# A Study of Data Parallelism Using a Java Based Framework-Aparapi

Submitted by

**Anindita Mondal**

Registration No. : 137325 of 2016-17

Examination Roll No. : MCA196015

Under the supervision of

**Dr. Chintan Kumar Mandal**

Assistant Professor, Dept. of Computer Science & Engineering

Jadavpur University

Kolkata-700032

A thesis submitted in partial fulfillment of the requirement for degree of Master of Computer Application in the department of Computer Science & Engineering

May 2019

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work as part of my MCA studies and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Name : **Anindita Mondal**

Registration No. : 137325 of 2016-17

Examination Roll No. : MCA196015

_____

**Anindita Mondal**

# Jadavpur University

## Department of Computer Science

## Jadavpur, Kolkata-700032

# Certificate

This is to certify that the work in this project report entitled as **"A Study of Data Parallelism Using a Java Based Framework-Aparapi" has** been satisfactorily completed by **Anindita Mondal.** It is a bona-fide piece of work for the fulfillment of the requirements for the degree of Master of Computer Application, of the Department of Computer Science & Engineering, Faculty of Engineering & Technology, Jadavpur University. This work is done during the academic year 2018-19 under my guidance.

_____

**Dr. Chintan Kumar Mandal**
*Assistant Professor*
*Department of Computer Science and Engineering*
*Jadavpur University*

_____

**Dr. Mahantapas Kundu**
*Head of the Department*
*Professor*
*Department of Computer Science and Engineering*
*Jadavpur University*

_____

**Prof. Chiranjib Bhattacharjee**
*Dean*
*Faculty of Engineering and Technology*
*Jadavpur University*

# Jadavpur University

**Department of Computer Science**

**Jadavpur, Kolkata-700032**

## Certificate of Approval

This is to certify that the project titled **"A Study of Data Parallelism Using a Java Based Framework-Aparapi "** is a bona-fide record of work carried out by Anindita Mondal in partial fulfillments for the award of the degree of Master of Computer Application, of the Department of Computer Science & Technology, Jadavpur University during the period January 2019 to May 2019. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve this thesis only for the purpose for which it is submitted.

_____ _____

Signature of the Internal Examiner Signature of the External Examiner

Date: _____ Date:_____

# Acknowledgement

With my sincere respect and gratitude, I would like to thank my advisor and project guide **Dr. Chintan Kumar Mandal** for his continuous support for this project work, for his patience, motivation and enthusiasm. His guidance helped me throughout the duration of the work. His valuable suggestions inspired me a lot. I feel deeply honored that I got the opportunity to work under his guidance. This accomplishment would not have been possible without him. Thank you.

Regards,

**Anindita Mondal**

Master of Computer Application

Jadavpur University

# ABSTRACT

Data parallelism is a very lucrative concept now-a-days. In different sector of working like finance, education, audit etc, we see extensive use of data processing. To make this data processing faster we need parallelism and as we know many of the time we need different set of data for same type of process then data parallelism will benefit us. To achieve this to make the process easy, less time consuming and efficient we take help of Aparapi in java framework. After achieving data parallelism, our hunger for more efficiency brings another mouth watering concept name process parallelism. We work hard on it and achieve some data about it and wish to come up with some more detail analysis of it on next time.

## TABLE OF CONTENTS

Page no

Chapter 3



Chapter 4



Chapter 5

# List of Figures

# CHAPTER 1

# 1. INTRODUCTION

Data is nothing but some raw materials. By processing it, we will get information. In present scenario, we always need to process a huge data set to collect information. Many of the time we see, process remains same, only the set of data changes. Compilation of huge amount of data is very time consuming process that reduce our efficiency. So we need an alternative way to increase efficiency by reducing time of processing. To achieve this, we use data parallelism concept here.

**Data parallelism** is parallelization of data across multiple processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel.

A data parallel job on an array of 'n' elements can be divided equally among all the processors. Let us assume we want to sum all the elements of the given array and the time for a single addition operation is 'Ta' time of units. In the case of sequential execution, the time taken by the process will be n*Ta time units as it sums up all the elements of an array. On the other hand, if we execute this job as a data parallel job on 4 processors the time taken would reduce to (n/4)*Ta + merging overhead time units. Parallel execution results in a speedup of 4 over sequential execution. One important thing to note is that the locality of data references plays an important part in evaluating the performance of a data parallel programming model. Locality of data depends on the memory accesses performed by the program as well as the size of the cache.

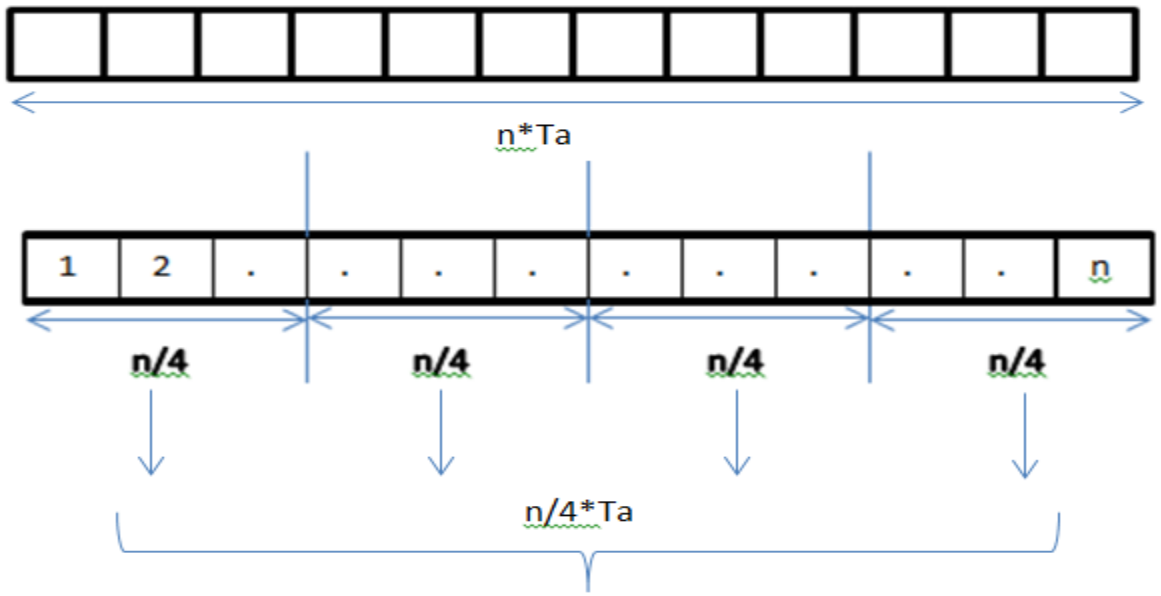


Fig 1. Data parallelism concept

Steps of parallelization,

- **Decomposition**:- The program is broken down into tasks, the smallest exploitable unit of concurrence.
- **Assignment**:- Tasks are assigned to processes.
- **Orchestration**:- Data access, communication, and synchronization of processes.
- **Mapping**:- Processes are bound to processors.

In our thesis, we implement this concept using kernel implementation using a java based framework called an Aparapi. Kernel will convert this code into OpenCL to make it run on **Graphics Processing Unit(GPU)**.

## 1.1 GRAPHICAL PROCESSING UNIT (GPU)

GPU computing is the use of a GPU (graphics processing unit) as a co-processor to accelerate CPUs for general-purpose scientific and engineering computing. The GPU accelerates applications running on the CPU by offloading some of the compute-intensive and time consuming portions of the code. The rest of the application still runs on the CPU. From a user's perspective, the application runs faster because it's using the massively parallel processing power of the GPU to boost performance. This is known as "heterogeneous" or "hybrid" computing.

A CPU consists of four to eight CPU cores, while the GPU consists of hundreds of smaller cores. Together, they operate to crunch through the data in the application. This massively parallel architecture is what gives the GPU its high compute performance. There are a number of GPU-accelerated applications that provide an easy way to access **high-performance computing (HPC).**

### 1.1.1 EARLY WORKS ON GPU

**Graphics processing units(GPU)** are devices present in most modern PCs. They provide a number of basic operations to the CPU, such as rendering an image in memory and then displaying that image onto the screen. A GPU will typically process a complex set of polygons, a map of scene to be rendered. It then applies textures to the polygons and then performs shading and lighting calculations.

One of the important steps was the development of programmable shaders. These were effectively little programs that the GPU ran to calculate different effects. No longer was the rendering fixed in the GPU through downloadable shaders, it could be manipulated. This was the first evolution of **general purpose graphical processor unit (GPGPU)** programming.

However, these shaders were operations that by their very nature took a set of 3D points that represented a polygon map. The **shaders** applied the same operation to many such datasets,in a hugely parallel manner, giving huge throughput of computing power.

## 1.2 OpenCL

**OpenCL (Open Computing Language)**is a framework or writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, Digital Signal Processors(DSPs) and other processors or hardware accelerators. **OpenCL** is an open standard that designed to utilize the computing power provided by GPUs for general computing applications. It is a **low**-level API that situates above GPU drivers and below applications. It is maintained by the **industrial** consortium Khronos who is also responsible for graphics API OpenGL.

In addition to its C-like programming language, OpenCL defines an application programming interface (API) that allows programs running on the host to launch kernels on the compute devices and manage device memory, which is (at least conceptually) separate from host memory. Programs in the OpenCL language are intended to be compiled at run-time, so that OpenCL-using applications are portable between implementations for various host devices.

### 1.2.1 WHY WE USE IT

OpenCL provides a standard interface for parallel computing using task- and data-based parallelism.Java code cannot directly run on GPU. So we will take help of OpenCL binding. Java code firstly convert into OpenCL code then it will execute directly on GPU. So OpenCL provides us environment on which we can run gpu programs directly.

# CHAPTER 2

## 2. APARAPI

**Aparapi** is an Open-source framework for expressing **data parallel** workloads in java. To perform this, developer extends a Kernel base class which compiles to java byte code using existing tool chain to debug the logic of their kernel implementation. It can perform this parallelism by executing the javacode in a java thread pool. But Aparapi allows java developers to take advantage of the computer power of GPU and APU devices by executing **data parallel code fragments** on the GPU rather than being confined to the local CPU. It does this by converting java bytecode to OpenCL dynamically at runtime and executing ion the GPU.

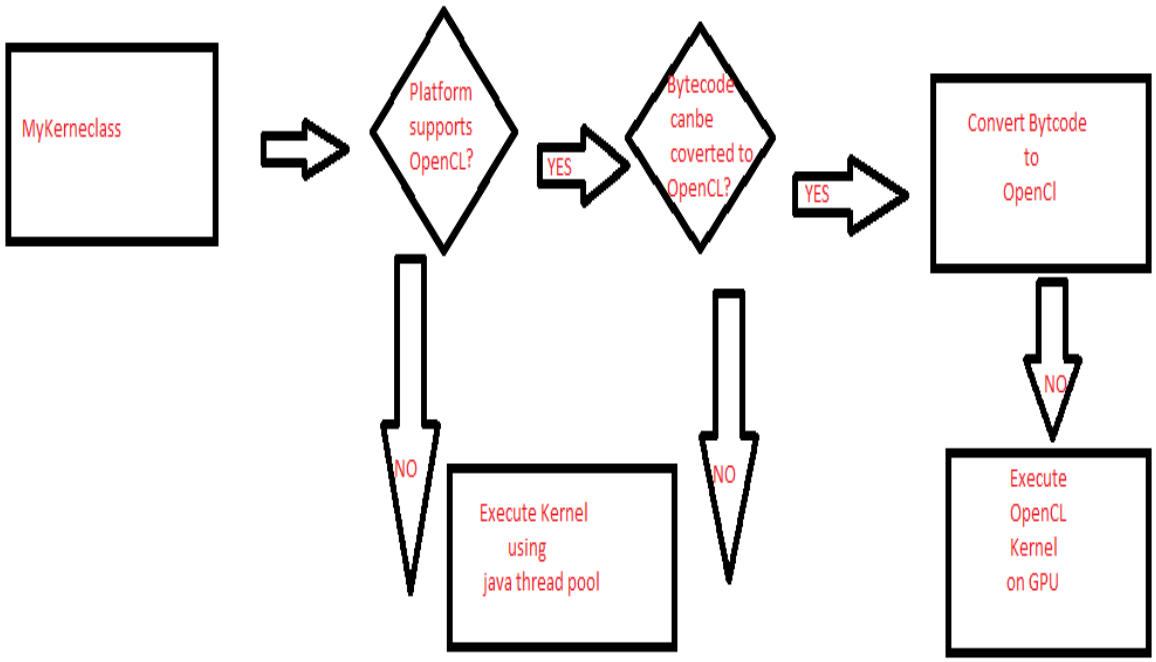


Fig 2. Flow diagram of kernel class processing

## 2.1 ANALYSIS OF APARAPI IMPLEMENTATION USING AN EXAMPLE

To view the implementation of aparapi we will an example of squaring each number of an integer array.

Firstly we will see the simple java code:

```
int[] square= new int[size];

int[] in = new int[size];

for(int i=0;i< size;i++){

        in[i] = (int) (math.random() * 1000);

}

for(int=0;i< size;i++){

        Square[i] = in[i] * in[i];

}
```

In this code, we take some random numbers and calculate their square values sequentially. But in parallel programming, the squaring of the numbers in the array will not execute sequentially. **It will execute in random order parallelly with each other.** That can be implemented by using aparapi by extending kernel class or by taking its object. Lets see how it is implemented:

```
Final int[] square= new int[size];

Final int[] in = new int[size];

for(int i=0;i< size;i++){

        in[i] = (int) (math.random() * 1000);

}

Kernel kernel = new Kernel();

@override

Public void run() {

        Int i= getGlobalID();
```

```
        Square[i]= in[i] * in[i];
}
Kernel.execute(size);
```

In this code we create an object of Kernel class and then get the random syntax of the element and take the square of it and store it in array. For this we need to import a package ***com.aparapi.Kernel***.

## 2.1.1 EXPRESSING DATA PARALLELISM IN APARAPI

Lets see what will happen when kernel.execute(size) is called:-

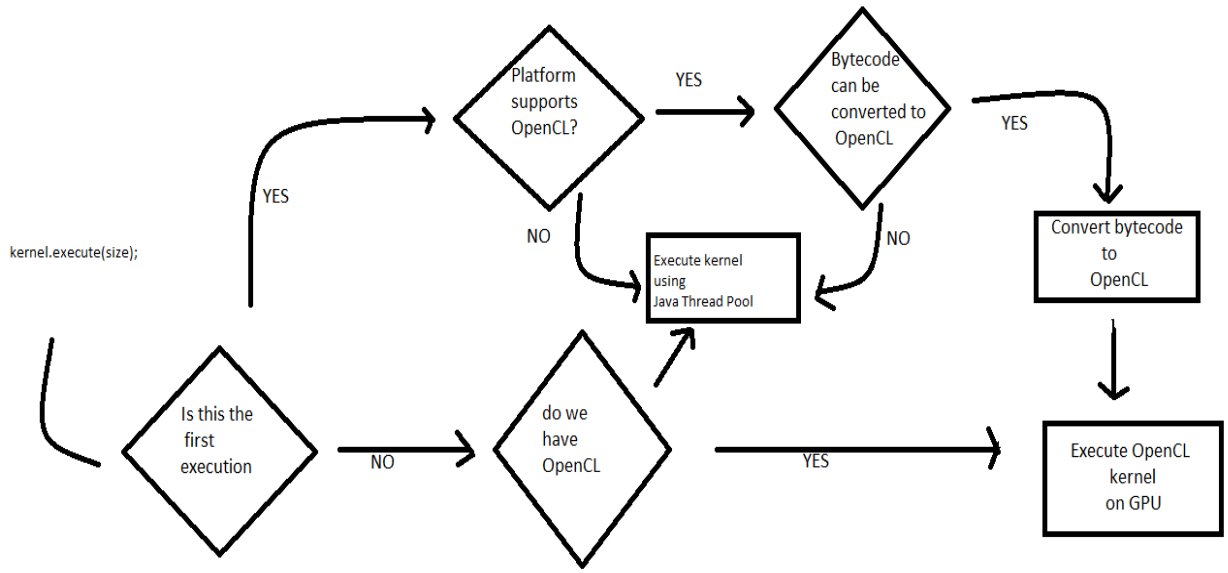


Fig 3. Flow diagram of execution of kernel.execute(size)

## 2.2 WHAT IS KERNEL?

The kernel is the central module of an operating system (OS). It is the part of the operating system that is loaded first, and it remains in main memory. As it stays in memory, it is important for the kernel to be as small as possible while still providing all the essential services required by other parts of the operating system and applications. The the kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system.

### 2.2.1 How kernel is implemented?

In computing, a **compute kernel** is a routine compiled for high throughput accelerators such as graphics processing units (GPUs), digital signal processors (DSPs) or field-programmable gate arrays (FPGAs) used by a main program (typically running on a central processing unit). They are sometimes called **compute shaders**, sharing execution units with vertex shaders and pixel shaders on GPUs, but are not limited to execution on one class of device, or graphics APIs.

Compute kernels roughly correspond to inner loops when implementing algorithms in traditional languages (except there is no implied sequential operation), or to code passed to internal iterators.

They may be specified by a separate programming language such as "OpenCL C" (managed by the OpenCL API), as "compute shaders" (managed by a graphics API such as OpenGL), or embedded directly in application code written in a high level language.

## 2.3 RANGE

It is a class of Aparapi api which defines the amount of parallelism will be occurred in a program. When kernel.execute() method is called range is passed as a rgument through it. suppose kernel.execute(50); which means range size is 50. That means 50 parallel execution will occur.

## 2.3.1 RANGE CLASS :

**public class Range extends RangeJNI**

A representation of 1, 2 or 3 dimensional range of execution. This class uses factory methods to allow one, two or three dimensional ranges to be created.
For a Kernel operating over the linear range 0..1024 without a specified groups size we would create a one dimensional Range using

**Range.create(1024);**

To request the same linear range but with a groupSize of 64 (range must be a multiple of group size!) we would use

**Range.create(1024,64);**

To request a two dimensional range over a grid (0..width)x(0..height) where width==512 and height=256 we would use

```
int width=512;
int height=256;
```

**Range.create2D(width,height);**

Again the above does not specify the group size. One will be chosen for you. If you want to specify the groupSize (say 16x8; 16 wide by 8 high) use

```
int width=512;
int height=256;
int groupWidth=16;
int groupHeight=8;
```

## 2.4 USE OF getGlobalID()

It is a method which returns a integer value. This value decides which index value data will be selected and executed. Suppose value comes 2 then the value present in index value 2 will be evaluated according to the program.

## 2.5 Code of calculation of square of random integer numbers stored in a array( repetition not allowed)

```
import com.aparapi.Kernel;
import com.aparapi.Range;

public class SquareKernel {
    public static void main(String[] _args) {
        final int[] values = new int[100];
         int num,sum=0;
         int j;

        // fill the values array
        for (int i = 0; i < values.length; i++) {
            int flag=0;
            num = (int) (Math.random() * 1000);
            for(j=0;j<=i;j++){
                if(values[j] == num ){
                    System.out.println("same number "+num+" occured");
                    flag=1;
                }
            }
            if(j>i && flag==0){
                values[i]=num;
            }
            else{
                i=i-1;
            }
        }

        for(int k=0;k<values.length;k++){
            sum = sum + values[k];
        }

        System.out.println("sum="+sum);
        final int[] squares = new int[values.length];
        final Range range = Range.create(values.length);

        Kernel kernel = new Kernel() {
            public void run() {
                int gid = getGlobalId();
                squares[gid] = values[gid] * values[gid];
            }
        };
        kernel.execute(range);
        for (int i = 0; i < values.length; i++) {
            System.out.printf("%4d %4d %8d\n", i, values[i], squares[i]);
        }
```

```
        kernel.dispose();
    }

}
```

## 2.5.1 Result

same number 495 occured
same number 506 occured
same number 299 occured
same number 342 occured
same number 309 occured
sum=54342

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 506 256036 | 16 514 264196 | 32 681 463761 | 48 680 462400 | 64 748 559504 | 80 993 986049 | 96 480 230400 | |
| 1 672 451584 | 17 663 439569 | 33 32 1024 | 49 880 774400 | 65 577 332929 | 81 804 646416 | 97 902 813604 | |
| 2 864 746496 | 18 215 46225 | 34 422 178084 | 50 596 355216 | 66 795 632025 | 82 820 672400 | 98 543 294849 | |
| 3 406 164836 | 19 252 63504 | 35 595 354025 | 51 247 61009 | 67 444 197136 | 83 863 744769 | 99 98 9604 | |
| 4 615 378225 | 20 968 937024 | 36 333 110889 | 52 299 89401 | 68 871 758641 | 84 128 16384 | | |
| 5 360 129600 | 21 627 393129 | 37 868 753424 | 53 145 21025 | 69 342 116964 | 85 474 224676 | | |
| 6 854 729316 | 22 790 624100 | 38 157 24649 | 54 709 502681 | 70 915 837225 | 86 154 23716 | | |
| 7 572 327184 | 23 119 14161 | 39 767 588289 | 55 631 398161 | 71 630 396900 | 87 483 233289 | | |
| 8 966 933156 | 24 845 714025 | 40 740 547600 | 56 305 93025 | 72 961 923521 | 88 62 3844 | | |
| 9 214 45796 | 25 437 190969 | 41 649 421201 | 57 494 244036 | 73 986 972196 | 89 807 651249 | | |
| 10 103 10609 | 26 343 117649 | 42 409 167281 | 58 210 44100 | 74 118 13924 | 90 385 148225 | | |
| 11 495 245025 | 27 475 225625 | 43 990 980100 | 59 278 77284 | 75 417 173889 | 91 632 399424 | | |
| 12 542 293764 | 28 45 2025 | 44 152 23104 | 60 369 136161 | 76 306 93636 | 92 692 478864 | | |
| 13 548 300304 | 29 309 95481 | 45 730 532900 | 61 686 470596 | 77 232 53824 | 93 821 674041 | | |
| 14 668 446224 | 30 566 320356 | 46 198 39204 | 62 167 27889 | 78 909 826281 | 94 960 921600 | | |
| 15 888 788544 | 31 665 442225 | 47 712 506944 | 63 815 664225 | 79 628 394384 | 95 310 96100 | | |

## 2.6 Prime number Checking parallely using aparapi

```
import com.aparapi.Kernel;
import java.util.Arrays;
import java.util.stream.IntStream;
public class GetPrime {
    public static void main(String[] args) {
        final int size = 25;
        final int[] a = IntStream.range(2, size + 2).toArray();
        final boolean[] primeNumbers = new boolean[size];

        Kernel kernel = new Kernel() {
            @Override
            public void run() {
                int gid = getGlobalId();
                int num = a[gid];
                System.out.println(num+"a["+gid+"]");
                boolean prime = true;
                for (int i = 2; i < num; i++) {
                    if (num % i == 0) {
                        prime = false;
                        //break is not supported
                    }
                }
                primeNumbers[gid] = prime;
            }
        };
        long startTime = System.currentTimeMillis();
        kernel.execute(size);
        System.out.printf("time taken: %s ms%n", System.currentTimeMillis() -
startTime);
        System.out.println(Arrays.toString(Arrays.copyOf(primeNumbers, size)));//just
print a sub array
        kernel.dispose();
    }
}
```

### 2.6.1 Result

a[0]=2 a[4]=6 a[8]=10 a[12]=14 a[16]=18 a[20]=22 a[2]=4 a[6]=8 a[10]=12 a[14]=16 a[18]=20 a[22]=24 a[3]=5 a[7]=9 a[11]=13 a[15]=17 a[19]=21 a[23]=25 a[1]=3 a[5]=7

a[9]=11 a[13]=15 a[17]=19 a[21]=23

time taken: 984 ms

| INDEX | VALUE | RESULT | INDEX | VALUE | RESULT | INDEX | VALUE | RESULT |
|---|---|---|---|---|---|---|---|---|
| 0 | 2 | TRUE | 10 | 12 | FALSE | 19 | 21 | FALSE |
| 4 | 6 | FALSE | 14 | 16 | FALSE | 23 | 25 | FALSE |
| 8 | 10 | FALSE | 18 | 20 | FALSE | 1 | 3 | TRUE |
| 12 | 14 | FALSE | 22 | 24 | FALSE | 5 | 7 | TRUE |
| 16 | 18 | FALSE | 3 | 5 | TRUE | 9 | 11 | TRUE |
| 20 | 22 | FALSE | 7 | 9 | FALSE | 13 | 15 | FALSE |
| 2 | 4 | FALSE | 11 | 13 | TRUE | 17 | 19 | TRUE |
| 6 | 8 | FALSE | 15 | 17 | TRUE | 21 | 23 | TRUE |

## 2.7 RESTRICTION ON kernel.run() CODE

- Only the Java primitive data types boolean, byte, short, int, long, and float and one-dimensional arrays of these primitive data types are supported by Aparapi.
- Aparapi support for the primitive data type double will depend on your graphics card, driver, and OpenCL version. Aparapi will query the device/platform to determine if double is supported (at runtime). If your platform does not support double, Aparapi will drop back to Java Thread Pool (JTP) mode.

- The primitive data type char is not supported.
- Elements of primitive array fields can be read from kernel code.
- Elements of primitive array fields can be written to by kernel code.
- Java creates 'hidden' fields for captured final primitive arrays (from anonymous inner classes) and they can be accessed as if they were fields of the kernel.

- Primitive scalar fields can only be read by the kernel code. Because kernel run-reachable methods execute in parallel in an indeterminate order, any reliance on the result of modifications to primitive scalar fields is discouraged even when executing in Java Thread Pool mode.
- Static final fields can be read from kernel code.
- Static non-final fields are not supported for either read or write.
- Arrays cannot be aliased either by direct local assignment or by passed arguments to other methods.
- References to or through a Java Object other than your kernel instance will cause Aparapi to abandon attempting to create OpenCL

- Static methods are not supported by Aparapi.
- Recursion is not supported, whether direct or indirect.

- Methods with varargs argument lists are not supported by Aparapi.
- Overloaded methods (i.e. methods with the same name but different signatures) are not supported by Aparapi.

- Exceptions are not supported (no throw, catch. or finally).
- New is not supported either for arrays or objects
- Synchronized blocks and synchronized methods are not supported.
- Only simple loops and conditionals are supported; switch, break, and continue are not supported.
- A variable cannot have its first assignment be the side effect of an expression evaluation or a method call.

## 2.8 HOW TO CONVERT A TWO DIMENSIONAL ARRAY INTO ONE DIMENSIONAL ARRAY

Lets take a two dimensional array:-

```
Int arr[5][5];

For(int i=0;i<5;i++){

        For(j=0;j<5;j++){

                Arr[i][j] = (int)(math.random()*100);

        }

}
```

Now we will implement it into one dimensional array:-

```
Int arr[5*5];

For(int i=0;i<5;i++){

        For(j=0;j<5;j++){

                Arr[(i*5) + j] = (int)(math.random()*100);

        }

}
```

### 2.8.1 Matrix multiplication program using aparapi

```
import java.util.Random;
import com.aparapi.Kernel;

public class MatrixMul {


        public static void main(String [] args)
        {

            final int r = 1024;
            final int c1 = r;
            final int c2 = r;
            AparapiMatMul ap = new AparapiMatMul(r, c1, c2);

            try {
                long  time1 = System.currentTimeMillis();
                //ap.setExecutionMode(Kernel.EXECUTION_MODE.JTP);
```

```
                //ap.setExecutionMode(Kernel.EXECUTION_MODE.GPU);
                //ap.setExecutionMode(Kernel.EXECUTION_MODE.CPU);
                ap.execute(r,c2);
                System.out.println("Time taken for kenel execution in  GPU mode is :"+
(System.currentTimeMillis() - time1));
            }catch(NullPointerException ne){
                ne.printStackTrace();
            }
            //ap.printResults();
            long time1 = System.currentTimeMillis();
            ap.normalMatMulCalc();
            System.out.println("Time taken for kenel execution in Sequential CPU mode
is :"+ (System.currentTimeMillis() - time1));
            ap.printResults();
            ap.compareResults();
            ap.dispose();
        }
    }

    class AparapiMatMul extends Kernel {

        float matA[];
        float matB[];
        float matC[];
        float C[];

        int rows ;
        int cols1;
        int cols2;

        @Override
        public void run() {
            int i = getGlobalId();
            int j = getPassId();
            float value = 0;
            for(int k = 0; k < cols1; k++)
            {
                value += matA[k + i * cols1] * matB[k * cols2 + j];
            }
            matC[i * cols1 + j] = value;
        }

        public AparapiMatMul(int r, int c1, int c2)
        {

            rows = r;
            cols1 = c1;
            cols2 = c2;

            matA = new float [r * c1];
            matB = new float [c1 * c2];
            matC = new float [r * c2];
            C = new float[r * c2];
            //matC should be initialized with zeros
            for(int i = 0; i < r; i++ )
            {
                for(int j = 0 ; j < c1; j++ )
                {
                    matC[i * c1 + j ] = 0;

                }
            }
```

```
            //Here matrix A is initialized with random numbers

            for(int i = 0; i < r; i++ )
            {
                for(int j = 0 ; j < c1; j++ )
                {
                    matA[i * c1 +j] = new Random().nextFloat();
                }
            }

            // Here matrix B is initialized with random numbers

            for(int i = 0; i < r; i++ )
            {
                for(int j = 0 ; j < c1; j++ )
                {
                    matB[i * c2 + j] = new Random().nextFloat();
                }
            }

        }

        public void printResults()
        {
            for(int i = 0; i < rows; i++ )
            {
                for(int j = 0 ; j < cols2; j++ )
                {
                    System.out.print(matC[i * cols2 + j]+"    ");
                }
                System.out.print("  ");
            }
        }

        public void normalMatMulCalc()
        {
            System.out.println();
            System.out.println("Sequential Execution on CPU");
            for(int i = 0;i < rows; i++)
            {
                for(int j = 0; j < cols2; j++)
                {
                    float sum = 0;
                    for(int k = 0; k < cols1; k++)
                    {
                        sum += matA[i*cols1+k] * matB[k*rows+j];
                    }
                    C[i * cols2 + j] = sum;
                }

            }
        }
        public void compareResults()
        {
            boolean equal = true;
            for(int i = 0; i < rows * cols2 ; i++)
            {
                if(matC[i] != C[i])
                {
                    equal = false;
                    break;
                }
            }
```

```
            if(!equal)
                System.out.println("Results are not equal");
            else
                System.out.println("Results are equal.. Tested thoroughly!!!");
        }

    }
```

# CHAPTER 3

# 3. APPLICATION OF PARALLELISM ON BUG'S ALGORITHM

## 3.1 INTRODUCTION

Robots are basically a mechanical device which performs automated tasks, either according to human supervision, a predefined program or a set of general guidelines using artificial intelligence techniques. Nowadays, industrial robots are intensively used in wide variety of applications. Most of the industrial robots are motionless. They operate from a fixed position and have limited operating range. These robots efficiently complete tasks such as welding, drilling, assembling, painting and packaging.

Mobile robots in general are those robots which can move from place to place across the ground. Mobility gives a robot a much greater flexibility to perform new, complex, exciting tasks. Navigation of autonomous mobile robot is a very challenging problem. There exist many algorithms for navigation of autonomous mobile robot. One of the most popular one is Intelligent Bug Algorithm (IBA). There are many versions of IBA which have already been proposed previously.

### 3.1.1 WHY WE NEED IT?

In a multi robotic environment, when a robot get an instruction to perform required task. It start moving towards the destination. When it moves it try to avoid collision with other robots and obstacles comes in front of its. To solve this problem there is different type of bug algorithm available. We will discuss them step by step.

## 3.2 BUG ALGORITHM 1

**Algorithm:**

**Input:** A point robot with a tactile sensor

**Output:** A path to the Destination or a conclusion no such path exists

*while* Forever *do*
    *repeat*
        From Start, move toward Destination.
    *until* Destination is reached or an obstacle is encountered.
    *if* Goal is reached *then*
        Exit.
    *end if*
    *repeat*
        Follow the obstacle boundary.
    *until* Destination is reached or an obstacle is re-encountered.
    Determine the leaving point on the perimeter that has the shortest distance to the Destination.

Go to the Leaving point.
*if* the robot were to move toward the Destination *then*
Conclude Destination is not reachable and exit.
*end if*
*end while*

The Bug1 algorithm was the first algorithm in the bug family [1, 11, 17] created by Lumelsky and Stepanov. In this algorithm the robot follows a straight line towards the destination. When it encounters an obstacle (Fig. 4), it traverses the boundary of the obstacle. Simultaneously the robot calculates the distance from current position to destination until it reaches the point from where it started traversing the obstacle (Fig.5). Whichever position has the minimum distance from the destination becomes the *leaving point* and the robot again traverses the obstacle and comes to the leaving point. After reaching the leaving point it calculates a straight line toward the destination (Fig. 6)). Then it starts following the line until another obstacle is re-encountered or it reaches the destination.

### 3.2.1 ADVANTAGES

- This algorithm doenot suffer local minima problem.

### 3.2.2 DISADVANTAGES

- When the robot is following the boundary of obstacle 1, it may collide with a nearby obstacle 2 where the gap between both the obstacles is less than the width of the robot; as a result the robot moves far away from the destination.
- The distance covered by the robot to avoid the obstacle is much more than the perimeter of the obstacle.

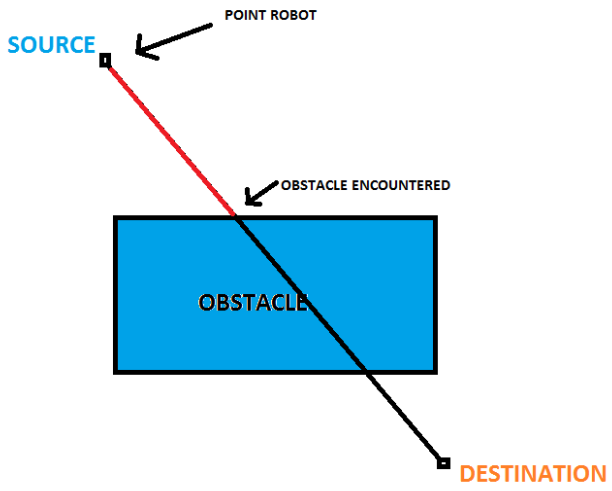


Fig. 4: Obstacle Detected

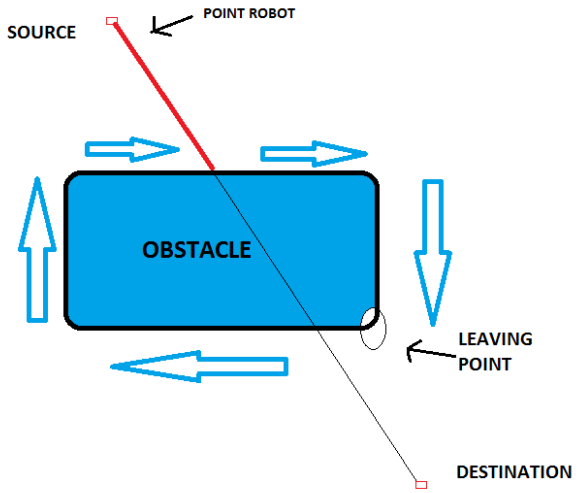


Fig. 5: Robot Traverses Back to the Starting Position and Calculates Leaving Point

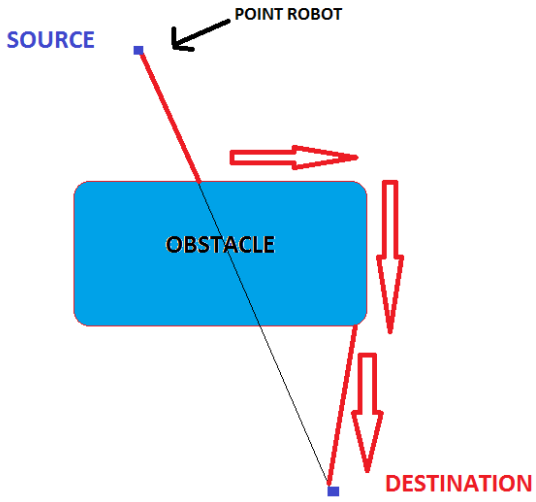


Fig. 6: Robot Traverses to Leaving Point then Moves to Destination

## 3.3 BUG ALGORITHM 2

### Algorithm:

**Input:** A point robot with a tactile sensor
**Output:** A path to Destination or a conclusion no such path exists

*while* True do
    *repeat*
        From Start, move toward Destination along slope (m-line).
    *until* Destination is reached or an obstacle is encountered at hit point.
    Turn left (or right).
    *repeat*
        Follow boundary
    *until* Destination is reached or hit point is re-encountered or mline is re-encountered at a point m such that m ≠ hit point (robot did not reach the hit point), d(m, Destination) < d(m, hit point ) (robot is closer), and if robot moves toward goal, it would not hit the obstacle
    *if* Goal is reached *then*
        Exit.
    *end if*
    *if* hit point is re-encountered *then*
        Conclude Destination is unreachable
    *end if*
    Let Start(i+1) = m
    Increment i
*end while*

The Bug2 algorithm was also created by Lumelsky and Stepanov [1, 11, 17]. In this algorithm the robot follows single non-repeated path throughout its trajectory. When the source and destination are set, the robot calculates the slope. It follows that slope and moves forward until encounters an obstacle (Fig. 3(a)). The robot then changes its behavior from 'move to goal' to 'obstacle avoidance'. While traversing the boundary of the obstacle, it continuously calculates the slope from its current position to the destination. When it reaches a point which has the same slope as of the original straight line (Fig. 3(b)), the robot again changes its behavior from 'obstacle avoidance' to 'move to goal' and follows the slope until it reaches another obstacle or destination (Fig. 3(c)). It is more efficient than Bug 1 algorithm as it allows the robot to reach the destination in less time following a short trajectory.

### 3.3.1 ADVANTAGES

- The robot follows single non-repeated path throughout its trajectory
- It is more efficient than Bug 1 algorithm as it allows the robot to reach the destination in less time following a short trajectory.

### 3.3.2 DISADVANTAGES

- Do not have capability to make optimum use of sensors data for

generation of shorter paths.

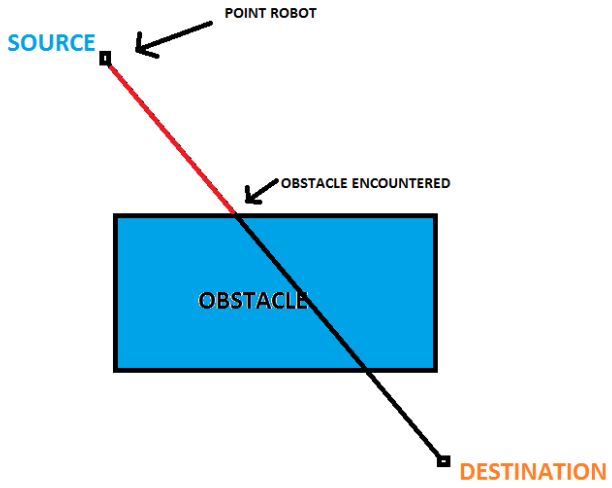


Fig. 7: Obstacle Detected

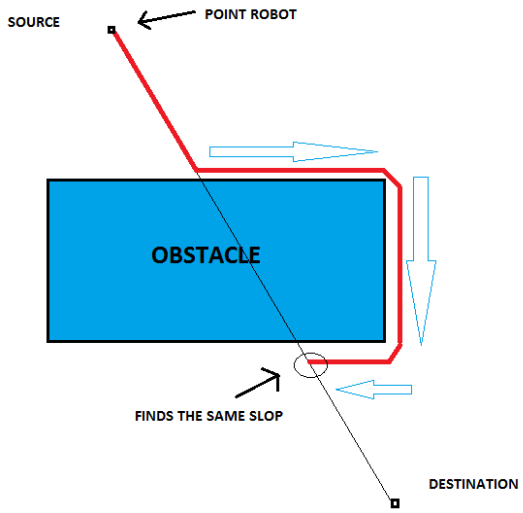


Fig. 8: Robot Traverses the Obstacle until it finds a Point with Same Slope

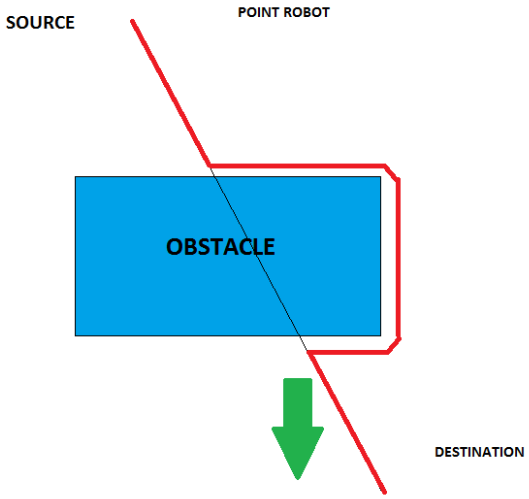


Fig. 9: Robot Traverses to the Destination

## 3.4 OUR GOAL

We all know a robot can do many works now-a-days. But when a robot move from one place to other its job become more versatile and it becomes more important to the world. When navigation comes first thing comes in our mind is how to avoid collision in the path of robot. To solve this problem we will take help of bug's algorithm. But our problem is not finished still. **Our motive is to create a navigation system for multiple robots where they can navigate from one place to other not only without colliding but also independently at the same time.**

### 3.4.1 OUR IMPLEMENTATION

We take javaFx environment for graphical view of this problem. We try to solve this problem for two robots. We used bug's algorithm for obstacle avoidance and for line drawing we used Bresenham line drawing algorithm. Here we take help of Aparapi for kernel implementation and give OpenCL binding and parallel environment to the whole procedure. Where two robot will act simultaneously and we will view it in javaFx.

### 3.4.2 OUR OBSERVATION

There are mainly two important observation we found. They are:-

One,When we use kernel some restriction comes automatically with it. We overcome some of them. We convert all the two dimensional code into one dimensional system. We worked with only primitive data types We follow all the restrictions very carefully but we know kernel does not allow any type of exception. But if we want to implement GUI portion using JavaFx we need to handle javaFX application Thread. As we can not handle any type of exception inside run so we can not implement it with JavaFX.
Another, process parallelism is not possible using Aparapi framework.

# CHAPTER 4

## 4. CONCLUSION AND OPEN CHALLENGE

Towards our goal, we found data parallelism which provide us efficient way of huge data processing. But same way, we come to know that process parallelism is not possible. Also we see JavaFx cannot run using aparapi framework. Viewing all types of scenario I propose a model which may solve our problems and achieve the goal.

I propose a server client model for this problem where in server we will perform parallelism and calculation and in client we will show the result. But for this we will need to search for a framework which will provide process parallelism.

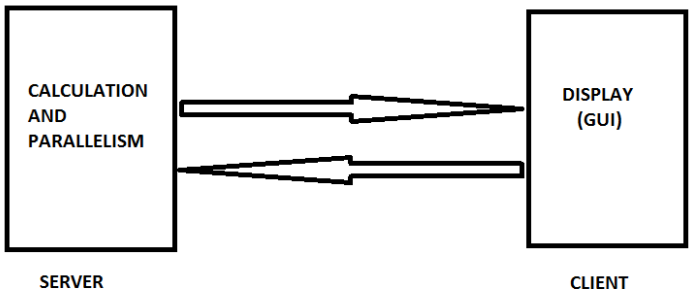


FIG 10. Proposed Model

# CHAPTER 5

## 5. REFERENCES


- https://github.com/Syncleus/aparapi-examples
- https://drive.google.com/drive/folders/1ZmT4cWn7YR5_8JnVbK3d6B3Uq1G8b3Mb
- CUDA Programming by Shane Cook
- https://vasanthexperiments.wordpress.com/2011/11/20/aparapi-java-matrix-multiplication-example/
- https://www.logicbig.com/tutorials/misc/gpu-programming/aparapi/intro-with-example.html
- http://aparapi.com/
- http://www.sciencedirect.com > topics