема вычислений, выполненных в параллельной задаче, к объему коммуникаций
***Ответ**: Гранулярность*

6. Закон _____ утверждает, что максимальное ускорение ограничено долей последовательной части программы.
***Ответ**: Амдала*

7. Закон _____ предполагает, что объём задачи растёт вместе с числом процессоров.
***Ответ**: Густавсона*

8. _____ – это отношение времени, затрачиваемого на проведение вычислений на однопроцессорной ВС, ко времени решения той же задачи на параллельной n-процессорной системе.
***Ответ**: Ускорение*

9. Формула закона Амдала: S = 1 / (f + (1-f)/p), где f – доля _____ части программы.
***Ответ**: последовательной*

10. Архитектура, где несколько процессоров имеют доступ к общей памяти, называется архитектурой с _____.
***Ответ**: общей памятью.*

11. Что такое ускорение (Speedup) в параллельных вычислениях?
1. Отношение числа процессоров к времени выполнения
2. Разность времен последовательного и параллельного выполнения
+3. Отношение времени последовательного выполнения к параллельному
4. Произведение времени выполнения на число процессоров

12. Какой закон определяет предельное ускорение при фиксированном размере задачи?
1. Закон Мура
+2. Закон Амдала
3. Закон Ньютона
4. Закон Густавсона

13. Что такое эффективность параллельной программы?
+1. Отношение ускорения к числу процессоров
2. Отношение числа процессоров к ускорению
3. Разность ускорения и числа процессоров
4. Сумма ускорения и числа процессоров

14. Какой закон является альтернативой закону Амдала?
1. Закон Мура
+2. Закон Густавсона-Барсиса
3. Закон Парето
4. Закон Меткалфа

15. Что характеризует архитектуру с общей памятью?
1. Каждый процессор имеет свою локальную память
+2. Несколько процессоров имеют доступ к общей памяти
3. Память распределена между узлами
4. Отсутствие разделяемой памяти

16. Какая архитектура предполагает, что каждый процессор имеет свою локальную память?
1. Архитектура с общей памятью
2. Гибридная архитектура
3. Централизованная архитектура
+4. Архитектура с распределенной памятью

17. Какая метрика показывает, насколько хорошо используются дополнительные процессоры?
1. Балансировка нагрузки
2. Ускорение
+3. Эффективность
4. Масштабируемость

18. Что такое параллелизм на уровне задач?
1. Одинаковые операции над разными данными
+2. Разные операции над разными данными
3. Последовательное выполнение задач
4. Только параллельная обработка изображений

19. Какая из перечисленных проблем НЕ характерна для параллельных вычислений?
1. Накладные расходы коммуникации
2. Неравномерность нагрузки

3. Увеличение последовательных частей кода
+4. Отсутствие необходимости в синхронизации

20. Может ли программа быть полностью распараллеленной?
1. программа всегда не может быть распараллелена;
+2. программа может быть распараллелена, но не полностью;
3. программа всегда может быть распараллелена.
4. программа всегда может быть только последовательной.

Типовые теоретические вопросы:

1. Что такое параллельные вычисления?
***Ответ**: Это одновременное выполнение нескольких вычислительных процессов для решения одной задачи, позволяющее сократить общее время решения.*

2. Какие четыре класса архитектур выделяет таксономия Флинна? Кратко опишите каждый.
Ответ:
- SISD — один поток инструкций, один поток данных (классический последовательный CPU),
- SIMD — один поток инструкций, несколько потоков данных (GPU, векторные процессоры),
- MISD — несколько потоков инструкций, один поток данных,
- MIMD — несколько потоков инструкций и данных (многоядерные CPU, кластеры, MPI-системы).

3. Что такое параллелизм на уровне задач (Task Parallelism)?
***Ответ**: Разные операции выполняются параллельно над разными данными или разными частями задачи.*

4. Что такое ускорение (Speedup) в параллельных вычислениях?
***Ответ**: Отношение времени выполнения последовательной программы к времени выполнения параллельной версии: $S(n) = T(1) / T(n)$*

5. Что описывает закон Амдала?
***Ответ**: Предельное ускорение программы при фиксированном размере задачи:*
$S(n) = 1 / (f + (1-f)/n)$, где f - доля последовательной части.

6. Что такое эффективность (Efficiency) параллельной программы?
***Ответ**: Отношение ускорения к числу процессоров: $E(n) = S(n) /n$, показывает, насколько эффективно используются процессоры.*

7. Что такое накладные расходы (Overhead) в параллельных вычислениях?
***Ответ**: Дополнительные затраты времени на организацию параллельного выполнения (синхронизация, коммуникация, управление).*

8. Что такое масштабируемость параллельной программы?
***Ответ**: Способность программы увеличивать производительность при добавлении вычислительных ресурсов.*

9. Что описывает закон Густавсона-Барсиса?
***Ответ**: Альтернатива закону Амдала, учитывающая увеличение размера задачи с ростом числа процессоров: $S(n) = f + (1-f) \times n$*

10. Какие факторы ограничивают ускорение в параллельных программах?

***Ответ**: Последовательные части кода, накладные расходы коммуникации, неравномерность нагрузки, конфликты доступа к ресурсам.*

ПК-1.2. Применяет современные инструментальные средства при разработке программного обеспечения

Программный интерфейс для передачи информации MPI

1. Для работы с MPI в программе на C необходимо подключить заголовочный файл _____.
***Ответ**: mpi.h*

2. Инициализация MPI выполняется вызовом функции _____.
***Ответ**: MPI_Init*

3. Завершение работы с MPI выполняется вызовом функции _____.
***Ответ**: MPI_Finalize*

4. Стандартный коммуникатор, объединяющий все процессы, называется _____.
***Ответ**: MPI_COMM_WORLD*

5. Тип данных MPI для double — _____.
***Ответ**: MPI_DOUBLE*

6. Коллективная операция, при которой все процессы получают данные от одного (корневого) процесса, называется _____.
***Ответ**: MPI_Bcast*

7. Операция-редукция, соответствующая сложению, — это _____.
***Ответ**: MPI_SUM*

8. Процесс с рангом ____ называется главным (root-процессом) и часто используется для координации работы.
***Ответ**: 0*

9. Функция MPI_Send используется для ____ сообщения, а MPI_Recv — для его ____.
***Ответ**: отправки, приема*

10. Если программа работает в ____ режиме, это означает, что отправка сообщения завершается, даже если соответствующего приема еще не было.
***Ответ**: буферизованном*

11. Какой заголовочный файл необходимо подключить в C для работы с MPI?
1. <mpi.hpp>
2. <mpich.h>
+3. <mpi.h>
4. <message_passing.h>

12. Что возвращает функция MPI_Comm_rank(MPI_COMM_WORLD, &rank)?
1. Общее число процессов
+2. Уникальный номер текущего процесса
3. Идентификатор коммуникатора
4. Размер сообщения

13. Какая операция собирает данные от всех процессов и складывает их, результат получает только корневой процесс?
1. MPI_Allreduce
+2. MPI_Reduce
3. MPI_Gather
4. MPI_Scatter

14. Какие две функции являются коллективными (требуют участия всех процессов в коммуникаторе)?
1. MPI_Send
+2. MPI_Bcast
3. MPI_Recv
+4. MPI_Reduce

15. Какой из перечисленных — не является встроенной операцией редукции в MPI?
1. MPI_SUM
2. MPI_MAX
+3. MPI_AVG
4. MPI_PROD

16. Может ли MPI_Send блокироваться, если получатель ещё не вызвал MPI_Recv?
1. Нет, MPI_Send всегда неблокирующий
+2. Да, стандартный MPI_Send может блокироваться (режим стандартный)
3. Только при использовании MPI_ANY_SOURCE
4. Только в режиме отладки

17. Что возвращает функция MPI_Comm_size(MPI_COMM_WORLD, &size)?
1. Ранг текущего процесса
+2. Общее количество процессов в коммуникаторе
3. Размер сообщения в байтах
4. Размер типа данных MPI

18. Что делает функция MPI_Barrier(MPI_COMM_WORLD)?
1. Завершает работу всех процессов
+2. Блокирует выполнение, пока все процессы в коммуникаторе не достигнут этого вызова
3. Создает барьер между процессами разных рангов
4. Синхронизирует только процессы с четными рангами

19. Что делает функция MPI_Gather?
1. Рассылает данные от одного процесса всем
2. Суммирует данные со всех процессов
3. Синхронизирует выполнение процессов
+4. Собирает данные от всех процессов на одном процессе

20. Какая функция используется для неблокирующей отправки сообщения?
1. MPI_Send
2. MPI_Ready
+3. MPI_Isend
4. MPI_Async

OpenCL - фреймворк для параллельного программирования

1. _____ — открытый кроссплатформенный стандарт для гетерогенных вычислений, управляемый Khronos Group.

***Ответ**: OpenCL*

2. Для запуска приложений на GPU от AMD или Intel (без NVIDIA) предпочтительнее использовать _____.
***Ответ**: OpenCL*

3. Уровень переносимости: _____ работает на CPU, GPU (NVIDIA/AMD/Intel), FPGA; _____ — только на GPU NVIDIA.
***Ответ**: OpenCL, CUDA*

4. Модель выполнения в OpenCL: рабочие элементы (work-items) → _____ → диапазон (NDRange).
***Ответ**: work-group*

5. В OpenCL функция, выполняемая на устройстве, называется _____.
***Ответ**: kernel, ядро*

6. В модели выполнения OpenCL наименьшая единица работы называется _____.
***Ответ**: work-item (рабочий элемент)*

7. Группа рабочих элементов в OpenCL называется _____.
***Ответ**: work-group (рабочая группа)*

8. N-мерное пространство элементов в OpenCL называется _____.
***Ответ**: NDRange*

9. Программа, которая выполняется на CPU и управляет выполнением на GPU, называется _____ программой.
Ответ: хост-программа (host program)

10. Основное преимущество OpenCL перед CUDA - это его _____.
***Ответ**: кроссплатформенность*

11. Какие типы памяти явно определены в модели памяти OpenCL?
1. Только глобальная и локальная
2. Только приватная и постоянная
+3. Глобальная, локальная, приватная и постоянная
4. Только кэш-память

12. Что такое kernel-функция в OpenCL?
1. Функция операционной системы
2. Функция, выполняемая на host-процессоре
+3. Функция, компилируемая и выполняемая на устройстве OpenCL
4. Функция для работы с файлами

13. Какой из этих этапов НЕ является частью типичного workflow OpenCL?
1. Выбор платформы и устройства
2. Создание контекста и очередей команд
3. Компиляция kernel во время выполнения
+4. Установка операционной системы

14. Какая основная цель использования локальной памяти в OpenCL?
1. Увеличить объем доступной памяти

+2. Уменьшить задержки при доступе к часто используемым данным
3. Хранить результаты выполнения kernel
4. Обмениваться данными с host-программой

15. Что такое контекст (context) в OpenCL?
1. Очередь команд для устройства
+2. Набор устройств, совместно использующих ресурсы памяти
3. Пространство выполнения kernel
4. Регионы памяти device

16. Какой механизм используется для управления выполнением kernel на устройстве?
1. Потоки операционной системы
+2. Очереди команд (command queues)
3. Непосредственный вызов функций
4. Сигналы прерывания

17. Какое утверждение о приватной памяти верно?
1. Доступна всем work-items в NDRange
2. Разделяется между work-items в work-group
+3. Приватна для каждого work-item
4. Доступна с host-программы

18. Как получить глобальный ID рабочего элемента в ядре?
1. threadIdx.x
+2. get_global_id(0)
3. blockIdx.x * blockDim.x + threadIdx.x
4. work_item_id()

19. Какие две из перечисленных технологий являются открытыми стандартами (не проприетарными)?
1. CUDA
+2. OpenCL
3. ROCm
+4. SYCL

20. Почему OpenCL особенно полезен для embedded- и edge-устройств?
1. Потому что требует мощного GPU
2. Потому что работает только на NVIDIA
+3. Потому что поддерживается на CPU, GPU, FPGA (в т.ч. ARM Mali, Intel iGPU, Raspberry Pi)
4. Потому что проще CUDA

Типовые теоретические вопросы:

1. В чём принципиальное отличие модели MPI от модели общей памяти (например, OpenMP)?
Вариант ответа*: В MPI процессы не имеют общей оперативной памяти — каждый работает в изолированном адресном пространстве и взаимодействует только через отправку/приём сообщений. В OpenMP потоки разделяют память одного процесса и синхронизируются.*

2. Что такое коммуникатор в MPI?
Вариант ответа*: Коммуникатор - группа процессов, которые могут взаимодействовать друг с другом.*

3. Что такое ранг (rank) процесса и почему он начинается с 0?

***Вариант ответа**: Ранг — уникальный целочисленный идентификатор процесса в коммуникаторе (обычно от 0 до size–1). Нумерация с нуля принята по аналогии с массивами в Си и упрощает расчёты (например, деление работы: процесс i обрабатывает блок i).*

4. Что возвращает функция MPI_Comm_size(MPI_COMM_WORLD, &size) и когда её значение может отличаться от числа запущенных процессов?

***Вариант ответа**: Она возвращает общее число процессов в коммуникаторе MPI_COMM_WORLD. Значение всегда равно числу процессов, заданному при запуске и не меняется в ходе выполнения.*

5. Почему MPI_Finalize() должен вызываться всеми процессами — и что произойдёт, если один процесс его пропустит?

***Вариант ответа**: Потому что MPI_Finalize() — коллективная операция: все процессы обязаны в ней участвовать. Если один процесс не вызовет её — программа зависнет, так как остальные будут ждать его завершения.*

6. Чем отличается MPI_Send от MPI_Recv по блокировке? Может ли MPI_Send «зависнуть»?

***Вариант ответа**: Обе функции — блокирующие: MPI_Send возвращает управление только после того, как сообщение можно безопасно повторно использовать; MPI_Recv — после полного приёма. MPI_Send может «зависнуть», если получатель не вызвал MPI_Recv.*

7. Что такое коллективная операция в MPI и чем она отличается от точечной (send/recv)?

***Вариант ответа**: Коллективная операция — это вызов, в котором участвуют все процессы коммуникатора (например, MPI_Bcast, MPI_Reduce). В отличие от точечных операций (от одного к одному), коллективные обычно более эффективны за счёт оптимизированных алгоритмов.*

8. В чем состоит основная идея архитектуры Host-Device в OpenCL?

***Вариант ответа**:* Host (хост) управляет выполнением на CPU, Device (устройство) выполняет параллельные вычисления. Host контролирует память, очереди команд и выполнение kernel-функций.

9. Что такое kernel-функция в OpenCL?

***Вариант ответа**: Kernel - функция, выполняемая на устройстве OpenCL. Компилируется во время выполнения и выполняется множеством work-items параллельно.*

10. Объясните иерархию выполнения в OpenCL.

***Вариант ответа**: Work-item (отдельный элемент) → Work-group (группа элементов) → NDRange (все элементы). Work-items в группе могут синхронизироваться и использовать локальную память.*

11. Какие основные компоненты платформы OpenCL?

***Вариант ответа**: Платформа, устройства, контекст, очереди команд, программа, kernel, буферы памяти.*

12. Что такое NDRange и какова его роль?

***Вариант ответа**: NDRange - N-мерное индексное пространство, определяющее организацию work-items. Задает глобальный и локальный размер для параллельного выполнения.**

13. Какие типы памяти существуют в модели памяти OpenCL?

***Вариант ответа**:: Глобальная, локальная, приватная, постоянная. Различаются по области видимости, времени жизни и производительности.*

14. Для чего используется локальная память в OpenCL?
Вариант ответа: *Для разделяемой памяти внутри work-group. Обеспечивает быстрый доступ к данным, совместно используемым work-items группы.*

15. Какие основные этапы работы с OpenCL программой?
Вариант ответа: *Выбор платформы/устройства, создание контекста и очереди, компиляция программы, создание kernel, выделение памяти, выполнение, чтение результатов.*

ПК-9	Способен применять языки программирования C/C++ для решения задач в области ИИ

ПК-9.1: Разрабатывает и отлаживает эффективные многопоточные решения на C++, тестирует, испытывает и оценивает качество таких решений

Типовые тестовые вопросы:

1. Первое SIMD-расширение для x86, представленное Intel в 1997 г., называлось _____.
Ответ: *MMX*

2. Автоматическая векторизация циклов компилятором называется _____.
Ответ*: автовекторизация*

3. Для подсказки компилятору векторизовать цикл в OpenMP используется директива _____.
Ответ: *#pragma omp simd*

4. При работе с SIMD важно, чтобы данные в памяти были _____ по границе 16, 32 или 64 байт.
Ответ*: выровнены*

5. Теоретический прирост производительности при суммировании массива float с использованием AVX (256 бит) вместо скалярного кода составляет до _____ раз.
Ответ: *8*
(256 / 32 = 8 значений float за такт)

6. Интринсики — это _____ функции, которые компилятор преобразует в одну или несколько SIMD-инструкций.
Ответ*: встроенные*

7. Высокий уровень _____ (например, if внутри цикла) затрудняет или делает невозможной векторизацию.
Ответ*: ветвления (или условных переходов)*

8. SIMD-инструкции исполняются не на отдельных ядрах, а на _____ блоках внутри одного ядра процессора.
Ответ*: векторных (или SIMD-, ALU-векторных)*

9. Для использования SIMD-инструкций в коде программист часто может использовать встроенные функции (intrinsics) или писать код на _____.
Ответ*: ассемблере*

10. Чтобы код использовал SIMD-инструкции, критически важно обеспечивать _____ данных в памяти для эффективной загрузки в широкие регистры.
***Ответ**: выравнивание*

11. Основная единица работы в SIMD – это _____, а не отдельный скаляр.
***Ответ**: вектор.*

12. Для работы с числами одинарной точности (float) в 128-битном регистре можно разместить _____ элементов.
***Ответ**: четыре (128 / 32 = 4)*

13. Если количество элементов в массиве не кратно размеру SIMD-пакета, остающиеся элементы обычно обрабатываются в _____ цикле.
***Ответ**: скалярном.*

14. OpenMP — это стандарт для _____ параллельного программирования на языках C, C++ и Fortran.
***Ответ**: многопоточного.*

15. Для распараллеливания циклов в OpenMP часто используется директива _____.
***Ответ**: parallel for.*

16. В OpenMP переменные могут быть объявлены как shared или _____.
***Ответ**: private.*

17. Число потоков в OpenMP можно задать с помощью функции _____.
***Ответ**: omp_set_num_threads.*

18. Для предотвращения состояния гонки в OpenMP используется директива __ _____.
***Ответ**: critical.*

19. OpenMP поддерживает различные схемы распределения итераций, такие как static и _____.
***Ответ**: dynamic.*

20. Использование OpenMP позволяет значительно _____ время выполнения вычислительных задач.
***Ответ**: сократить.*

21. Что такое SIMD-инструкции в контексте параллельных вычислений?
1. Инструкции, предназначенные только для последовательной обработки данных
+ 2. Инструкции, позволяющие выполнять одновременную обработку нескольких данных одним набором команд
3. Инструкции, которые заменяют все остальные виды инструкций в процессоре
4. Инструкции, используемые только в графических процессорах

22. Какое преимущество дает использование SIMD-инструкций при обработке мультимедийных данных?
1. Снижение энергопотребления за счет уменьшения количества инструкций
2. Обеспечение совместимости с устаревшим программным обеспечением
3. Уменьшение размера программного кода
+4. Увеличение скорости обработки за счет параллельной обработки элементов данных

23. Какой из перечисленных процессоров наиболее эффективно использует SIMD-инструкции?
1. Процессор без поддержки SIMD-инструкций
+2. Процессор с расширенной поддержкой инструкций SSE или AVX
3. Процессор, оптимизированный только для однопоточной работы
5. Процессор с низкой тактовым частотой

24. Что необходимо для эффективного использования SIMD-инструкций в программном обеспечении?
+1. Оптимизация данных и алгоритмов для параллельной обработки
2. Обеспечение совместимости только с одноплатными системами
3. Минимизация использования регистров процессора
3. Использование только однопоточных алгоритмов

25. Какая из следующих задач лучше всего подходит для ускорения с помощью SIMD-инструкций?
1. Обработка последовательных команд в управлении устройствами
+2. Обработка больших массивов чисел для вычислений в научных задачах
3. Обработка пользовательского интерфейса
4. Обработка событий в операционной системе

26. Какие ограничения существуют при использовании SIMD-инструкций?
+1. Ограничение на тип данных и их выравнивание в памяти
2. Ограничение только на однопоточную обработку данных
3. Ограничение только на работу с графическими данными
4. Отсутствие поддержки в большинстве современных процессоров

27. Что такое векторизация в контексте SIMD?
1. Процесс разделения данных на отдельные потоки без использования инструкций SIMD
+2. Преобразование последовательных операций в параллельные с помощью векторных инструкций
3. Обеспечение совместимости кода с разными архитектурами процессоров
4. Процесс преобразования кода в последовательный режим выполнения

28. Какая роль у регистров SIMD в процессоре?
+1. Хранение векторных данных для одновременной обработки
2. Хранение управляющих команд для последовательной обработки
3. Обеспечение доступа к оперативной памяти
4. Хранение инструкций для выполнения в режиме ядра

29. Какое влияние оказывает использование SIMD-инструкций на энергопотребление системы?
1. Не влияет на энергопотребление
2. Всегда приводит к увеличению тепловыделения
3. Обязательно увеличивает энергопотребление из-за дополнительных операций
+4. Может снизить энергопотребление за счет уменьшения общего времени выполнения задач

30. Какой из циклов наиболее подходит для SIMD-векторизации?
```
1. for (i=1; i<N; i++) a[i] = a[i-1] + 1;
2. for (i=0; i<N; i++) if (a[i]>0) b[i]=1; else b[i]=0;
+3. for (i=0; i<N; i++) c[i] = a[i] + b[i];
4. for (i=0; i<N; i++) printf("%f", a[i]);
```

31. Как совместить OpenMP и SIMD в одном цикле?

```
1. #pragma omp simd parallel for
+2. #pragma omp parallel for simd
3. #pragma omp simd; #pragma omp parallel for
4.  #pragma omp vector for
```

32. Какой эффект даёт комбинация OpenMP + SIMD по сравнению со скалярным однопоточным кодом?

1. Ускорение ≈ (число ядер)
2. Ускорение ≈ (ширина SIMD-регистра)
+3. Ускорение ≈ (число ядер) × (ширина SIMD-регистра)
4. Ускорение ≈ max(ядра, SIMD)

33. Какая клауза OpenMP нужна для безопасного суммирования sum += a[i] в параллельном цикле?

1. private(sum)
2. shared(sum)
+3. reduction(+:sum)
4. atomic(sum)

34. Какая директива OpenMP отключает неявный барьер после цикла?

1. noblock
2. async
+3. nowait
4. fire_and_forget

35. Какие две технологии позволяют добиться параллелизма на современном CPU?

+1. OpenMP
2. CUDA
+3. SIMD
4. MPI

36. Почему #pragma omp critical может свести на нет прирост от параллелизации?

+1. Из-за накладных расходов на синхронизацию
2. Потому что он отключает SIMD
3. Потому что он создаёт новые потоки
4. Из-за кэш-промахов

Типовые теоретические вопросы:

1. Что означает аббревиатура SIMD и как этот принцип повышает производительность?

Вариант ответа: *SIMD — Single Instruction, Multiple Data. Одна инструкция процессора применяется параллельно к нескольким данным (например, 4 float в одном 128-битном регистре), что увеличивает вычислительную плотность и эффективно использует векторные блоки ALU.*

2. Перечислите три SIMD-расширения для архитектуры x86 и укажите ширину их регистров.

Вариант ответа: *MMX (64 бит), SSE (128 бит), AVX (256 бит). (Допустимо: AVX-512 — 512 бит)*

3. Какие два основных способа использовать SIMD в C/C++ на уровне программиста?

Вариант ответа:

1) Ручное написание кода с использованием интринсиков (например, _mm_add_ps).

2) Полагаться на автовекторизацию компилятора

4. Почему циклы с зависимостями по данным (например, a[i] = a[i-1] + b[i]) не поддаются векторизации?

Вариант ответа: *Потому что результат итерации i зависит от результата итерации i-1. SIMD требует независимости итераций — иначе нарушается корректность вычислений.*

5. Что такое автовекторизация и какие факторы способствуют её успешному применению компилятором?

Вариант ответа: *Автовекторизация — автоматическое преобразование компилятором скалярного цикла в векторный SIMD-код. Успех зависит от: отсутствия зависимостей, простых арифметических операций, выровненных данных, отсутствия вызовов функций с побочными эффектами.*

6. Как директива #pragma omp simd отличается от #pragma omp parallel for?

Вариант ответа: *simd указывает компилятору векторизовать тело цикла (работает внутри одного потока), тогда как parallel for распределяет итерации цикла между разными потоками. Их можно комбинировать: #pragma omp parallel for simd.*

7. Объясните назначение клаузы reduction(+:sum) в OpenMP. Как она работает?

Вариант ответа: *Обеспечивает корректное суммирование общей переменной sum из нескольких потоков. Каждый поток создаёт локальную копию sum, инициализированную нулём; после завершения параллельной области локальные значения складываются в глобальную переменную.*

8. Приведите пример цикла, который хорошо векторизуется, и объясните, почему.

Вариант ответа:

*for (int i = 0; i < N; ++i) c[i] = a[i] + b[i] * 0.5f;*

Итерации независимы, операции простые (умножение и сложение), доступ к памяти последователен → идеально для SIMD.

9. Какие клаузы в OpenMP можно использовать для управления синхронизацией в циклах без явного barrier?

Вариант ответа: nowait — отключает неявный барьер в конце for, sections, single. Позволяет потокам продолжить выполнение сразу после завершения своей части.

10. Чем #pragma omp atomic отличается от #pragma omp critical? В каких случаях предпочтительнее atomic?

Вариант ответа: *atomic применяется только к одной простой операции (например, x++) и реализуется одной инструкцией. critical — к произвольному блоку кода, дороже.*

ПК-9.2: Разрабатывает и отлаживает системы ИИ на C++ под конкретные аппаратные платформы с ограничениями по вычислительной мощности, в том числе для встроенных систем

Типовые тестовые вопросы:

1. Основное преимущество GPU перед CPU – это высокая _____ параллелизма (миллионы потоков).

Ответ: *степень (или уровень)*

2. Группа из _____ потоков, выполняющихся совместно на одном SM, называется warp.
***Ответ**: 32*

3. Память, общая для всех потоков в одном блоке и имеющая низкую задержку, — это _____ память.
***Ответ**: shared (разделяемая)*

4. Запуск ядра `vecAdd` с 64 блоками и 256 потоками в блоке записывается как:
vecAdd _____ (d_a, d_b, d_c, N);
***Ответ**: <<<64, 256>>>*

5. Глобальный индекс потока в одномерной сетке вычисляется как:
int idx = _____ * _____ + _____;
***Ответ**: blockIdx.x, blockDim.x, threadIdx.x*

6. Для выделения памяти на GPU используется функция _____.
***Ответ**: cudaMalloc*

7. Копирование данных с CPU на GPU выполняется функцией _____ .
***Ответ**: cudaMemcpy*

8. Чтобы дождаться завершения всех GPU-операций, вызывают _____();.
***Ответ**: cudaDeviceSynchronize*

9. Для освобождения памяти на GPU используется функция _____.
***Ответ**: cudaFree*

10. В архитектуре CUDA вычислительные устройства GPU называются _____.
***Ответ:** устройства (devices)*

11: Функция, выполняемая на GPU в CUDA, называется _____.
***Ответ:** ядром (kernel)*

12. В модели программирования CUDA, CPU называется _____ и управляет выполнением на GPU.
***Ответ:** хостом (host)*

13. Основная единица выполнения в CUDA называется _____, которая группируется в блоки.
***Ответ:** потоком (thread)*

14. Блоки потоков в CUDA организуются в _____, которые выполняются на одном Streaming Multiprocessor.
***Ответ:** сетку (grid)*

15. Память GPU, доступная всем блокам сетки, называется _____ памятью.
***Ответ:** глобальной (global)*

16. _____ память является самой быстрой, но и самой маленькой по объему в иерархии памяти GPU.
***Ответ:** Регистровая*

17. Переменная _____ в CUDA указывает уникальный идентификатор потока в блоке.

Ответ: *threadIdx*

18. Для синхронизации потоков внутри одного блока используется функция _____.

Ответ: *__syncthreads()*

19. Что такое гетерогенная вычислительная система?

1. Система, использующая процессоры только одного типа

+2. Система, использующая различные типы вычислительных блоков (GPP, GPU, DSP, FPGA, ASIC)

3. Система, предназначенная исключительно для обработки графики

4. Система, не имеющая операционной системы

20. Основное преимущество гетерогенных систем заключается в том, что...

1. все типы процессоров универсальны и выполняют любые задачи одинаково хорошо.

+2. они позволяют использовать каждый тип вычислительных элементов для наиболее подходящих ему задач.

3. они полностью заменяют центральные процессоры (CPU).

4. они не требуют параллельного программирования.

21. Какое из утверждений о CPU и GPU в гетерогенной системе ВЕРНО?

1. GPU лучше подходит для последовательных задач, а CPU — для массово-параллельных.

+2. CPU лучше подходит для управления программой и последовательной логики, а GPU — для массово-параллельных вычислений.

3. CPU и GPU идентичны по архитектуре и выполняют задачи одинаково.

4. GPU полностью заменяет CPU в гетерогенных системах.

22. Чем архитектура GPU принципиально отличается от архитектуры CPU?

1. GPU имеет одно мощное ядро, оптимизированное для минимальной задержки.

+2. GPU содержит множество упрощенных ядер, ориентированных на максимальную пропускную способность.

3. GPU не имеет собственной памяти.

4. В GPU отсутствуют арифметико-логические устройства.

23. Какое из утверждений лучше всего описывает принцип работы GPU?

1. Выполнение сложной логики с множеством условных переходов.

2. Последовательная обработка данных.

+3. Массовый параллелизм: одновременное выполнение одной и той же операции на множестве данных (SIMD).

4. Управление работой всех компонентов компьютера.

24. Почему GPU плохо справляется с задачами, содержащими много ветвлений?

1. Потому что у него нет кэш-памяти.

+2. Потому что ядра GPU объединены в группы (warp) и выполняют одни и те же инструкции.

3. Потому что он не поддерживает операции с плавающей точкой.

4. Потому что его тактовая частота слишком низка.

25. Для чего используется директива __syncthreads() в CUDA?

1. Для синхронизации всех потоков в сетке

+2. Для синхронизации потоков в пределах одного блока

3. для копирования данных с хоста на устройство

4. Для запуска ядра на GPU

26. Функция, которая выполняется на GPU в модели программирования CUDA, называется...
1. Драйвер (Driver)
2. Поток (Thread)
+3. Ядро (Kernel)
4. Хост (Host)

27. Как организована иерархия потоков в CUDA?
1. Потоки -> Сетки (Grids) -> Блоки (Blocks)
2. Блоки (Blocks) -> Потоки -> Сетки (Grids)
+3. Потоки объединяются в блоки, которые образуют сетку (Grid)
4. Иерархии потоков не существует

28. Какая переменная в коде CUDA-ядра содержит индекс потока в пределах своего блока?
1. blockIdx.x
2. blockDim.x
+3. threadIdx.x
4. gridDim.x

29. Почему передача данных между хостом (CPU) и устройством (GPU) является узким местом?
+1. Память хоста и устройства физически разделена, а передача по шине (например, PCIe) занимает время
2. GPU не может работать с данными в оперативной памяти CPU
3. Данные должны быть зашифрованы перед передачей
4. CPU и GPU работают на разных частотах

30. Что из перечисленного НЕ является типичным оптимизируемым типом памяти в CUDA?
1. Разделяемая память (Shared Memory)
2. Глобальная память (Global Memory)
3. Память констант (Constant Memory)
+4. Оперативная память хоста (Host RAM)

Типовые теоретические вопросы

1. Что такое гетерогенные вычисления и почему они эффективны для современных высокопроизводительных задач?

Вариант ответа: *Гетерогенные вычисления — использование в одной системе разных типов вычислителей (CPU, GPU, FPGA и др.), каждый из которых оптимизирован под свой класс задач. CPU эффективен для последовательных и сильно разветвленных задач, GPU — для массово-параллельных вычислений с высокой арифметической интенсивностью. Это позволяет достичь баланса между латентностью, пропускной способностью и энергоэффективностью.*

2. Опишите иерархию памяти в GPU NVIDIA (от самой быстрой к самой медленной).

Вариант ответа:

1) Регистры — приватные для каждого потока, самые быстрые.

2) Shared-память — общая для потоков в одном блоке, низкая задержка, программируемый кэш.

3) L1-кэш / L2-кэш — общие для SM / всего GPU.

4) Глобальная память (DRAM) — основная память GPU, высокая пропускная способность, но высокая задержка.

3. Почему shared-память критически важна для производительности CUDA-программ?

***Вариант ответа**: Потому что shared-память имеет задержку на порядки ниже, чем глобальная память, и позволяет избежать «узкого места» при частом доступе к одним и тем же данным (например, в свёртках, матричном умножении). Использование shared-памяти как программируемого кэша — ключевой приём оптимизации.*

4. Что означают спецификаторы __global__, __device__ и __host__ в CUDA? Приведите пример их комбинации.

Вариант ответа:

- __global__ — функция ядра: вызывается с CPU, выполняется на GPU.

- __device__ — функция: вызывается и выполняется только на GPU.

- __host__ — функция: только на CPU (по умолчанию).

Комбинация: __host__ __device__ void f() — компилируется и для CPU, и для GPU (например, для утилитарных функций).

5. Как вычисляется глобальный индекс потока в одномерной сетке? Почему именно так?

***Вариант ответа:** int idx = blockIdx.x * blockDim.x + threadIdx.x;*

*Потому что блоки (grid) содержат фиксированное число потоков (blockDim.x), и blockIdx.x задаёт смещение блока, а threadIdx.x — позицию внутри блока. Это обеспечивает непрерывную нумерацию потоков от 0 до gridDim.x * blockDim.x – 1.*

6. Зачем нужно явно управлять памятью (cudaMalloc, cudaMemcpy) в CUDA, в отличие от обычных программ на CPU?

***Вариант ответа**: Потому что GPU имеет отдельную физическую память (VRAM), не совместимую по адресному пространству с RAM CPU. Данные не могут быть доступны напрямую — их нужно явно выделять на GPU и копировать между хостом и устройством. Это даёт контроль над задержками и пропускной способностью.*

7. Что делает `cudaDeviceSynchronize()` и когда его обязательно использовать?

***Вариант ответа**: Эта функция блокирует CPU-поток до завершения всех ранее запущенных ядер и операций копирования на GPU. Её обязательно использовать перед освобождением памяти (cudaFree), перед копированием результата с GPU на CPU, и при замере времени (wall-clock), чтобы избежать гонок.*

8. Почему запуск ядра (<<<...>>>) является *асинхронной* операцией? Как это влияет на производительность?

***Вариант ответа**: Потому что CPU продолжает выполнение сразу после запуска, не дожидаясь завершения ядра. Это позволяет перекрывать вычисления на GPU с работой CPU.*

9. Что такое Compute Capability и почему важно его учитывать при разработке CUDA-приложений?

***Вариант ответа**: Compute Capability — версия архитектуры GPU, определяющая поддерживаемые инструкции (Tensor Cores, динамический параллелизм), лимиты (размер блока, shared-память) и функции.*

10. Почему для малых задач (например, N = 1000) ускорение на GPU может быть отрицательным?

***Вариант ответа**: Потому что накладные расходы на запуск ядра, копирование данных между CPU и GPU и синхронизацию превышают выигрыш от параллельных вычислений. GPU эффективен при достаточно большом объёме работы, чтобы «загрузить» тысячи ядер.*

11. Как профилировать CUDA-приложение для поиска узких мест?

***Вариант ответа**: Используют Nsight Compute (для ядер) и Nsight Systems (для всей системы).*

12. Почему CUDA остаётся доминирующей платформой для ИИ, несмотря на наличие открытых альтернатив (OpenCL, SYCL)?

***Вариант ответа**: Потому что:*

- Глубокая оптимизация под архитектуру NVIDIA (Tensor Cores, cuDNN, cuBLAS)

- Зрелая экосистема (PyTorch/TensorFlow по умолчанию используют CUDA)

- Инструменты (Nsight, profilers), документация, сообщество

- Высокая производительность и стабильность — критично для production.

Типовые теоретические вопросы для зачета по дисциплине

1. Тенденции развития современных процессоров: многопоточность, многоядерность, ИИ-ускорители.
2. Классификация архитектур ВС Флинна. Классификация Ванга и Бриггса.
3. Уровни параллелизма. Степень гранулярности. Метрики параллелизма.
4. Закономерности параллельных вычислений. Закон Амдала. Закон Густафсона. Закон Сана-Ная.
5. Основные характеристики вычислительных систем с общей памятью.
6. Технология параллельного программирования OpenMP. Основные понятия и определения. Модель вычислений и классы переменных.
7. OpenMP. Параллельные и последовательные области. Директива parallel.
8. OpenMP. Конструкции распределения работы.
9. Синхронизация в OpenMP.
10. Гетерогенные ВС. Архитектура GPU.
11. Обзор средств программирования для GPU.
12. CUDA. Понятие потока, блока, сети блоков. Функция-ядро. Спецификаторы функций и переменных.
13. CUDA. Иерархия памяти на GPU.
14. CUDA. Глобальная память. Шаблон работы с глобальной памятью. Использование pinned-памяти.
15. CUDA. CUDA-потоки.
16. CUDA. Разделяемая память. Шаблон работы с разделяемой памятью. Оптимизация работы с разделяемой памятью. Банк-конфликты.
17. OpenCL. Основные понятия и определения. Обобщенные модели.
18. OpenCL. Модель платформы и модель исполнения.
19. OpenCL. Модель памяти, объекты памяти и модель программирования.
20. OpenCL. Среда программирования
21. Ядро OpenCL. Типы данных OpenCL. Квалификаторы адресного пространства и доступа.

22. Технология параллельного программирования MPI. Основные понятия и определения. Структура MPI-программы. Типы данных.

23. MPI. Передача и прием сообщений с блокировкой.

24. MPI. Тупиковые ситуации и способы борьбы с ними. Прием и передача сообщений без блокировки.

25. MPI. Коллективные операции.