**Программа учебной дисциплины «Параллельное программирование»**

Утверждена

Академическим советом ООП

Протокол № от «\_\_»\_\_\_\_\_20\_\_ г.

|  |  |
| --- | --- |
| Автор | Аветисян А.И., член-корр. РАН, проф., д.ф.-.м.н. arut@ispras.ru |
| Число кредитов | 5 |
| Контактная работа (час.) | 64 |
| Самостоятельная работа (час.) | 126 |
| Курс | 1 |
| Формат изучения дисциплины | без использования онлайн курса |

1. **ЦЕЛЬ, РЕЗУЛЬТАТЫ ОСВОЕНИЯ ДИСЦИПЛИНЫ И ПРЕРЕКВИЗИТЫ**

Целью освоения дисциплины «Параллельное программирование» является формирование у студентов теоретических знаний и навыков разработки, исследования производительности, оптимизации и отладки параллельных программ для современных архитектур процессоров и графических акселераторов.

В результате освоения дисциплины студент должен:

**знать:**

- фундаментальные понятия и основные законы параллельного программирования;

- современные проблемы соответствующих разделов системного программирования;

- основные трудности параллельного программирования, причины их возникновения и пути их преодоления;

- основные свойства соответствующего системного программного обеспечения;

- классы параллельных архитектур и особенности разработки параллельных программ для этих классов.

**уметь:**

- понять поставленную задачу;

- использовать свои знания для разработки эффективных параллельных программ;

- оценивать корректность программы, ее масштабируемость;

- самостоятельно проектировать, реализовывать, отлаживать и сопровождать высокоэффективные параллельные программы.

**владеть:**

- навыками освоения большого объема информации и решения задач параллельного программирования ( в том числе, сложных);

- навыками самостоятельной работы и освоения новых дисциплин;

- культурой постановки, анализа и решения математических и прикладных задач, требующих для своего решения использования математических подходов и методов параллельного программирования.

Изучение дисциплины «Параллельное программирование» базируется на следующих дисциплинах:

- Программирование,

- Информатика, математическая логика и теория алгоритмов,

- Построение и анализ алгоритмов,

- Архитектура вычислительных систем,

- Операционные системы.

Для освоения учебной дисциплины студенты должны владеть следующими знаниями и компетенциями:

* Способен рефлексировать (оценивать и перерабатывать, анализировать и синтезировать) освоенные научные методы и способы деятельности для применения на практике.
* Способен предлагать концепции, модели, создавать и апробировать новые способы и инструменты профессиональной деятельности для применения на практике.
* Способен к самостоятельному освоению новых методов исследований, изменению научного и производственного профиля своей деятельности.
* Способен анализировать, верифицировать, оценивать полноту информации, найденной и полученной из различных источников в ходе профессиональной деятельности, при необходимости восполнять и синтезировать недостающую информацию.

1. **СОДЕРЖАНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ**

1. Уровни параллелизма в современных компьютерах. Теоретические подходы: законы Амдаля, Густафсона. Оценки пиковой производительности. Memory wall. Performance/portability tradeoff.

2. Параллелизм в пределах одного контекста выполнения. Параллелизм на уровне команд. VLIW. SIMD. Delayed branches. Компиляторные преобразования повышающие ILP. Modulo scheduling.

3. Векторный параллелизм (SIMD). Ограничения компиляторного анализа. Возможности и ограничения явной векторизации через интринсики. Компромисс между производительностью и переносимостью. Структурирование кода для облегчения компиляторного анализа.

4. Вычисления с плавающей точкой. Принципы IEEE-754. Ограничения и возможности компиляторной трансляции.

5. Иерархия кешей. Оптимизация доступа к памяти. Префетчинг. Cache-aware и cache-oblivious алгоритмы. Возможные компиляторные оптимизации.

6. Параллелизм на многоядерных CPU. Кеши и поддержка согласованности памяти. False sharing.

7. Низкоуровневые примитивы межконтекстного взаимодействия: атомарные операции, семафоры, futex. Поддержка на уровне языка Си и особенности компиляторной поддержки.

8. OpenMP. Классическая функциональность: параллелизм на уровне нитей. Анализ и трансляция OpenMP в компиляторе.

9. Анализ производительности на CPU. Основные инструменты разработчика: valgrind (cachegrind), perf, ocperf. Использование компиляторных возможностей.

10. Параллелизм на графических акселераторах. Явные интерфейсы программирования: CUDA, OpenCL. Инструменты разработки в CUDA.

11. Оптимизация в CUDA. Оптимизация доступа к памяти. Компромисс между ILP, регистровым давлением, TLP. Warp-synchronous programming. Сравнение с подходами, применяемыми на CPU.

12. OpenACC и OpenMP 4.0: параллелизм для акселераторов. Подходы к трансляции кода в различных реализациях. Специализация OpenMP-кода для акселераторов.

1. **ОЦЕНИВАНИЕ**

В рамках курса слушателям предлагается выполнить контрольную работу, домашнее задание, сдать промежуточный и итоговый экзамены. Оценки за контрольную работу, за домашнее задание и за каждый экзамен выставляются по 10-ти балльной шкале.

Оценка за курс складывается из оценок за выполнение контрольной работы *Оконт* (10 баллов), домашнего задания *Одом* (10 баллов) и оценки за итоговый экзамен *Оэкз* (10 баллов).

В диплом выставляется результирующая оценка по учебной дисциплине, которая формируется по следующей формуле:

*Орезульт = 0,3\*Оконт + 0,3\*Одом + 0,4\*Оэкз*

1. **ПРИМЕРЫ ОЦЕНОЧНЫХ СРЕДСТВ**
   1. Примеры заданий для контрольной работы

Напишите функцию, вычисляющую индекс максимального элемента в массиве с эффективным использованием векторных инструкций.

Укажите ошибки работы с общими данными в приведенном OpenMP-коде.

Предложите возможные способы повышения производительности приведенной программы, использующей интерфейс CUDA.

* 1. Примеры контрольных вопросов для экзамена

1. Параллелизм в пределах одного процессорного ядра и контекста выполнения: архитектурные возможности и компиляторные подходы к повышению параллелизма на уровне команд.

2. SIMD-расширения: поддержка в ОС и компиляторах, доступные возможности использования (ассемблер, расширения языка, автовекторизация).

3. Вычисления с плавающей точкой. Принципы IEEE-754. Ограничения и возможности компиляторной трансляции.

4. Организация кеш-памяти. Автоматический и явный префетчинг. Cache-aware и cache-oblivious алгоритмы. Инструмент pahole.

5. Поддержка согласованности кешей на многоядерных процессорах. Эффект false sharing.

6. Низкоуровневые средства взаимодействия параллельных потоков: атомарные операции, futex. Поддержка атомарных операций в языке Си.

7. OpenMP. Основные принципы (fork-join параллелизм на общей памяти, аннотации кода в виде прагм). Компиляторная трансляция OpenMP-конструкций.

8. Профилирование программ: через инструментирование кода, на модельном процессоре (Cachegrind, Callgrind). Профилирование с помощью аппаратных счетчиков.

9. Вычисления на графических акселераторах. Иерархия параллелизма вычислительных модулей и ее отражение в модели программирования.

1. **РЕСУРСЫ**
   1. **Основная литература**
   2. 1. Paul McKenney. Is Parallel Programming Hard, And, If So, What Can You Do About It?
   3. **Дополнительная литература**
2. William Kahan. Why do we need a floating-point arithmetic standard?
3. David Kirk, Wen-mei Hwu. Programming Massively Parallel Processors, Second Edition: A Hands-on Approach
4. David Paterson. Computer Architecture, Fifth Edition: A Quantitative Approach.
5. Calvin Lin, Lawrence Snyder. Principles of Parallel Programming, 1st Edition.
6. Brendan Gregg. Systems Performance: Enterprise and the Cloud, 1st Edition.
7. Barbara Chapman, Gabriele Jost, Ruud van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming.
8. Michael Quinn. Parallel Programming in C with MPI and OpenMP.
9. Randy Allen, Ken Kennedy. Optimizing Compilers for Modern Architectures: A Dependence-based Approach, 1st Edition.
   1. **Программное обеспечение**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Условия доступа** |
| 1. | Пакет MinGW или дистрибутив Linux с инструментами разработки (GCC, GDB, linux-perf, Valgrind) | *Свободно распространяемое ПО* |
| 2. | Пакет CUDA Toolkit | *Свободно распространяемое ПО* |

* 1. **Профессиональные базы данных, информационные справочные системы,   
     интернет-ресурсы (электронные образовательные ресурсы)**

* 1. **Материально-техническое обеспечение дисциплины**

Учебные аудитории для лекционных занятий по дисциплине обеспечивают использование и демонстрацию тематических иллюстраций, соответствующих программе дисциплины в составе:

ПЭВМ с доступом в Интернет (операционная система, офисные программы, антивирусные программы);

мультимедийный проектор с дистанционным управлением.