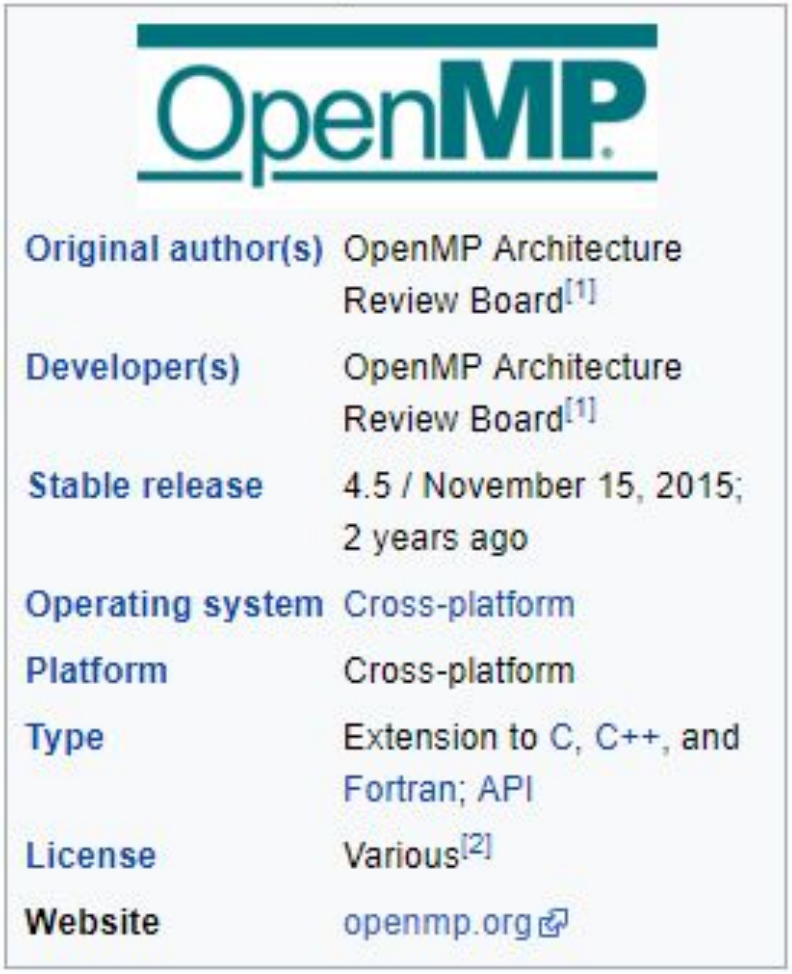
Технологии параллельного программирования

Соколова Анастасия ЭВМб-14-1

ТЕХНОЛОГИИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

- OpenMP
- TBB
- MPI
- CUDA
- OpenCL
- OpenACC
- Chapel

OpenMP



- [3] https://en.wikipedia.org/wiki/OpenMP

OpenMP

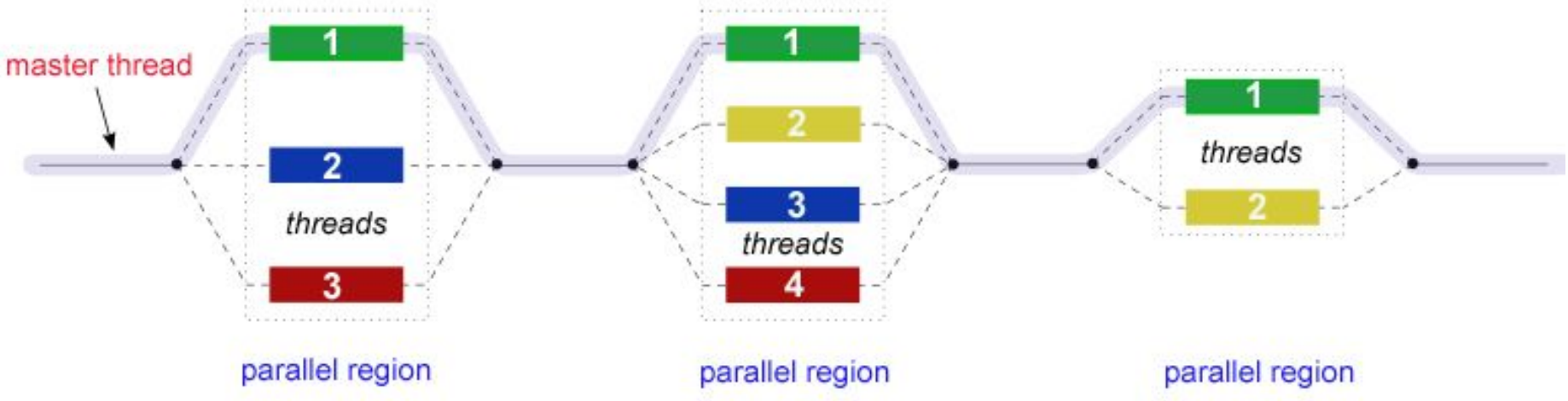
Разработку спецификации OpenMP ведут несколько крупных производителей ВТ и ПО

AMD, IBM, Intel,
Cray, HP, Fujitsu,
Nvidia, NEC,
Red Hat, Texas Instruments,
Oracle Corporation

Их работа регулируется некоммерческой организацией OpenMP Architecture Review Board (ARB)

OpenMP

- OpenMP uses a portable, scalable model with a simple interface for developing parallel applications for platforms ranging from the **standard desktop computer to the supercomputer.**



OpenMP

- По умолчанию в компиляторах поддержка OpenMP выключена. Для её включения следует использовать дополнительную опцию
  gcc, g++, gfortran: опция -fopenmp
- icc, icpc, ifort: опция -openmp под linux, /Qopenmp под windows
- xlc, xlc++, xlf: опция -qsmp=omp
- MSVC: опция /openmp
- Borland: поддержка OpenMP отсутствует

OpenMP

C

```c
/* Демонстрационный код, не имеющий полезного смысла */

#include <stdio.h>

int main (void)
{
  printf ("serial region 1\n");

#pragma omp parallel
  {
    printf ("parallel region\n");
  }

  printf ("serial region 2\n");

  return 0;
}
```

```
$ gcc t.c
$ ./a.out
serial region 1
parallel region
serial region 2

$ gcc -fopenmp t.c
$ ./a.out
serial region 1
parallel region
parallel region
parallel region
parallel region
parallel region
serial region 2
```

C

```c
/* Демонстрационный код, не имеющий полезного смысла */

#include <stdio.h>

int main (void)
{
  printf ("serial region 1\n");

#pragma omp parallel
  {
    printf ("a");
    printf ("b");
    printf ("c");
    printf ("d");
    printf ("e");
    printf ("f");
    printf ("g");
    printf ("h");
    printf ("\n");
  }

  printf ("serial region 2\n");

  return 0;
}
```

Код:

```
$ gcc -fopenmp t.c
$ ./a.out
serial region 1
abcdaebfacagbdbhcec
dfdegefhfg
ghh

serial region 2
```


http://www.cyberforum.ru/blogs/18334/blog2965.html

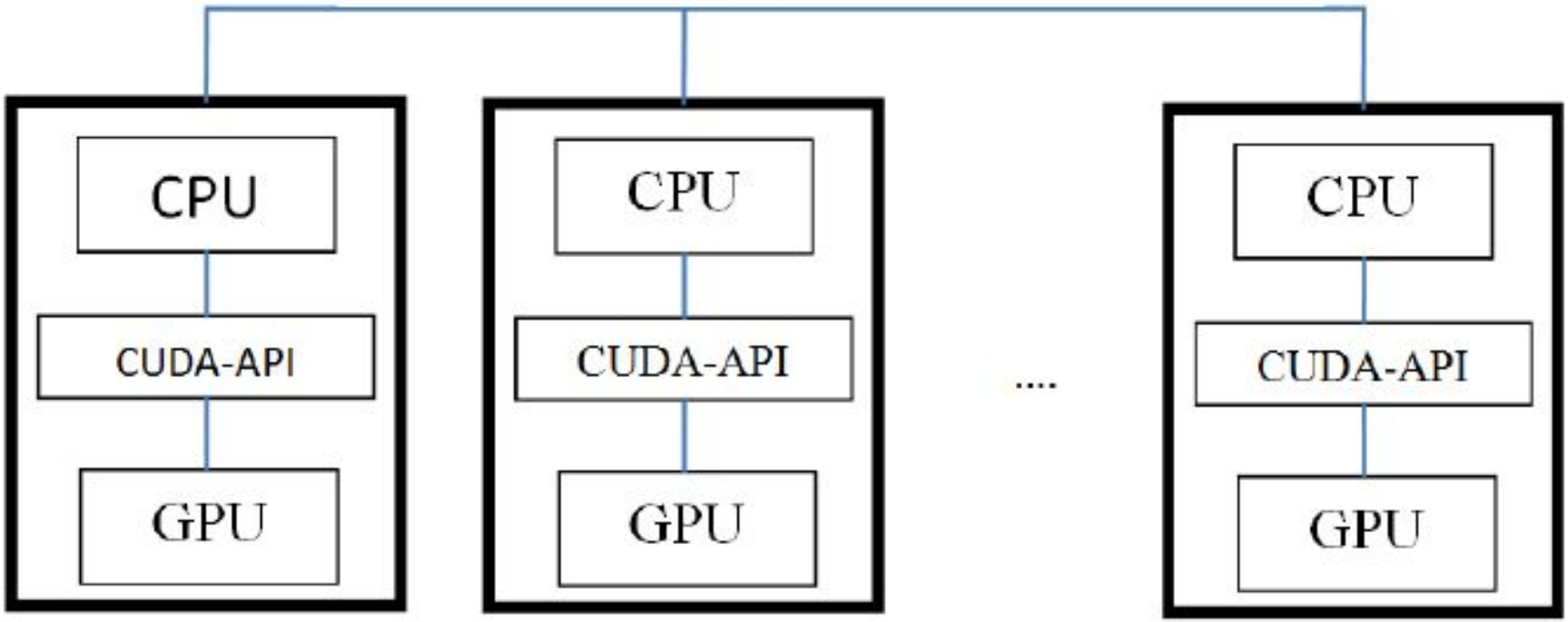
- ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ OPEN MP НА C++ СМОТРИ ТУТ

http://www.cyberforum.ru/blogs/18334/blog2965.html

- **Message Passing Interface** (MPI, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Разработан Уильямом Гроуппом, Эвином Ласком и другими.
- MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для кластеров и суперкомпьютеров . Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.
- **Существуют реализации MPI для языков Фортран, Java, Си и Си++.**

MPI

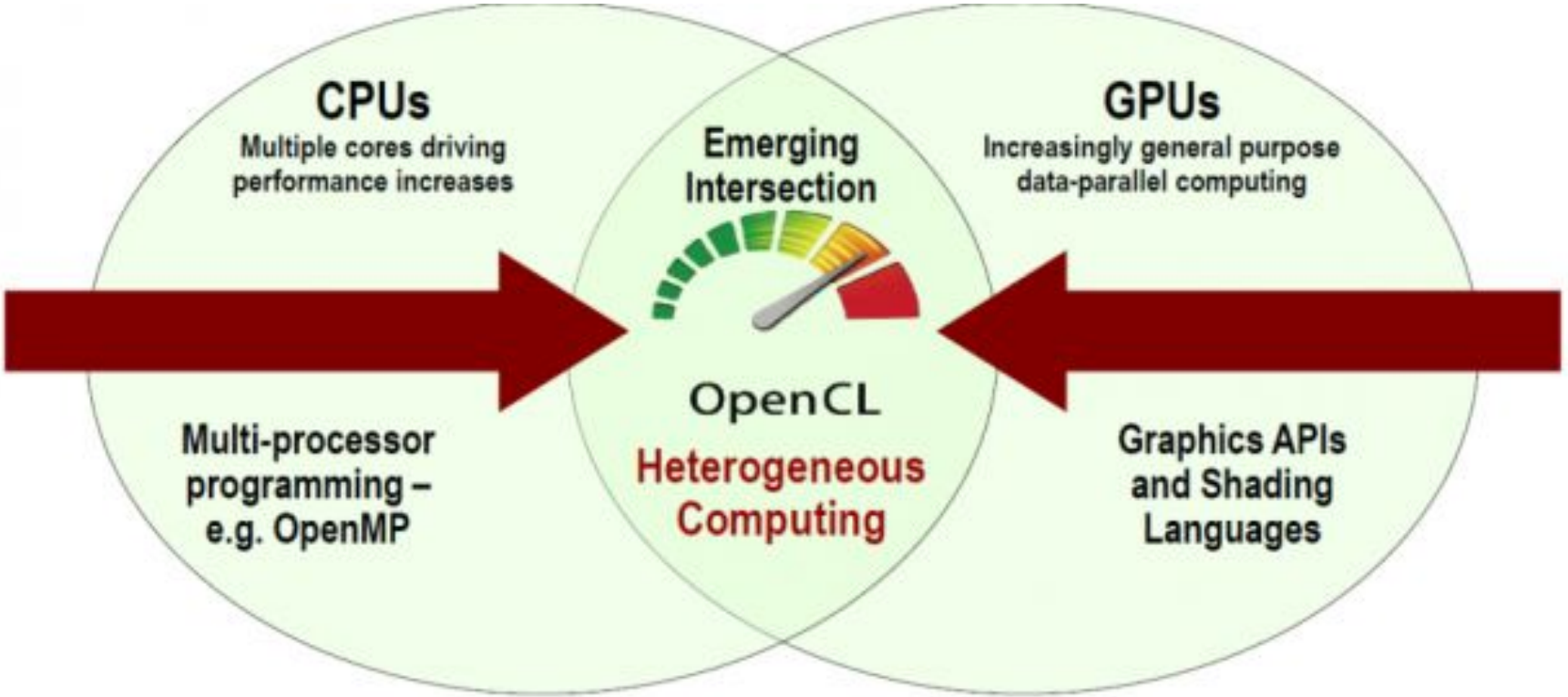
MPI communication between nodes



- В первую очередь MPI ориентирован на системы **с распределенной памятью**, то есть когда затраты на передачу данных велики, в то время как OpenMP ориентирован на системы **с общей памятью** (многоядерные с общим кешем). Обе технологии могут использоваться совместно, чтобы оптимально использовать в кластере многоядерные системы.

INTEL® THREADING BUILDING BLOCKS (INTEL® TBB) BOF SESSION

- **Intel Threading Building Blocks** (также известная как **TBB**) — кроссплатформенная библиотека шаблонов C++, разработанная компанией Intel для параллельного программирования.
- Версия 1.0 была выпущена фирмой Интел 29 августа 2006, через год после выпуска своего первого двуядерного процессора Pentium D.
- Поддерживаемые операционные системы
  - Коммерческая версия TBB 4.0 поддерживает Microsoft Windows (XP или выше),
  - Mac OS X(версия 10.5.8 или выше) и
  - Linux, используя различные компиляторы (Visual C++ (версия 8.0 или выше, только на Windows), Intel C++ compiler (версия 11.1 или выше) или GNU Compiler Collection (gcc, версия 3.4 и выше)). Кроме того, сообщество открытой версии TBB портировало её на Sun Solaris, PowerPC, Xbox 360, QNX Neutrino, и FreeBSD.


CPUs
Multiple cores driving performance increases
Emerging Intersection
GPUs
Increasingly general purpose data-parallel computing
OpenCL
Heterogeneous Computing
Multi-processor programming – e.g. OpenMP
Graphics APIs and Shading Languages

OpenCL

- The open standard for parallel programming of heterogeneous systems
- OpenCL ™ (Open Computing Language) - это открытый, стандарт для кросс-платформенного параллельного программирования различных процессоров, имеющихся на персональных компьютерах, серверах, мобильных устройствах и встроенных платформах. OpenCL значительно улучшает скорость и отзывчивость широкого спектра приложений во многих рыночных категориях, включая игровые и развлекательные звания, научное и медицинское программное обеспечение, профессиональные креативные инструменты, обработку зрения и обучение и вывод нейронной сети.

https://www.khronos.org/opencl/

OpenCL – как и с чего начать можно найти примеры тут (C++):

https://habrahabr.ru/post/261323/

CUDA vs OpenCL

- CUDA
- это архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров).

- Поддержка архитектур
  - x86, x86-64, Itanium, SpursEngine (Cell), NVidia GPU, AMD GPU, VIA (S3 Graphics) GPU.
- Для каждого из этих типов процессов существует свой SDK (ну кроме разве что VIA), свой язык программирования и программная модель.

CUDA vs OpenCL

- То есть если Вы захотите чтобы ваш движок рендеринга или программа расчета нагрузок на крыло боинга 787 работала на простой рабочей станции, суперкомпьютере BlueGene, или компьютере оборудованном двумя ускорителями NVidia Tesla – Вам будет необходимо переписывать достаточно большую часть программы, так как каждая из платформ в силу своей архитектуры имеет набор жестких ограничений.

CUDA vs OpenCL

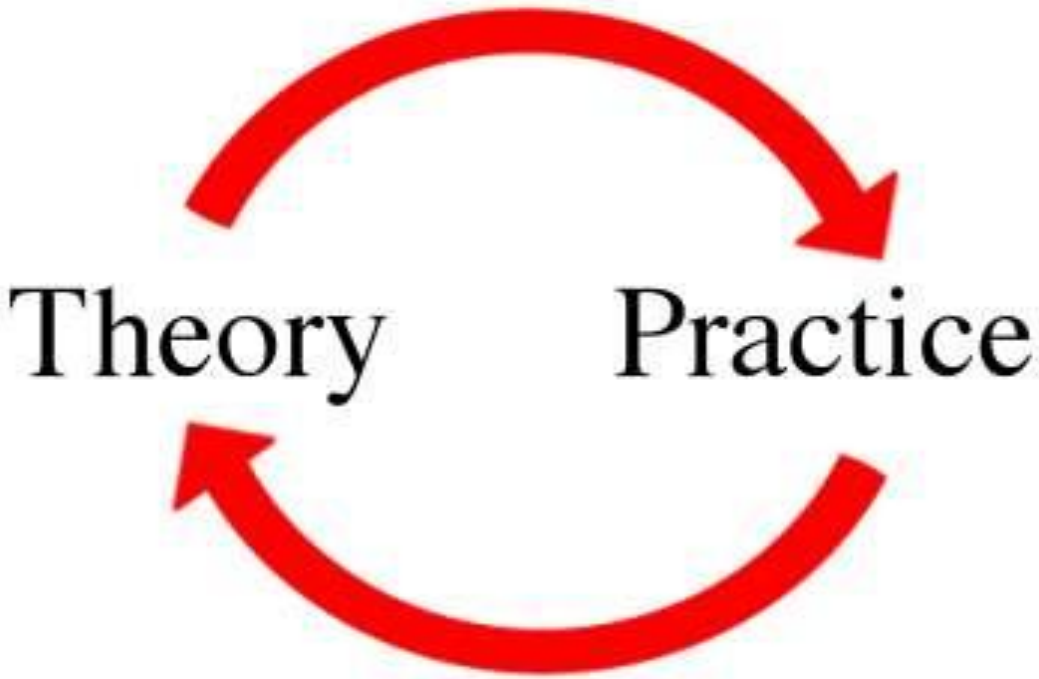
- было решено создать некий единый стандарт для программ, исполняющихся в гетерогенной среде. Это означает, что программа, вообще говоря, должна быть способна исполняться на компьютере, в котором установлены одновременно GPU NVidia и AMD, Toshiba SpursEngine итд.

CUDA vs OpenCL

- Решение проблемы

Для разработки открытого стандарта решили привлечь людей, у которых уже есть опыт в разработке подобного стандарта: Khronos Group, на чьей совести уже OpenGL и OpenML и еще много всего. OpenCL является торговой маркой Apple Inc.

В разработке (и финансировании, конечно же), кроме Apple, участвовали такие компании IT как AMD, IBM, Activision Blizzard, Intel, NVidia и тд.


Theory
Practice

https://habrahabr.ru/post/147796/

- Попробуем решить одну простую задачу с помощью актуальных технологий параллельного программирования (OpenMP, TBB, MPI, CUDA, OpenCL, OpenACC, Chapel).

Вычислим число Пи путем численного интегрирования, будем использовать метод прямоугольников

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$

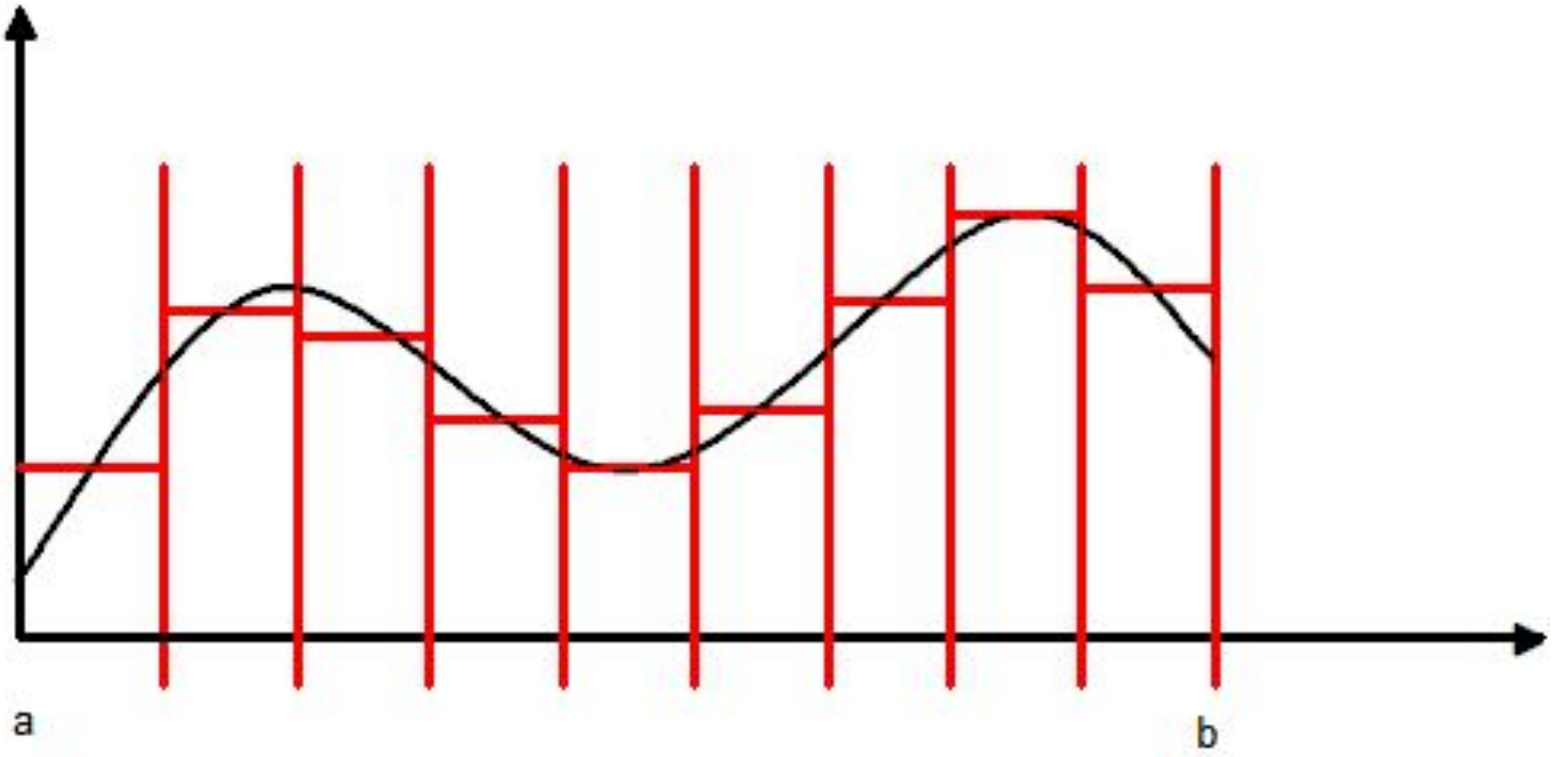
Объявим количество шагов, на которые разобьем интеграл

$$step = \frac{|a - b|}{n}$$

a – левый предел интеграла

b – правый предел интеграла

n – количество отрезков на которые разбиваем ось OX


a
b

Вычислим площадь криволинейной трапеции, посчитав значение функции в каждом прямоугольнике и домножив на основание

```
 //определение значения функции в середине шага
//Умножение значения функции в середине шага на основание
        for (int i=0;i<n;i++){
            x=(i+0.5)*step;
            y=(4/(1+x*x));//вычислили значение функции
            pi=pi+y*step; //умножили значение функции на основание прямоугольника
        }
```

Попробуем решить задачу сначала последовательным алгоритмом, а затем с использованием технологий параллельного программирования. И сравним время выполнения программ

Последовательное программирование

```
package parallelprog;
public class ParallelProg {
    static double x=0;
    static double a=0;//левый предел
    static double b=1;//правый предел
    static double y=0;//значение функции в точке X
    static double pi=0;//значение интеграла
    static double n=1000000;//количество шагов
    static double step=0; //ширина шага
    static long   start = 0;  //время начала выполнения процесса
    static long   finish = 0; //время окончания
    static double   time = 0; //разница времени начала и конца
```

```
 public static void main(String[] args) {
start = System.nanoTime();//Отметим время начала
        step=Math.abs(a-b)/n;
        System.out.println("Начинаем вычислять интеграл");
        //определение значения функции в середине шага
        for (int i=0;i<n;i++){
            x=(i+0.5)*step;
            y=(4/(1+x*x));//вычислили значение функции
            pi=pi+y*step; //умножили значение функции на основание
прямоугольника
        }
        //Закончив вычислять интеграл, проверим прошедшее время
        finish = System.nanoTime();
        time=finish-start;
        //Переводим в секунды
        time= (double)time / 1000000000.0;
        System.out.println("Закончили вычислять интеграл");
        System.out.println("pi="+pi);
        System.out.println("Время выполнения: "+time+" сек");
    }
```

Результат последовательного программирования

Результат №1

```
Время выполнения: 0.15465879сек
```

Результат №2

```
Время выполнения: 0.100236524сек
```

ТЕПЕРЬ ПОПРОБУЕМ РЕШИТЬ ЗАДАЧУ С ПОМОЩЬЮ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

- Самая доступная простым пользователям параллельная архитектура — это обычный многоядерный процессор или несколько процессоров на одной материнской плате.

- Одно ядро также способно исполнять параллельно несколько потоков — такой режим называется псевдо-параллельным или конкурентным. Ядро переключается между процессами, выделяя каждому квант времени. В принципе такой режим выполнения уже может привести к росту производительности за счет сокрытия латентности памяти.

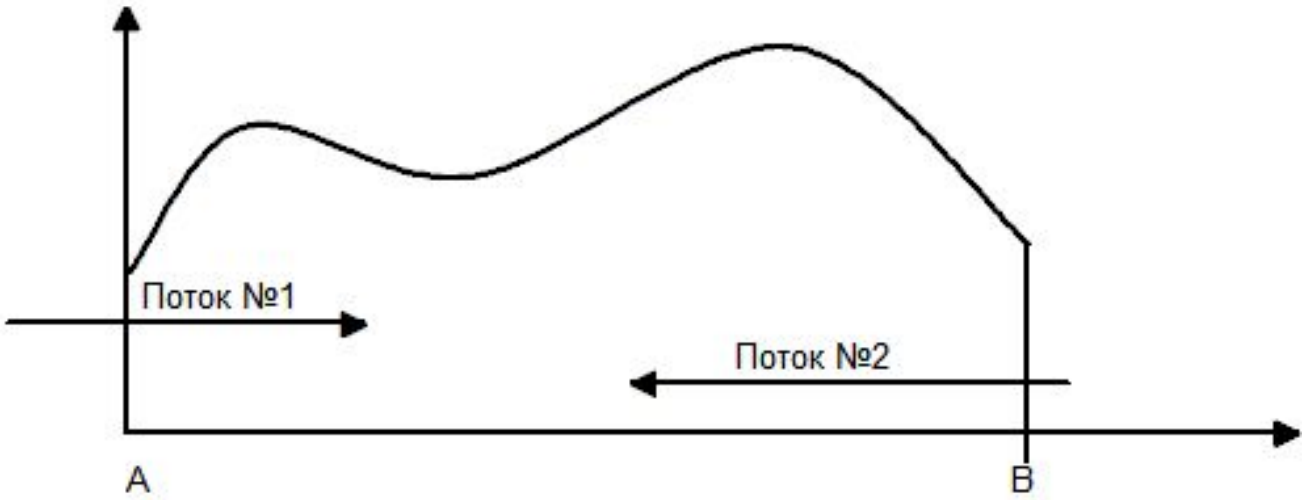
- Латентность (задержка) - это время, которое затрачивается на чтение из памяти одного слова данных (восьми байт) . Чем ниже латентность оперативной памяти, тем меньше центральный процессор будет находиться в состоянии простоя.

- Самый «исторический» способ использовать сразу несколько ядер на процессоре — это механизм потоков операционной системы, который существовал задолго до истинно-параллельных процессоров для конкретности хотя бы ради более удобного написания программ.

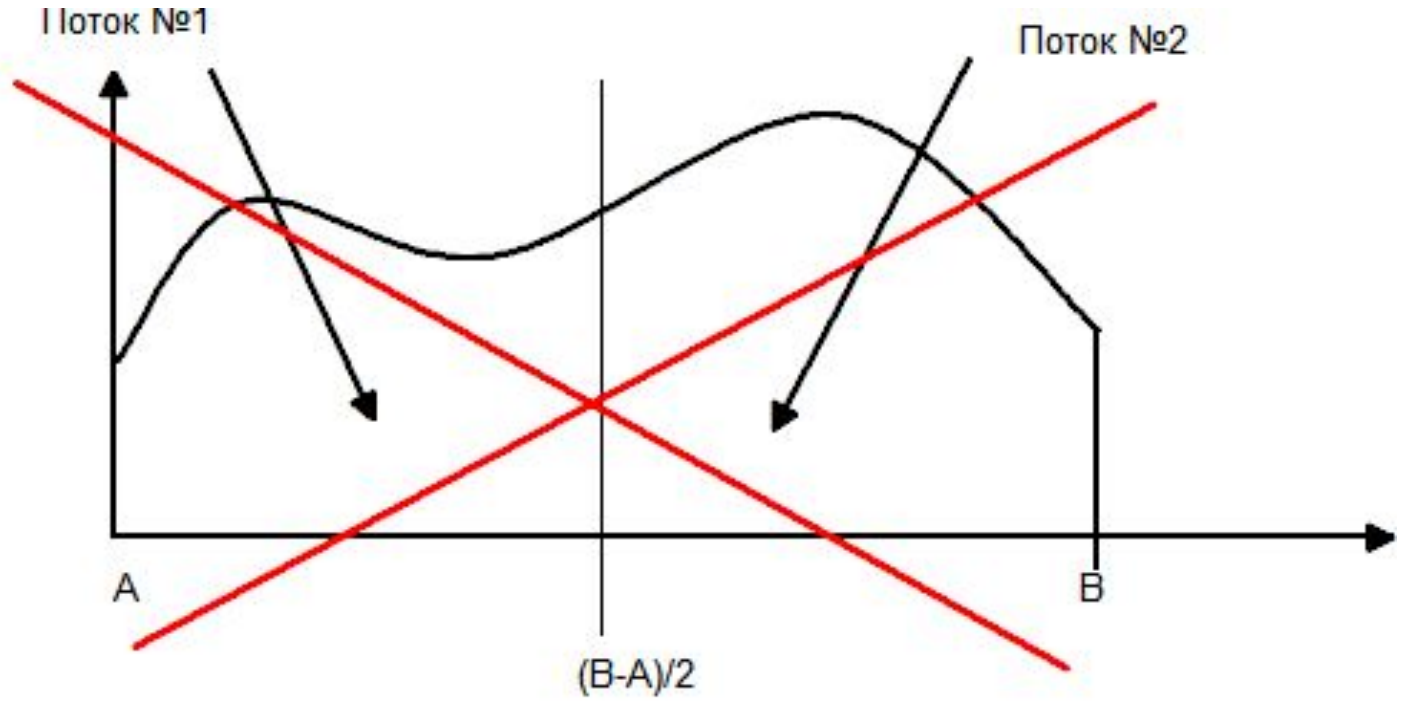
- С точки зрения программиста важно то, что параллельные потоки, исполняемые на разных ядрах или процессорах видят одно и то же адресное пространство, т.е нет нужды явно передавать данные между потоками. Зато если вдруг разные потоки пишут\читают одну и ту же переменную — то придётся озаботиться синхронизацией.

? ВЕРНЕМСЯ К НАШЕЙ ЗАДАЧЕ: ВЫЧИСЛЕНИЕ ИНТЕГРАЛА

- РАЗОБЬЕМ ИНТЕГРАЛ НА ЛЕВУЮ И ПРАВУЮ ЧАСТИ. ПЕРВЫЙ ПОТОК БУДЕТ ВЫЧИСЛЯТЬ ИНТЕГРАЛ СЛЕВА НАПРАВО, НАЧИНАЯ ОТ ТОЧКИ А, ДРУГОЙ ПОТОК БУДЕТ ВЫЧИСЛЯТЬ ИНТЕГРАЛ СПРАВА НАЛЕВО, НАЧИНАЯ ОТ ПРАВОЙ ГРАНИЦЫ

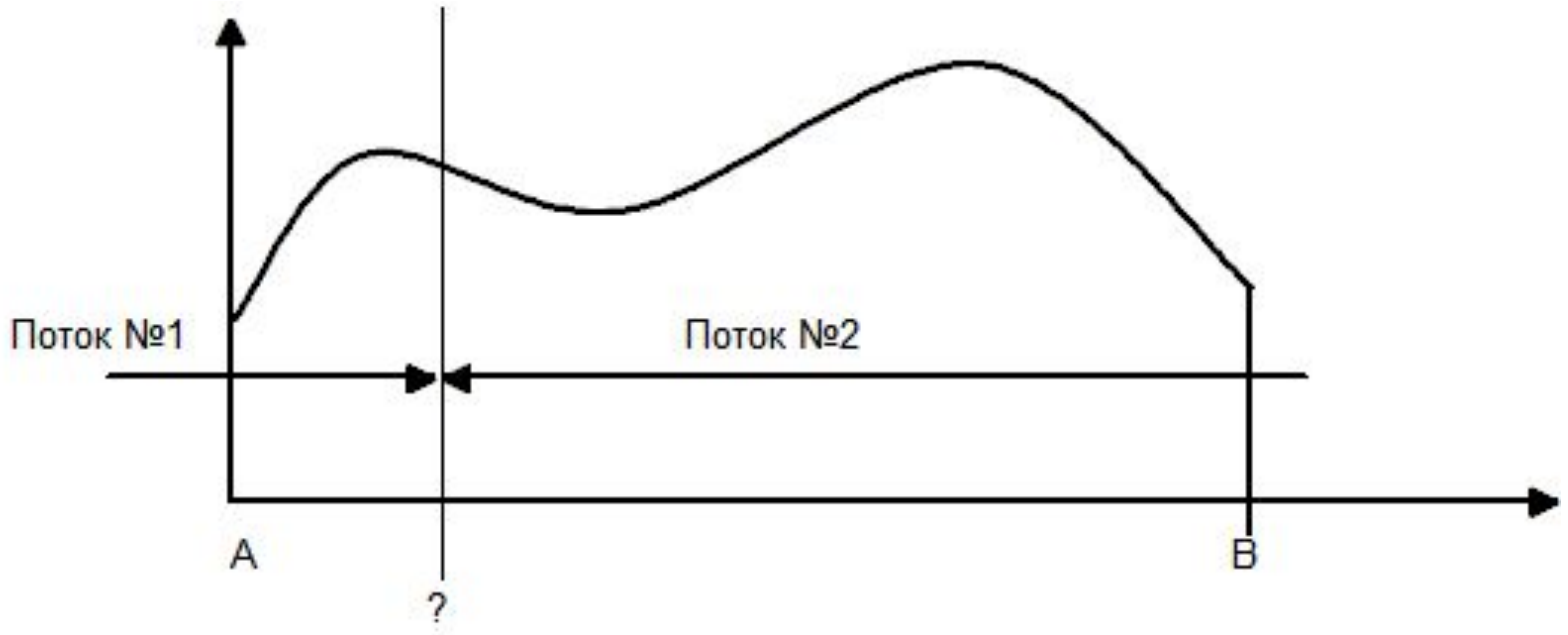


- ВОПРОС: Почему будет неэффективным решением разбить интервал AB на равные доли и отдать параллельным потокам на исполнение, чтобы один поток вычислял интеграл от A до |A-B|/2, а второй от B до |A-B|/2 ????



- ОТВЕТ: Если один из потоков посчитает свою часть раньше — то соответствующее ядро будет простаивать, т.е. мы потеряем производительность.

- Поэтому оба потока будут вычислять интеграл с разной скоростью, пока не встретятся лицом к лицу

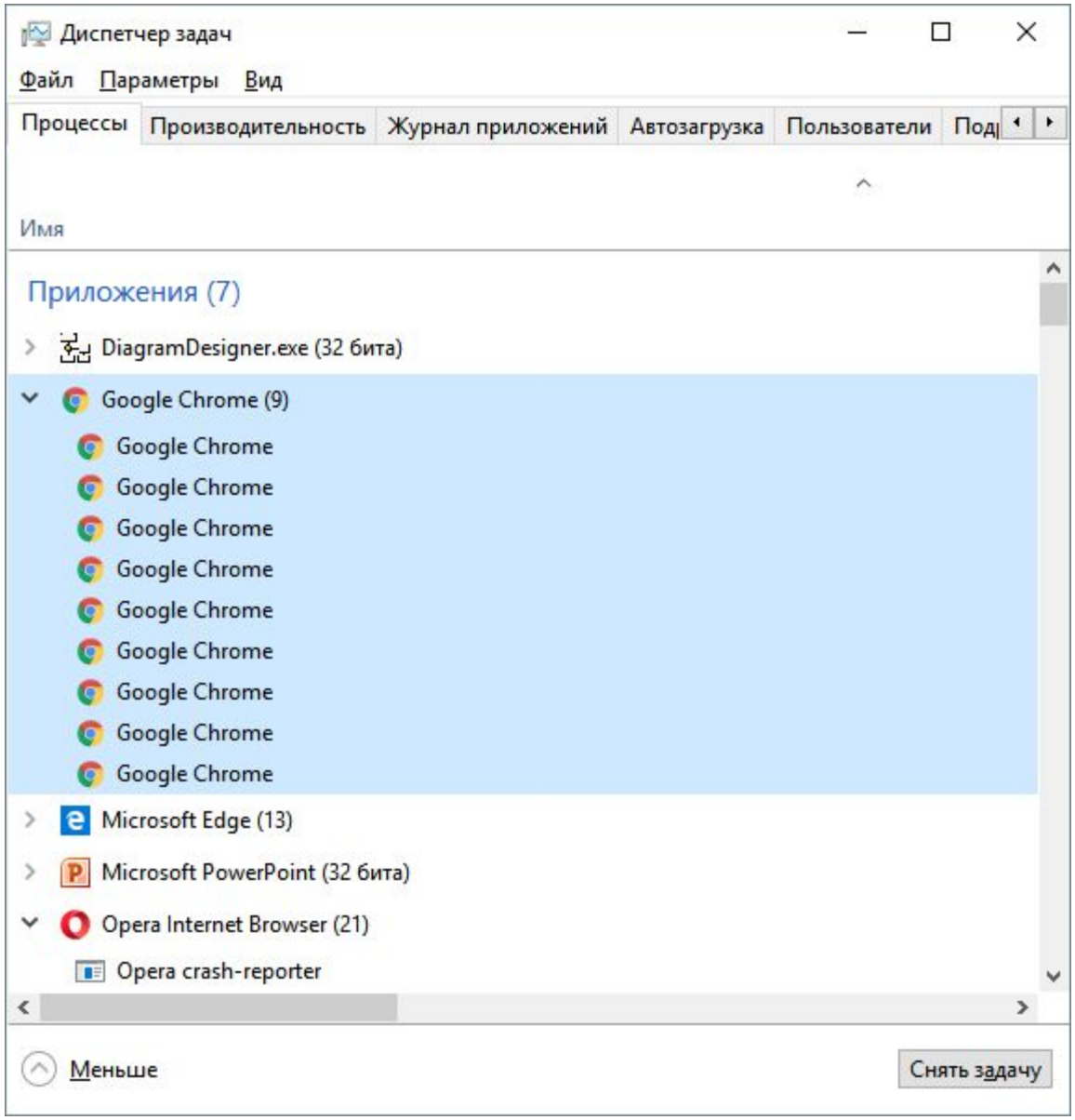


?Что касаемо программы

Многопоточность в Java

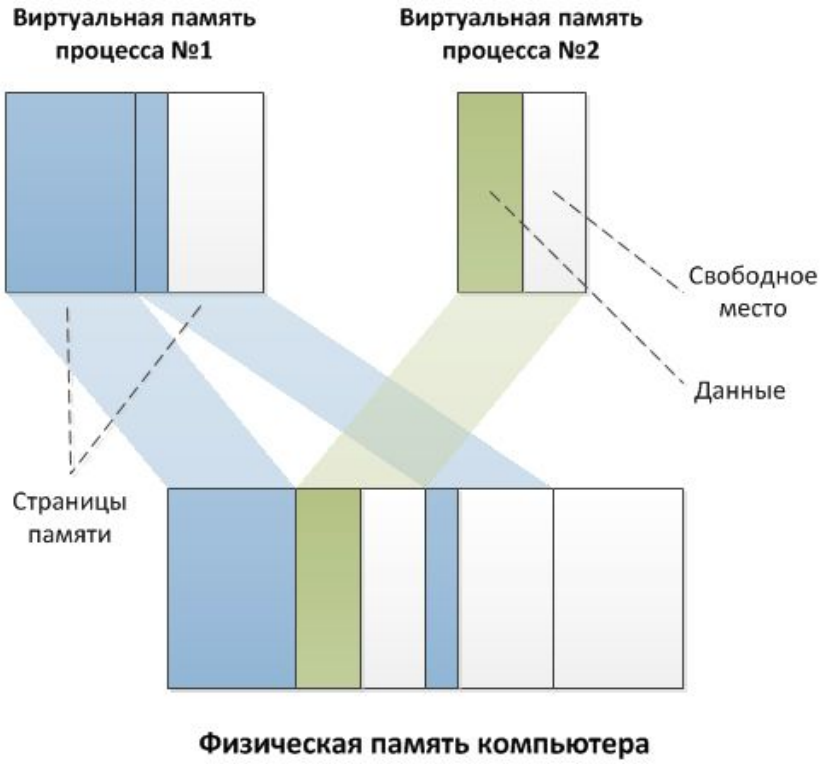
Процесс/Stream

- Процесс — это совокупность кода и данных, разделяющих общее виртуальное адресное пространство (VAS). Чаще всего одна программа состоит из одного процесса
- Процессы изолированы друг от друга, поэтому прямой доступ к памяти чужого процесса невозможен.
- Для каждого процесса ОС создает VAS, к которому процесс имеет прямой доступ. Это пространство принадлежит процессу, содержит только его данные и находится в полном его распоряжении. ОС же отвечает за то, как виртуальное пространство процесса проецируется на физическую память.



Браузер Chrome создает отдельный процесс для каждой вкладки, что дает ему некоторые преимущества, вроде независимости вкладок друг от друга

? ОС оперирует так называемыми страницами памяти, которые представляют собой просто область определенного фиксированного размера. Если процессу становится недостаточно памяти, система выделяет ему дополнительные страницы из физической памяти. Страницы виртуальной памяти могут проецироваться на физическую память в произвольном порядке.



При запуске программы ОС создает процесс, загружая в его адресное пространство код и данные программы, а затем запускает главный поток созданного процесса.

Поток/Нить/ Thread

- Один поток – это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.
- Следует отдельно обговорить фразу «параллельно с другими потоками». Известно, что на одно ядро процессора, в каждый момент времени, приходится один поток. То есть одноядерный процессор может обрабатывать команды только последовательно, по одной за раз (в упрощенном случае). Однако запуск нескольких параллельных потоков возможен и в системах с одноядерными процессорами. В этом случае система будет периодически переключаться между потоками, поочередно давая выполняться то одному, то другому потоку. Такая схема называется псевдо-параллелизмом.

Псевдо-параллелизм.

- Система запоминает состояние (контекст) каждого потока, перед тем как переключиться на другой поток, и восстанавливает его по возвращению к выполнению потока. В контекст потока входят такие параметры, как стек, набор значений регистров процессора, адрес исполняемой команды и прочее…
- Проще говоря, при псевдопараллельном выполнении потоков процессор мечется между выполнением нескольких потоков, выполняя по очереди часть каждого из них.

УКРЕПИМ ЗНАНИЯ:

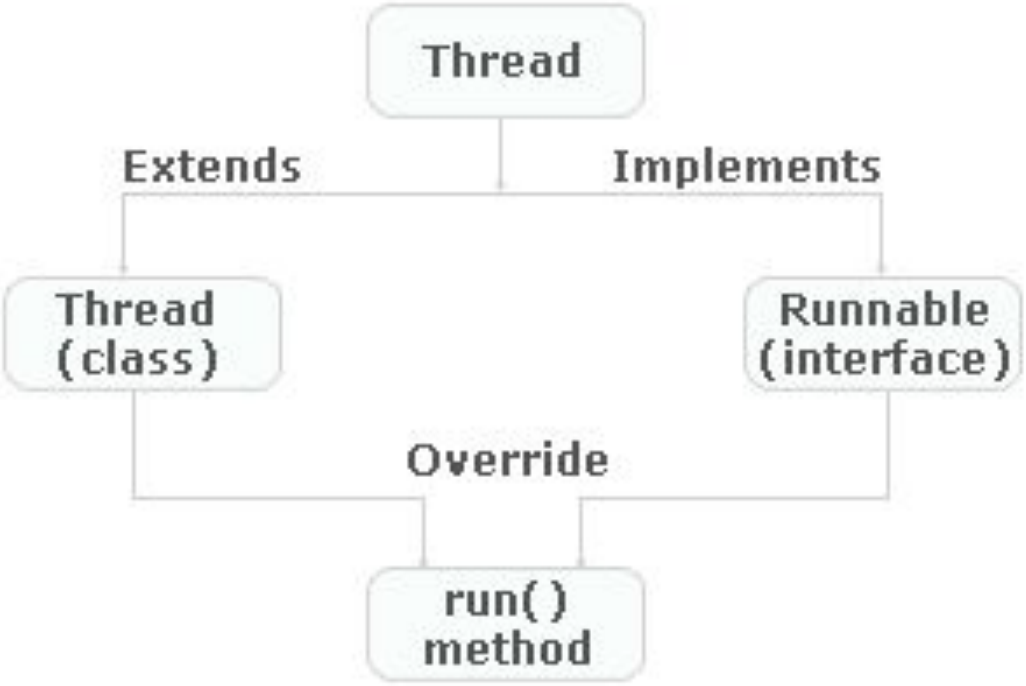
- Процессы изолированы друг от друга, поэтому прямой доступ к памяти чужого процесса невозможен.
- параллельные потоки, исполняемые на разных ядрах или процессорах видят одно и то же адресное пространство, т.е нет нужды явно передавать данные между потоками.

JAVA

- Каждый процесс имеет хотя бы один выполняющийся поток. Тот поток, с которого начинается выполнение программы, называется главным. В языке Java, после создания процесса, выполнение главного потока начинается с метода main(). Затем, по мере необходимости, запускаются другие побочные потоки.

  В языке Java поток представляется в виде объекта-потомка класса Thread.

КАК СОЗДАТЬ ПОТОКИ В JAVA



https://habrahabr.ru/post/164487/

СПОСОБ №1 СОЗДАТЬ ПОТОК В JAVA С ПОМОЩЬЮ ИНТЕРФЕЙСА RUNNABLE

```
public class MyThread implements Runnable{
    @Override
    public void run() {  /*Код который должен выполнить    поток*/ }
}
```

```
public class GeneralProcess {
    static MyThread t1, t2;

    public static void main(String[] args) {
        t1 = new MyThread();
         t2 = new MyThread();

         Thread mT1 = new Thread(t1);//создание потока
         mT1.start();//Запуск потока

         Thread mT2 = new Thread(t2);//создание потока
         mT2.start();//Запуск потока
….
}
```

СПОСОБ №2: НАСЛЕДОВАНИЕ ОТ КЛАССА THREAD

```
public class MyThread extends Thread{
    @Override
    public void run() {  /*Код который должен выполнить поток*/ }
}
```

```
public class GeneralProcess {
    static MyThread t1, t2;

    public static void main(String[] args) {
        t1 = new MyThread();
         t2 = new MyThread();

         Thread mT1 = new Thread(t1);//создание потока
         mT1.start();//Запуск потока

         Thread mT2 = new Thread(t2);//создание потока
         mT2.start();//Запуск потока
….
}
```

В ЧЕМ РАЗНИЦА МЕЖДУ РЕАЛИЗАЦИЕЙ ИНТЕРФЕЙСА RUNNABLE И НАСЛЕДОВАНИЕМ ОТ КЛАССА THREAD?

http://qaru.site/questions/57/implements-runnable-vs-extends-thread

ДАВАЙТЕ ИЗУЧИМ КОД:

```
//Implement Runnable Interface...
 class ImplementsRunnable implements Runnable {

private int counter = 0;

public void run() {
    counter++;
    System.out.println("ImplementsRunnable : Counter : " + counter);
 }
}

//Extend Thread class...
class ExtendsThread extends Thread {

private int counter = 0;

public void run() {
    counter++;
    System.out.println("ExtendsThread : Counter : " + counter);
 }
}
```

```
//Use above classes here in main to understand the differences more clearly...
public class ThreadVsRunnable {

public static void main(String args[]) throws Exception {
    // Multiple threads share the same object.
    ImplementsRunnable rc = new ImplementsRunnable();
    Thread t1 = new Thread(rc);
    t1.start();
    Thread.sleep(1000); // Waiting for 1 second before starting next thread
    Thread t2 = new Thread(rc);
    t2.start();
    Thread.sleep(1000); // Waiting for 1 second before starting next thread
    Thread t3 = new Thread(rc);
    t3.start();

    // Creating new instance for every thread access.
    ExtendsThread tc1 = new ExtendsThread();
    tc1.start();
    Thread.sleep(1000); // Waiting for 1 second before starting next thread
    ExtendsThread tc2 = new ExtendsThread();
    tc2.start();
    Thread.sleep(1000); // Waiting for 1 second before starting next thread
    ExtendsThread tc3 = new ExtendsThread();
    tc3.start();
 }
}
```

Вывод указанной программы.

```
ImplementsRunnable : Counter : 1
ImplementsRunnable : Counter : 2
ImplementsRunnable : Counter : 3
ExtendsThread : Counter : 1
ExtendsThread : Counter : 1
ExtendsThread : Counter : 1
```

- В интерфейсе Runnable создается только один экземпляр класса, и он разделяется различными потоками. Таким образом, значение счетчика увеличивается для каждого доступа к потоку.
- В то время как подход класса Thread, вы должны создать отдельный экземпляр для каждого потока. Следовательно, для каждого экземпляра класса выделяется различная память и каждый имеет отдельный счетчик, значение остается таким же, что означает, что приращение не произойдет, потому что ни одна из ссылок на объекты не является такой же.

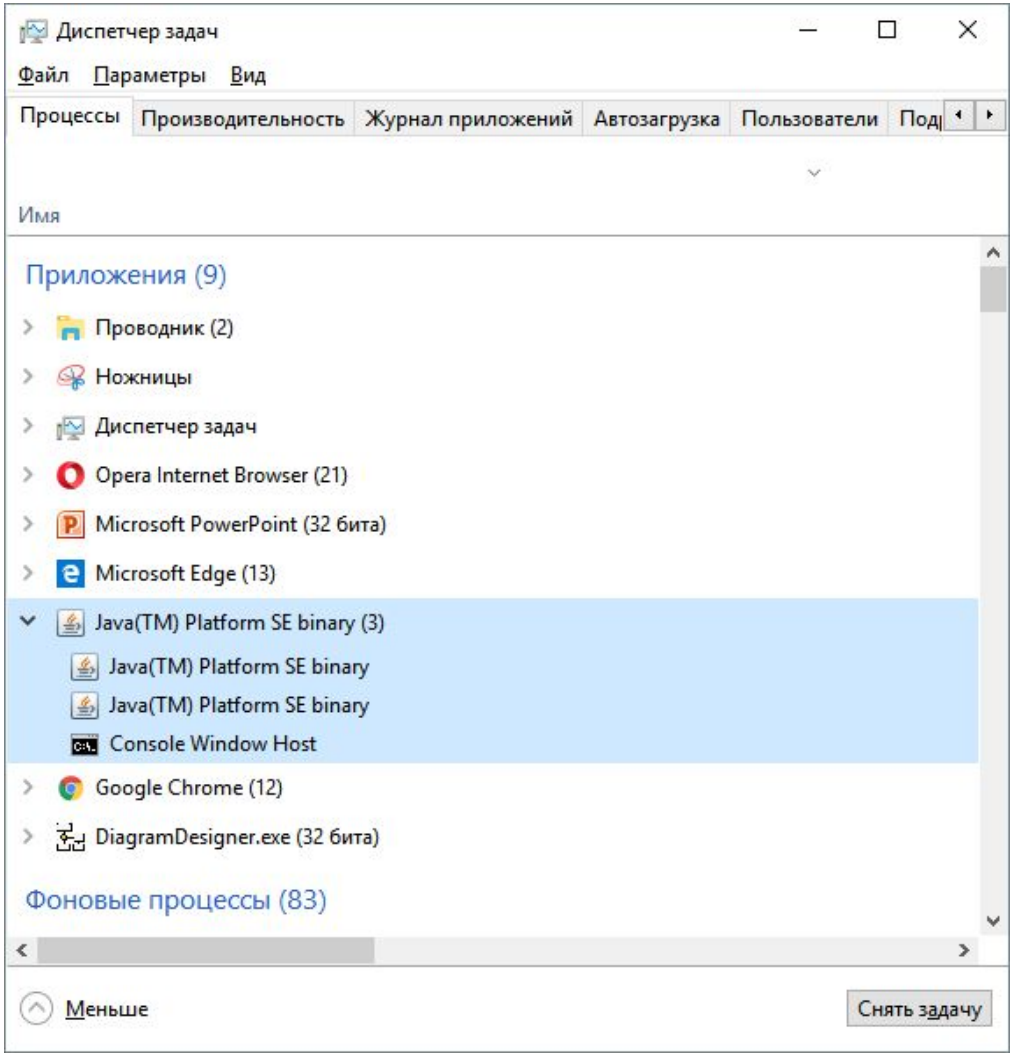
- **Когда использовать Runnable?**
  Используйте интерфейс Runnable, если вы хотите получить доступ к одному и тому же ресурсу из группы потоков. Избегайте использования класса Thread здесь, поскольку создание нескольких объектов потребляет больше памяти

- Если вы хотите реализовать или расширить какой-либо другой класс, тогда Runnable интерфейс наиболее предпочтителен другим, если вы не хотите, чтобы какой-либо другой класс расширялся или реализовывался, тогда класс Thread предпочтительнее

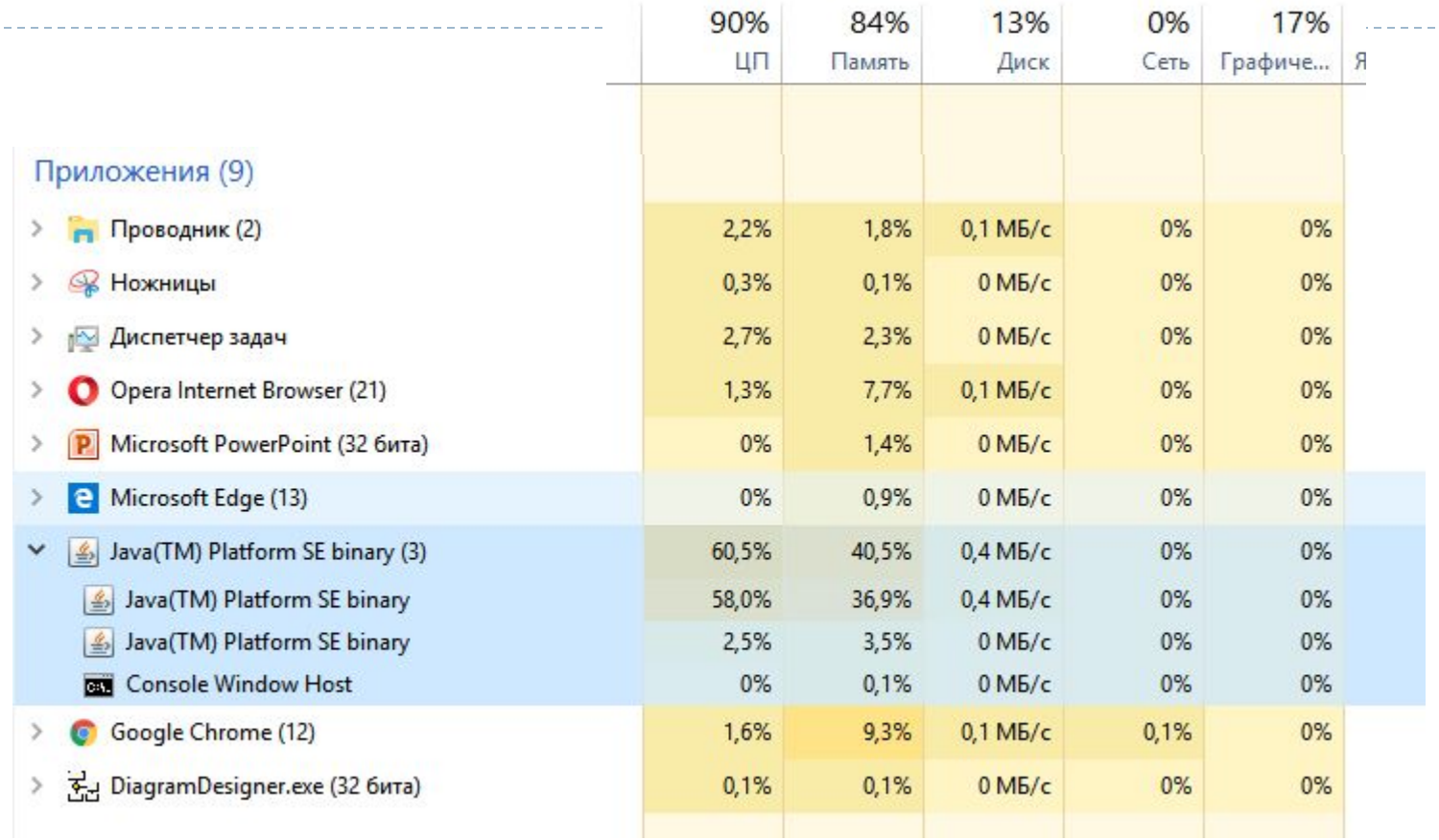
- Когда вы реализуете интерфейс Runnable, вы можете сохранить пространство для своего класса для расширения любого другого класса.
- Java не поддерживает множественное наследование, а это значит, что вы можете расширять только один класс на Java, поэтому, как только вы расширили класс Thread, вы потеряли свой шанс и не можете расширять или наследовать другой класс на Java.

? В задачи нашего курса не входит освоение ЯВУ, давайте все же вернемся к нашей задаче, которую мы хотели реализовать параллельно: задача вычисления интеграла

Мы создали 2 потока, которые вычисляют интеграл.



борются за один и тот же ресурс – ядро. Пример №1

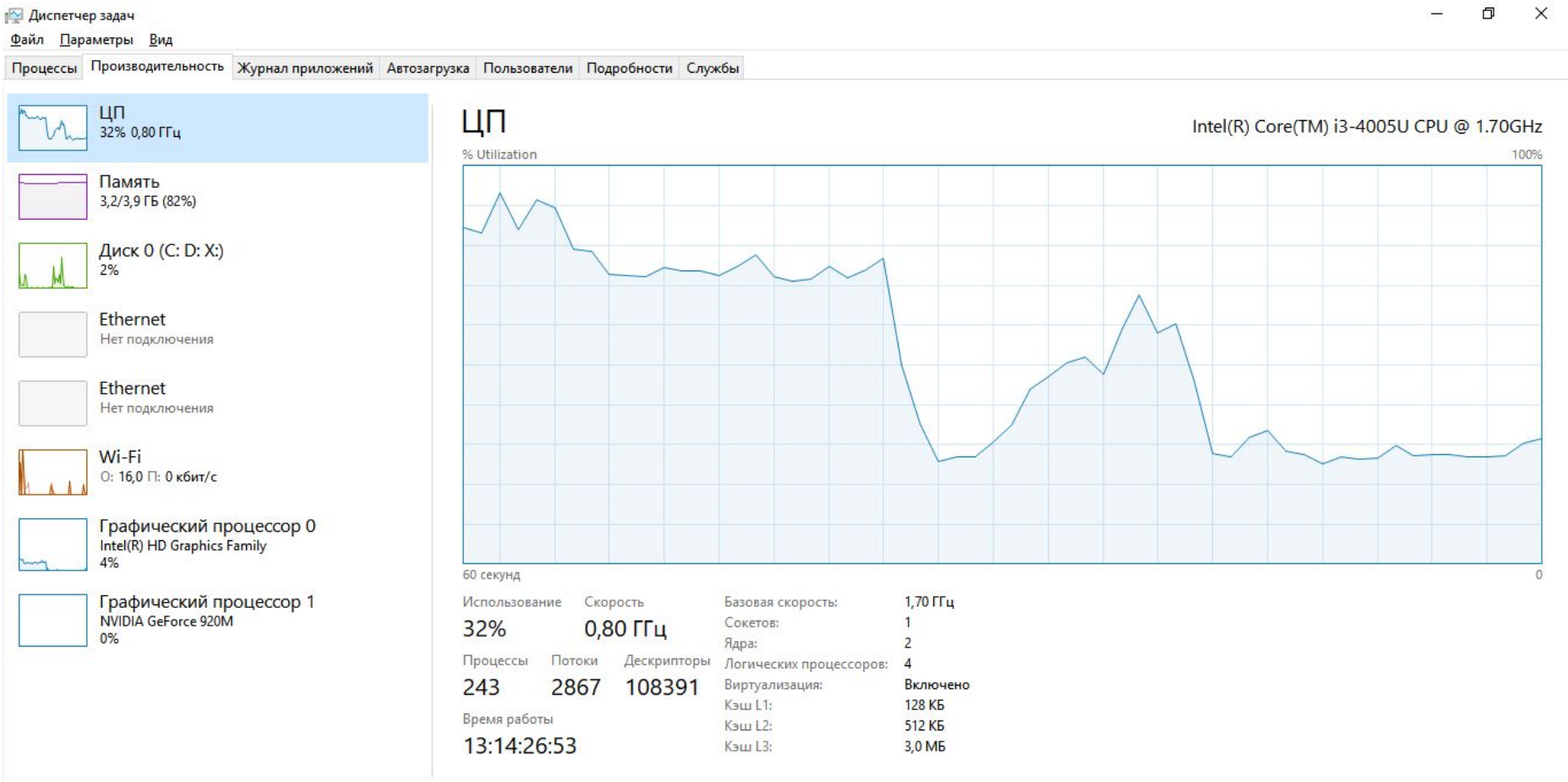


	90% ЦП	84% Память	13% Диск	0% Сеть	17% Графиче...
Приложения (9)					
Проводник (2)	2,2%	1,8%	0,1 МБ/с	0%	0%
Ножницы	0,3%	0,1%	0 МБ/с	0%	0%
Диспетчер задач	2,7%	2,3%	0 МБ/с	0%	0%
Opera Internet Browser (21)	1,3%	7,7%	0,1 МБ/с	0%	0%
Microsoft PowerPoint (32 бита)	0%	1,4%	0 МБ/с	0%	0%
Microsoft Edge (13)	0%	0,9%	0 МБ/с	0%	0%
Java(TM) Platform SE binary (3)	60,5%	40,5%	0,4 МБ/с	0%	0%
Java(TM) Platform SE binary	58,0%	36,9%	0,4 МБ/с	0%	0%
Java(TM) Platform SE binary	2,5%	3,5%	0 МБ/с	0%	0%
Console Window Host	0%	0,1%	0 МБ/с	0%	0%
Google Chrome (12)	1,6%	9,3%	0,1 МБ/с	0,1%	0%
DiagramDesigner.exe (32 бита)	0,1%	0,1%	0 МБ/с	0%	0%

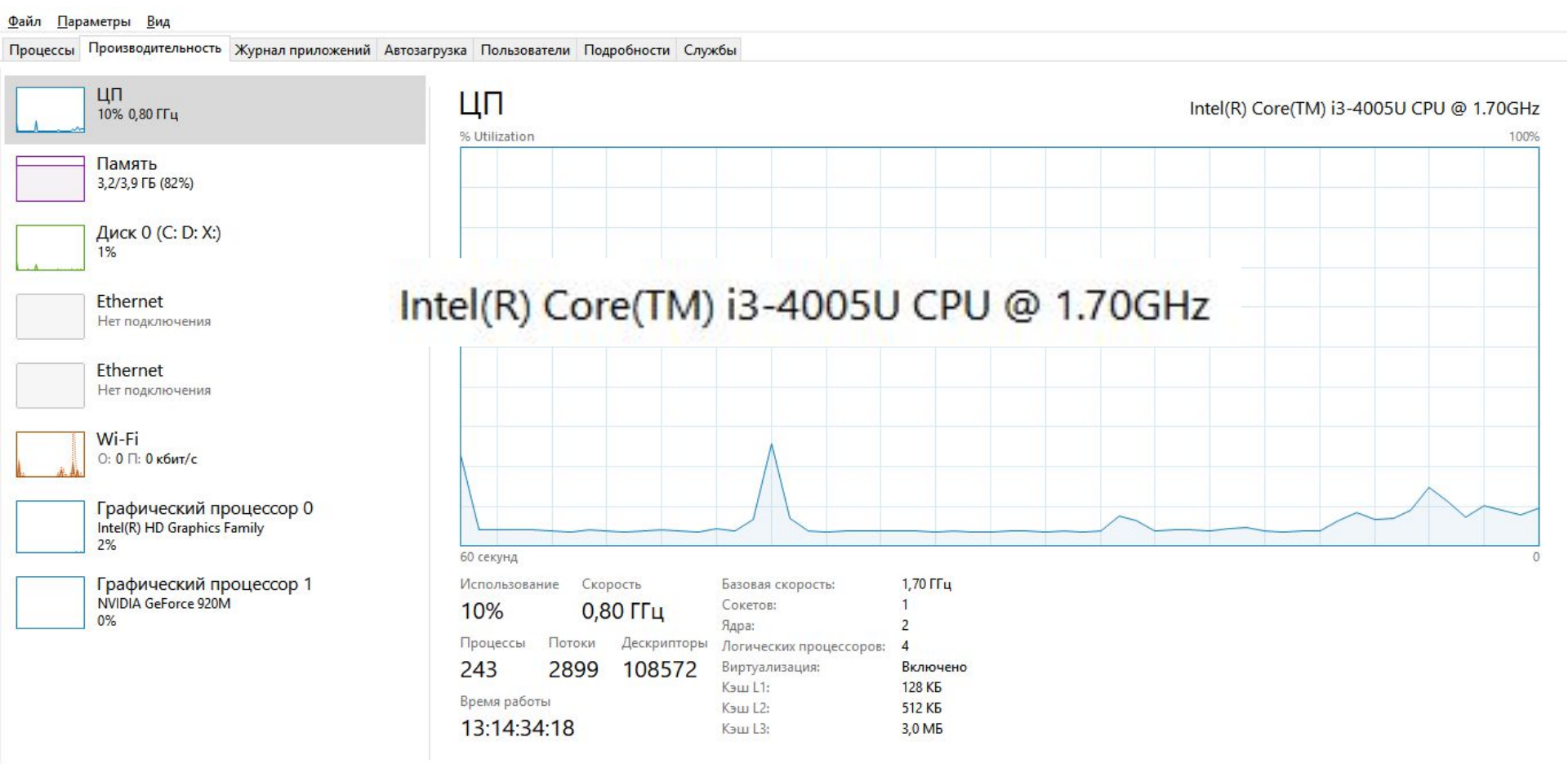
борются за один и тот же ресурс – ядро. Пример №2

Имя	78% ЦП	84% Память	1% Диск	0% Сеть	24% Графиче...	Ядро графического п
Приложения (9)						
Проводник (2)	1,4%	1,7%	0 МБ/с	0%	0%	
Ножницы	0,1%	0,1%	0 МБ/с	0%	0%	
Диспетчер задач	6,1%	1,6%	0,1 МБ/с	0%	0%	
Opera Internet Browser (21)	0,8%	11,3%	0,1 МБ/с	0,1%	0%	
Microsoft PowerPoint (32 бита)	0%	1,8%	0 МБ/с	0%	0%	
Microsoft Edge (13)	0%	1,5%	0 МБ/с	0%	0%	
Java(TM) Platform SE binary (3)	59,0%	35,1%	0,4 МБ/с	0%	0%	
Java(TM) Platform SE binary	57,4%	33,2%	0,4 МБ/с	0%	0%	
Java(TM) Platform SE binary	1,5%	1,6%	0 МБ/с	0%	0%	
Console Window Host	0%	0,3%	0 МБ/с	0%	0%	

Вот что происходит с процессором, когда оба потока запущены.



Вот что происходит с процессором, когда оба потока прекратили вычисления.



РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Время вычисления (сек)	Послед.	Паралл.	Кол-во итераций
	Поток main	Оба потока	
Тест 1	3 мин 35 сек	2 мин 53 сек	1000000
Тест 2	2 мин 32 сек	2 мин 9 сек	1000000

Мы видим, что при параллельном выполнении задачи мы улучшили показатели времени на 19,53% в первом тесте и на 15,13% во втором тестировании, при разбиении интервала AB на 1 000 000 отрезков, и вычисляя площадь на каждом отрезке

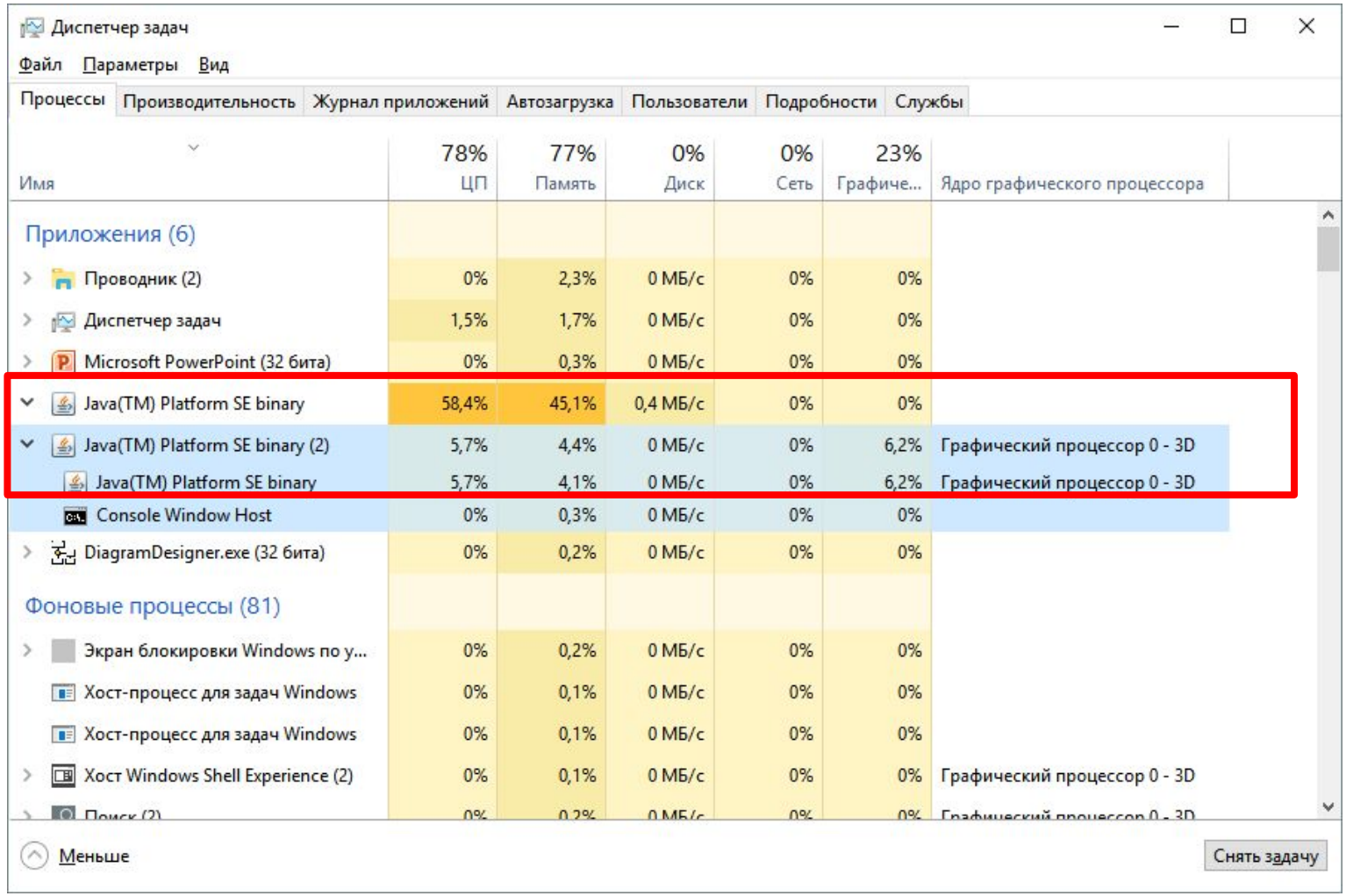
- ? ВОПРОС: ПОЧЕМУ ВРЕМЯ ВЫЧИСЛЕНИЯ ИНТЕГРАЛА НЕ УЛУЧШИЛОСЬ ПРИ ПАРАЛЛЕЛЬНОМ ПОДХОДЕ НА 50% , ВЕДЬ ПО ИДЕИ ЕСЛИ ДВЕ ПОДЗАДАЧИ ВЫПОЛНЯЮТСЯ ПАРАЛЛЕЛЬНО И НЕЗАВИСИМО ДРУГ ОТ ДРУГА, ТО ЗАДАЧА ДОЛЖНА РЕШИТЬСЯ В 2 РАЗА БЫСТРЕЕ?

- ОТВЕТ:
- В данном примере обе задачи были однотипные, а значит им требовались одни и те же АЛУ (вычисления с плавающей запятой). Если бы задачи были разного типа, например одна на целочисленные вычисления, другая на вычисления с плавающей запятой то, тогда им бы понабились разные АЛУ, и они бы не стояли в очереди к одному аппаратному ресурсу.

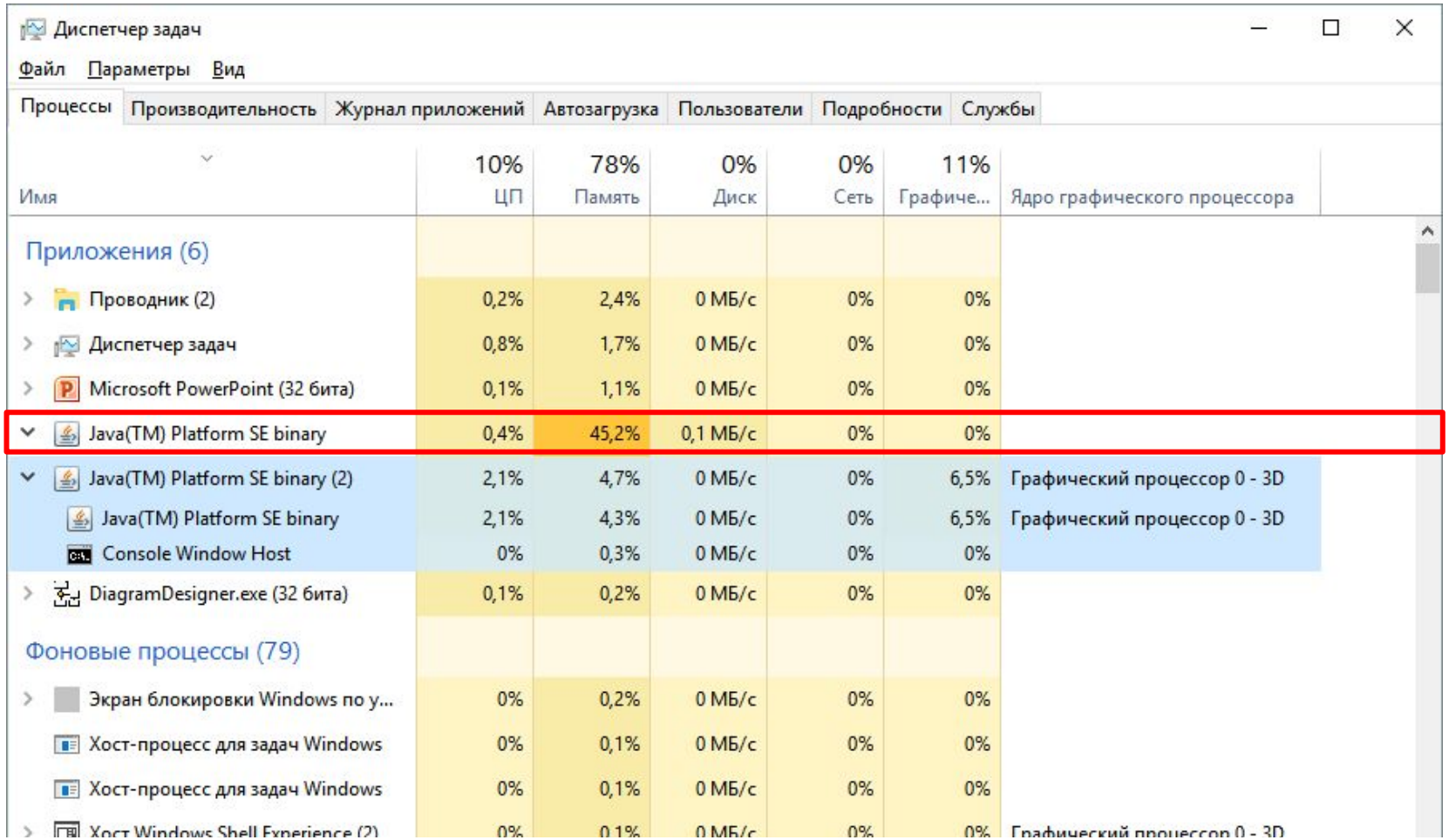
ПРОВЕДЕМ НОВЫЕ ИССЛЕДОВАНИЯ!

- Создадим два параллельных потока:
  - Первый поток будет вычислять интеграл для числа Pi (как в примере раньше)
  - Второй поток будет работать с графикой и ему понадобиться рисовать фигуру (треугольник) каждый раз на разном фоне (то на белом фоне, то на черном, в зависимости от переключения логической переменной)

КОГДА ОБА ПОТОКА РАБОТАЮТ



ПОТОК 1 ЗАВЕРШИЛ ВЫЧИСЛЕНИЯ



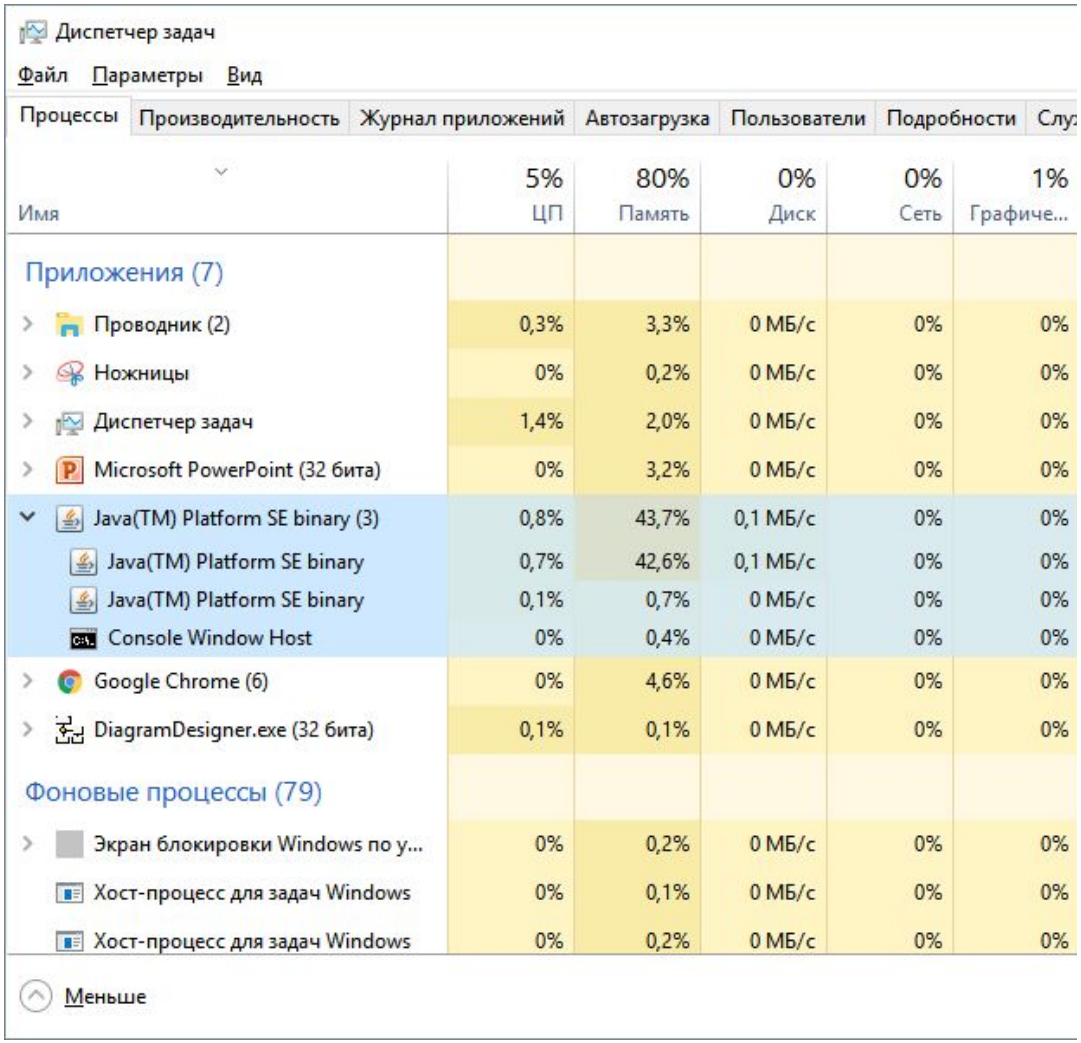
ОБРАТИМСЯ К ФИЛОСОФИИ ☺

- Ответит ли нам программа на вечный вопрос: Что появилось раньше: Яйцо или Курица?
- Один поток выводит на экран сообщение, что раньше появилось Яйцо, другой выводит ответ, что первее Курица.
- Программа будет работать в течении 5 секунд и каждый раз предугадать, какой ответ она выдаст трудно.
- Но мы попытаемся предугадать что ответит программа. Проведем 100 исследований и найдем вероятность того или иного ответа.

РЕЗУЛЬТАТЫ СПОРА:

```
run:
Спор начат..
Курица!
Яйцо!
Яйцо!
Курица!
Курица!
Яйцо!
Курица!
Яйцо!
Курица!
Яйцо!
Яйцо!
Спор закончен..
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 6 секунды)
```

НАГРУЗКА НА ЦП МИНИМАЛЬНА (ВЫЧИСЛЕНИЙ НЕТ)



ПРОВЕДЕМ 100 АНАЛОГИЧНЫХ ТЕСТОВ, ЧТОБЫ УЗНАТЬ ВЕРОЯТНОСТЬ ВЫДАЧИ ОТВЕТА

РЕЗУЛЬТАТЫ ТЕСТИРОАНИЯ:

ЯЙЦО vs КУРИЦА

ПОБЕЖДАЕТ ЯЙЦО СО СЧЕТОМ

88:12

?THE END!