B. TECH. PROJECT REPORT

On

Integrated Scheduling and Allocation for Simultaneous Exploration of Unrolling Factor and Datapath during Power-Delay Trade-off

BY

Juhi Naik & Pavan Tarigopula

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2014

Integrated Scheduling and Allocation for Simultaneous Exploration of Unrolling Factor and Datapath during Power-Delay Trade-off

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Juhi Naik and Pavan Tarigopula

Guided by:
Dr. Anirban Sengupta, Assistant Professor, IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2014

CANDIDATE'S DECLARATION

We hereby declare that the project entitled **"Integrated Scheduling and Allocation for Simultaneous Exploration of Unrolling Factor and Datapath during Power-Delay Trade-off"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of Dr. Anirban Sengupta, Assistant Professor, Computer Science and Engineering, IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student(s) with date

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

Signature of BTP Guide(s) with dates and their designation

Preface

This report on “Integrated Scheduling and Allocation for Simultaneous Exploration of Unrolling Factor and Datapath during Power-Delay Trade-off” is prepared under the guidance of Dr. Anirban Sengupta.

Through this report we have tried to give a detailed explanation of the design and working of the novel approach devised by us for integrated scheduling and allocation while simultaneously exploring unroll factor and datapath. We hope this will be useful in making the process of High Level Synthesis (HLS), used in the building of various digital systems, more efficient.

We have tried to the best of our abilities and knowledge to explain the content in a lucid manner. We have also added figures and comparative tables to make it more illustrative and provided pseudo-codes of algorithms, wherever necessary.

Juhi Naik(1100115) & **Pavan Tarigopula(1100134)**
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Acknowledgements

I wish to express my gratitude to Dr. Anirban Sengupta, Assistant Professor, IIT Indore for his kind support and valuable guidance. I wish to thank Ms. Saumya Bhadauria and Mr. Vipul Kumar Mishra, Ph. D. Scholars, Computer Science and Engineering, IIT Indore for their guidance all throughout the project. Also, I wish to thank my project partner, Mr. Pavan Tarigopula, B. Tech IV Year, Computer Science and Engineering, IIT Indore for his support throughout the project.

It is their help and support, due to which we became able to complete the project and the technical report.

Without their support this project as well as report would not have been possible.

Juhi Naik(1100115) & **Pavan Tarigopula(1100134)**
B. Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Abstract

This report presents a novel integrated design space exploration (DSE) methodology driven by bacterial foraging optimization algorithm(BFOA) that is equipped with the capability to explore an optimal combination of datapath resource configuration and loop unrolling factor (UF) for optimal scheduling for control data flow graphs (CDFGs). The proposed approach also comprises of novel bacterium encoding strategy for both 'datapath bacterium' and 'auxiliary bacterium'. The approach presented supports operation chaining in scheduling during delay evaluation as well as considers the impact of loop unrolling on the configuration of multiplexer size during power evaluation. The above all have not jointly been addressed in literature so far. Finally, the results of proposed approach on standard benchmarks yielded significantly improved quality of result (cost minimization) and reduced exploration runtime compared to recent genetic algorithm (GA) driven DSE approaches.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivation

Design Space Exploration (DSE) refers to the process of designing a data flow or schedule by searching for a solution which is optimal in terms of certain objectives (e.g. Power, latency etc.) while constrained by certain factors (e.g. No. of resources available). It involves trying out different alternate design parameters to "explore the design space" [12]. Exhaustively evaluating each and every design point, however, is prohibitive for the large design spaces that are usually found in general applications. Loop Unrolling is a way of achieving Instruction Level Parallelism so as to decrease the latency at the cost of increased number of resources as well as power [13]. Deciding when to apply loop unrolling and by what factor itself leads to widening the design space to a large extent, thus increasing the complexity of the problem manifold. To overcome the problem of DSE combined with automated loop unrolling, we incorporate knowledge of the design space into the search strategy by using the evolutionary mechanisms of Bacterial Foraging Optimization (BFO) i.e. chemo taxis, swarming, reproduction, and elimination-dispersal [6] tailored to the needs of our problem. As mentioned earlier, an exhaustive exploration of the design space is impractical due to the sheer size of the design space. Thus, to get an optimal design point, we need to use sophisticated AI methods and heuristics as well as calculated trade-offs. Such trade-off decisions include not only the balance between the various variables like power, latency, loop unrolling factor etc. but also deciding the precision required for the design point versus the time required to achieve that precision. Thus, active work is going on to implement faster and more intelligent search algorithms.

Unrolling of a loop mainly consists of replications of its body corresponding to consecutive iterations. This not only increases the level of parallelism but also, when unrolled fully, can enable the reuse of data across the newly exposed inner loop leading to a substantial reduction of the number of memory access if the reused data is kept in registers [13]. In this project we aim to apply one of the newer AI algorithms, BFO, to create a tool for High level Synthesis (HLS) for traditional DSE combined with the added factor - loop unrolling.

1.2 Preamble

High level synthesis (HLS) involves transition from an algorithmic level description of an application in CDFG to its equivalent register transfer level (RTL) counterpart which implements the specified behavior as well as satisfies the conflicting user constraints such as power and delay. The above process includes exploration which complicates with the inclusion of an ancillary variable called 'loop unrolling factor' during exploration of an optimal scheduling during HLS. Such an intricate and intractable problem requires administration of advanced algorithms equipped with adaptive capabilities to change direction when a certain search path is found unrewarding. BFO mechanism is known to have the aforesaid capabilities to search an optimal solution which motivated us to adopt its framework as our exploration backbone [1, 6].

Problem description

The DSE problem with HLS can be stated as follows :

Given a high-level design specification with a budget of i iterations, and HLS tool, find the best approximate Pareto- optimal solution of RTL designs without exceeding i.

The following figure depicts the input and output flow for the HLS tool.

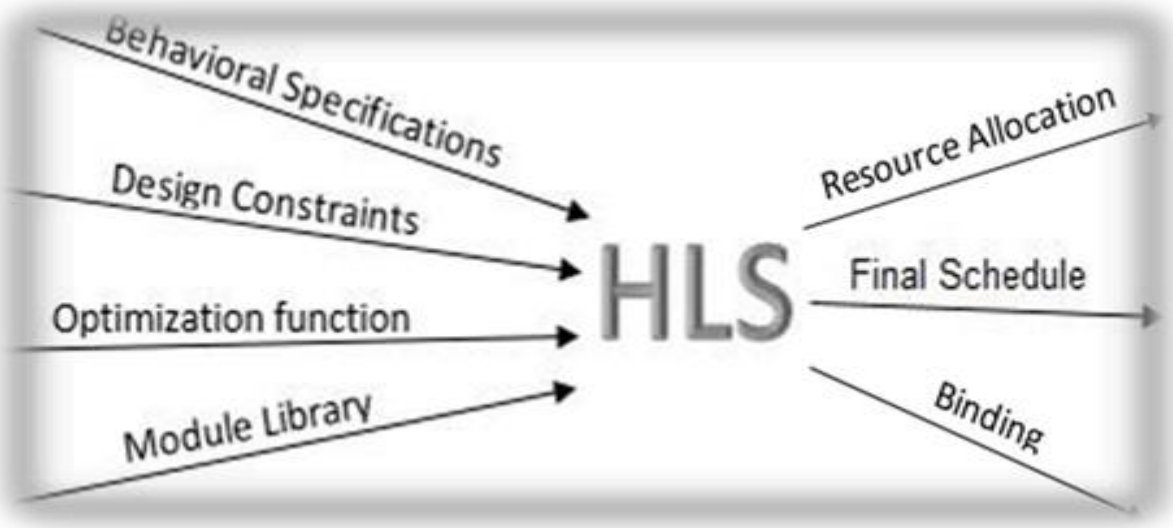


Figure 1 Design Flow of HLS

Explanation of the inputs and outputs

Behavioural Specifications – The specifications will be given as a Control Flow Graph (CFG) in .txt format giving the behaviour and functionality of the application required. In a CFG representation the interconnected vertices are called basic blocks (BB). Each basic block contains a sequence of data operations ended by a control flow statement as its last instruction. This means that the operations inside a BB are completely data oriented and can be represented as data flow graph (DFG) [14].

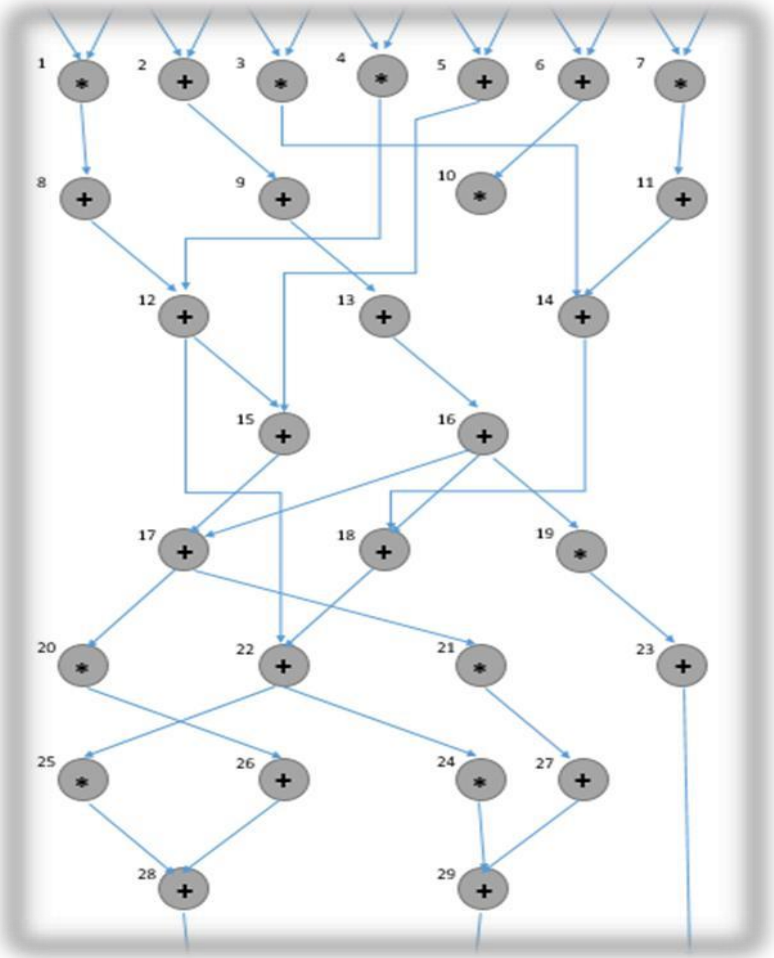


Figure 2 Data Flow Graph of an application

```
*,I,I,1,+,I,I,2,*,I,I,3,*,I,I,4,+,I,I,5,+,I,I,6,*,I,I,7
+,1,1,8,+,2,2,9,*,6,6,10,+,7,7,11
+,4,8,12,+,9,9,13,+,11,3,14
+,5,12,15,+,13,13,16
+,15,16,17,+,16,14,18,*,16,16,19
*,17,17,20,*,17,17,21,+,18,10,22,+,19,19,23
*,22,22,24,*,22,22,25,+,20,20,26,+,21,21,27
+,25,26,28,+,24,27,29
```

Figure 3 Input to the application in the form of a .txt file

Design Constraints – There are two types of design constraints applicable,

1) The implicit ones that we can get from the Behavioural Specs and Module Library like – Precedence Relations of operations, Number of available hardware resource units available as well as the power and time taken for one instruction on each type of hardware etc.;

```
adder(16),0.739pj,2030au,270ns,3
subtractor(16),0.739pj,2030au,270ns,3
multiplier(16),9.8pj,2464au,11000ns,8
comparator(16),0.739pj,2030au,270ns,2
mux2:1,0.1pj,126au,20ns,0
```

Figure 4 Example of a Module Library

Optimization Function – The optimization function, BFO, will be a part of the actual coding of the tool itself. However, certain factors (e.g. Minimum/Maximum number of iterations to be done) will be taken as input through the GUI.

Final Schedule – The HLS tool, after applying DSE using the above parameters, allocating resources and binding modules to the operations, will give a Pareto-optimal design point in terms of its schedule, i.e. the operations to be done in each clock cycle. This schedule can be represented in .txt format too.

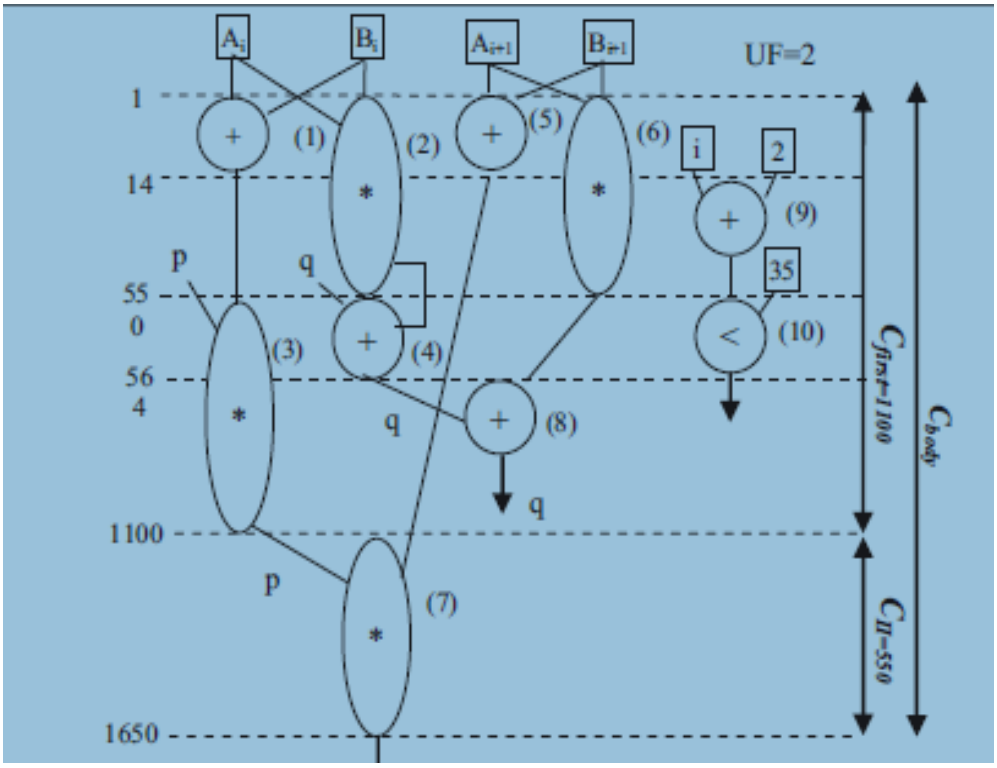


Figure 5 Example of a schedule: Automated exploration of datapath and unrolling factor during power–performance tradeoff in architectural synthesis using multi-dimensional PSO algorithm, Anirban Sengupta, Vipul Kumar Mishra

Loop Unrolling

Loop unrolling is a type of Instruction level parallelism where a set of consecutive iterations in the loop are unrolled and performed parallelly. After unrolling by a factor of U, the cycle count (latency) of the block reduces by U while the gate count (and hence power) increases by U (approx.)

```
for(i=0; i<36; i++) {
P=P * (A[i] + B[i]);
q=q + (A[i] * B[i]);
 }
```

C code before un-rolling

```
for(i=0; i<36; i+=2) {
P=P * (A[i] + B[i]); P=P * (A[i+1] + B[i+1]);
q=q + (A[i] * B[i]); q=q + (A[i+1] * B[i+1]);
}
```

C code after unrolling with an unroll factor of 2

There are a few things however that need to be kept in mind while performing unrolling:

1) The unrolling factor has to be a divisor of the iteration count of loop.

2) Actual unrolling is too expensive to perform for analysis purposes given the extreme complexities of the applications on which the algorithm will potentially be applied. Hence, a method needs to be devised to calculate the optimal U without performing any actual unrolling.

3) The advantages or returns due to unrolling start diminishing after sometime. Hence, we need to find this balanced design point.

Constraints for Loop Unrolling

Taking care of tailing iterations in case of an unroll factor which does not perfectly divide the iteration count.

Discarding unfeasible unroll factors to save computation time.

Decreasing/Avoiding repetitive unrolling for the same unroll factors but different resource constraints.

Mathematical Aspect

Since the problem is a multi-objective optimization problem, it has three main parts:

Variables - Combination of which forms the basic optimization function

Constraints - To be satisfied by the variables

Optimization function – A combination of the multiple objectives to be achieved with an appropriate weight for each.

Variables

Array of number of resources of each type to be used in schedule i.e. [R1, R2, …. , Rn] where Ri is one of the resource types given in the module library. Array of unroll factors in each direction (in case of nested loops) for eg. In case of the following loop:

```
for(i = 0; i<10; i++) {
        for(j=0;j<15;j++) {
        z = z * (i+j);
        }
}
```

The array of unroll factors could be [2,3] i.e. 2 in i direction and 3 in j direction.

1.3 Objectives

1. Represent the DSE with loop unrolling problem as a mathematical multi-objective optimization problem with respective constraints.
2. Map the major mechanisms of Bacterial Foraging optimization to the problem in hand.
3. Devise a way to analyze the impact of unrolling factor and find the optimal factor in loop unrolling without performing the actual unrolling.
4. Decide number of iterations (generations) required in BFO and produce a complete algorithm for the problem.
5. Implement the algorithm programmatically i.e. write an HLS tool (probably using Java).
6. Test and record results on various benchmarks and compare with other algorithms.

Chapter 2: Literature Survey

There has been no previous work that addresses automated integrated exploration of datapath resource configuration and loop unrolling factor for optimal operation chaining based scheduling during HLS. For example, in [1], though authors handled CDFGs, they did not handle exploration of an optimal scheduling using any priority function resolver. Further, GA based DSE was proposed in [2] whose main drawback is besides requiring manual intervention to decide UF, considers only evenly divisible UFs as potential candidates. This results in chances of an optimal UF being sacrificed. In [3] GA is used for exploring the design space, but it explores only datapath resource configuration as output, and is unable to perform exploration of loop unrolling factor of CDFGs for optimal scheduling. Approach [4] accepts only a scheduled data flow graph as an input as well as does not explore UF. This signifies the inability of their approach to resolve the scheduling problem as well as optimize unrolling factor for CDFGs. Further authors in [4] have proposes a cost function which does not consider resource binding (area of mux and demux). In [5] although exploration of an optimal scheduling is done, however, loop based CDFGs are not handled. Besides, GA is used which results in exponential time complexity. In [6] bacterial foraging optimization is used for exploration of datapath resource configuration, but optimization of loop unrolling with various candidate unrolling factors are not considered. Finally in [7], the authors introduced a tool for HLS whose limitation is that the unrolling factor for the loop is user-directed and is therefore not able to automatically determine optimal combination of UF and datapath configuration together during scheduling. Various deterministic and non-deterministic approaches were previously used to solve DSE problems. These have been discussed in the sections below and their drawbacks are given. We studied these approaches and logically concluded that BFOA is the most efficient approach.

Chapter 3: Analysis

Deterministic Methods [9]

- These focus is on the amount of influence each resource unit can have on each of the parameters of the objective function.

- A "priority factor" is calculated for each resource such as adder, multiplier etc. for each parameter namely area, time of execution and power consumption. This priority factor gives us an understanding on to what extent a particular type of resource can have its influence on the parameters of the objective function.

Based on the priority factors the search space is ordered with respect to each parameter and those points which satisfy all the constraints are considered. For example If there are forty design points in the design space and only a set of twenty satisfy the given power constraints and a set of ten satisfy the given time constraint. Then say there are only five design points which satisfy both the power and the time constraints. These five are now ordered based on the priority factor of the area and configuration which give the minimum magnitude is chosen as the global best and the solution for the DSE problem.

- The main drawback of this approach is that it is an exhaustive search of the design space and therefore takes a lot of time for evaluating each and every point and comparing them.
- This algorithm is stated to be taking hours of time for simpler DSE problems which doesn't involve any loop unrolling, imagine the time it would be taking for complex CDFGs.

Non-Deterministic Methods

- **Heuristic Based** - Heuristic approaches like simulated annealing, hill climbing randomly pick a possible solution inside the solution space and determine if it is better than the current best solution.

- **Evolutionary Algorithms** - Genetic algorithms reach to the solution over a few generations by eliminating the lower health off-springs.

- **Bio-Inspired** - Bio-inspired techniques like Particle-Swarm optimization and Bacterial Foraging optimization are developed by mimicking biological process of various organisms.

Simulated Annealing (SA)

Initially a point in the search space is picked randomly and exploration is done from this point. It moves to the next position in the search space only if it's better than the current position or if a probability factor is less than the (e^(-delta/temperature)).

delta=new position-old position and temperature is decreased by a factor 'alpha' in each iteration.

Genetic Algorithm (GA)

The genetic algorithm is a model of machine learning which derives its behavior from a metaphor of the processes of evolution in nature. This is done by the creation within a machine of a population of individuals represented by chromosomes, in essence a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of evolution. The various steps involved is explained briefly below

- The problem is encoded into chromosomes.
- The 'good' chromosomes which give low objective function value are crossed over with each other to eliminate low quality chromosomes and to predict better solutions.
- Success of the algorithm is highly dependent on the crossover mechanism and the encoding scheme of the problem to the chromosomes.
- GA is a slow starter compared to SA but it gives better results than SA and it has more than one exploration point compared to only one the SA has.
- The problem with GA is that it runs for exponential time even for lesser complex problems and as we are using a control flow data graph compared to the normal data graphs we are bound to get a much more execution time for the algorithm.

Particle Swarm Optimization

- Particle Swarm Optimization (PSO) is an evolutionary approach proposed by Eberhart and Kennedy, inspired by the behavior of bird flocks or schools of fish. This is a bio-inspired algorithm which is getting a great recognition these days.
- It operates on a population of candidate solutions referred to as a swarm.
- Each solution in a swarm is called a particle.

- The best solution in each swarm is called the 'particle best' and the best solution among all the swarms so far is the 'global best'. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm's best known position.
- When the swarm moves, each particle is subjected to a velocity which tends to propel it in the direction of pbest as well gbest.
- The pbest and gbest are updated based on the positions of the particles.
- Almost 'twice' as fast as Simulated Annealing [10].
- The drawbacks this algorithm has is that it’s very rigid by nature. It is dictated by many parameters such as inertia coefficient, velocity coefficient and accelerator coefficient which if not chosen suitably might not lead us to the solution and stuck at the local minima.

Bacterial Foraging Optimization Algorithm (BFOA)

BFOA is a relatively new bio-inspired algorithm which has been gaining a lot of recognition in various fields. It has been giving optimum results in lesser time compared to other approaches in various problems like job scheduling and at the same not very rigid as compared to the PSO. It is very simple and have very few factors that dictate the search criteria. It was proposed by Passino by mimicking the movement of the E.coli bacteria. The process basically consists of four steps

1. Chemotaxis
2. Swarming
3. Reproduction or replication
4. Elimination and dispersal

Chemotaxis

Chemotaxis is the process in which the locomotion of the bacteria is stimulated by tumbling and swimming using the flagella. It picks a direction randomly and swims in the direction if the objective function value decreases in that direction, otherwise it tumbles again and changes it direction.

Suppose if bacterium 'i' is at position theta(i), the new position of the bacteria will be "theta(i)+c(i)*(del(i)/|del(i)|)", where c(i) is the step-length and del(i) is the tumble vector.

Swarming

Swarming is a collective behavior observed in these bacteria. The bacteria are stimulated by high levels of 'succinate', which helps them to aggregate into groups and thus move as concentric patterns of swarms with high bacterial density.

Reproduction and Replication

Replication is a process in which, half of the bacteria of low health (high objective function value), are destroyed and the remaining half are replicated and put in the previous positions only. These bacteria again follow the chemotaxis independently.

Elimination and Dispersal

Changes in the local environment of the bacteria, such as temperature changes can occur killing a part of the bacteria population in a particular region and dispersing the rest to some random positions.

Mapping BFOA to a DSE problem

The BFOA explained above illustrates a normal BFOA which is nothing but an optimization algorithm. We need to map this general BFOA to the DSE problem. These are few issues which were dealt with during the design of the flowchart and the pseudo code of the algorithm for the exploration process.

1. Is replication algorithm necessary?

Ø If we use a replication algorithm, we will be re-evaluating the same point twice, which is time consuming and potentially not necessary. So we decided not to implement the replication algorithm.

2. What should be the size of the population?

Ø The size of the population is to be decided in such a way that, it should be sufficient to explore most of the design space and at the same time, should also not be time consuming. We decided to keep the default population size to be 3, but we have also tested with two more population sizes: 5 and 7.

3. How many elimination dispersal are to be introduced and what is the criteria for it?

Ø The elimination-dispersal serves the same purpose of the mutation algorithm in the Genetic Algorithm. It induces diversity in the solutions and avoids premature convergence. So conventionally, the elimination dispersal are 5 and the criteria is all the 5 elimination dispersal steps are evenly spaces among the number of chemotactic steps.

4. How will tumble operate?

Ø Tumble operates by choosing a vector which is made up of random components, which then is divided by its Euclidean norm, to give a unit vector, which is known as the tumble vector.

5. When will chemotactic step increment?

Ø There are two cases when a chemotactic step increments

I. If the newly found position after tumbling is better than the previous position of the bacteria.

II. If the bacteria cannot find a better position even after 5 tumblings.

6. How will violation during exploration be handled during chemotaxis and dispersal?

Ø If during exploration the bacteria moves out of the design space, we again bring it back and place it on the boundary, by which the violation during exploration is handled.

7. What is the elimination dispersal mechanism?

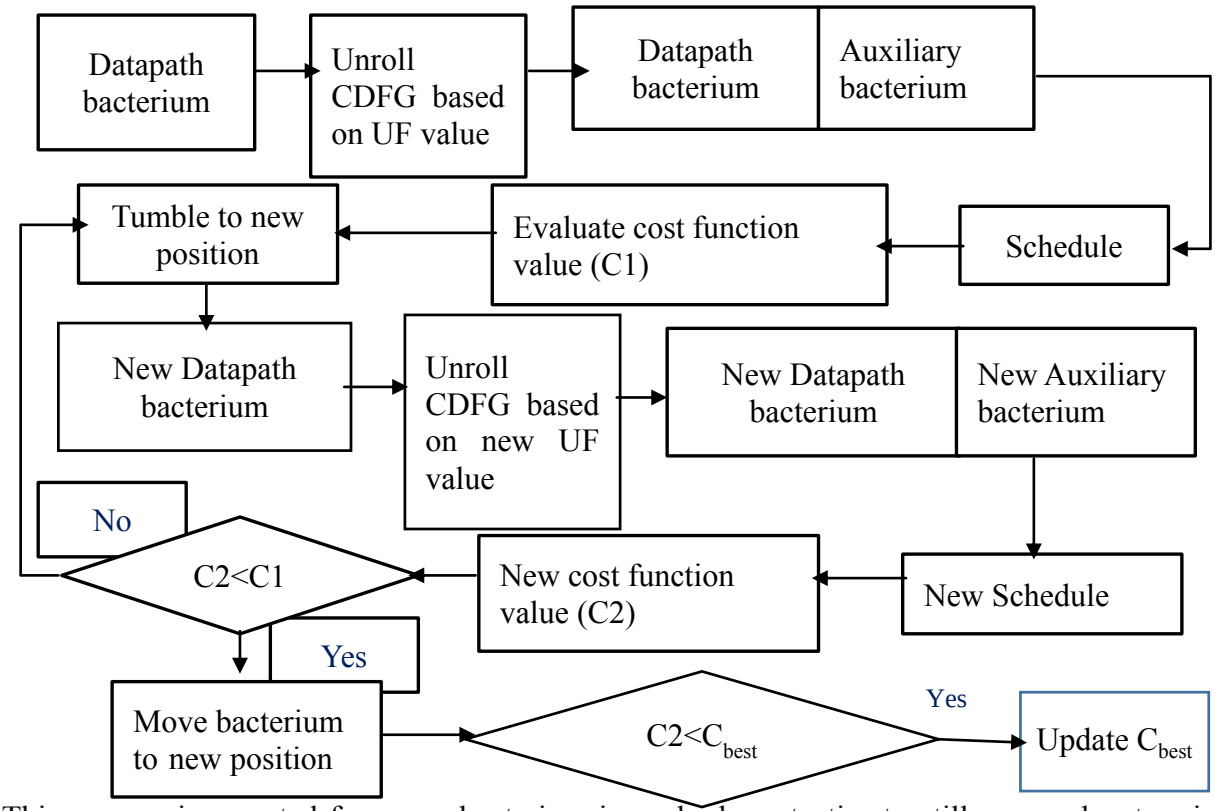
Ø The dispersal mechanism should be in such a way that it improves the health of the population, by killing the lower health bacterium and putting new ones at better positions than the previous ones.

8. What should be initial positions of the bacteria?

Ø The initial positions of the bacteria should be in such a way that they are equally spaced in the design space so that all the points have the equal probability of being visited.

Chapter 4: Proposed Algorithm

The input to the proposed framework is a Control Data Flow Graph (CDFG) of an application that describes the behavioral description of the data-path, set of user specified design constraints for power and delay, and the module library that contains viz. maximum resources available, clock cycles and area of each resource. The flow chart of the proposed BFOA driven exploration process is shown in Figure 6. The proposed methodology is based on a novel bacterium encoding structure (consisting of two components called 'datapath bacterium' and 'auxiliary bacterium') for control data flow graph. This bacterium encoding scheme is responsible for integrated exploration of optimal scheduling for loop based CDFGs. The 'datapath bacterium' is capable of simultaneously exploring the optimal resource configuration array and unrolling factor. On the other hand, the auxiliary bacterium is encoded through load value and utilization value metric which resolves priority conflict between multiple operations during scheduling. The 'auxiliary bacterium' behaves as a support bacterium to its corresponding 'datapath bacterium' and is not subjected to evolutionary operation (or evolution) during exploration.



This process is repeated for every bacterium in each chemotactic step till we reach a termination condition.

Termination Conditions: There are two termination conditions for the algorithm.

1) When we have reached the maximum limit of the chemotactic steps (N_c).
2) When the C_{best} hasn't changed for 30 chemotactic steps.

Figure 6 Proposed Flowchart of the Exploration Process

4.1 Encoding/Initialization of the Datapath Bacterium

The presented datapath bacterium (Bn) of the proposed generic bacterium structure for CDFGs is provided in eqn. (1). This proposed datapath bacterium comprises of two segments: i) array of resource types (resource configuration) ii) loop unrolling factor (UF). The size of the initial population of the bacteria is arbitrarily assumed as three which are equally spaced in the design space. The first bacterium (B1) has been encoded with resource configuration which results in the serial implementation (indicating worst case latency amongst design solutions in the space). The second bacterium (B2) has been encoded with resource configuration which results in the maximum parallel implementation (indicating best case latency amongst design solutions). The third bacterium (B3) is placed in the middle of the design space (mid value (MV) between serial and parallel implementation bacteria). The initial configurations of the bacteria are described as follows:

Bn = ((R1), (R2)……. (Rn), (UF)) (1)

B1= (min(R1), min(R2)…….min(Rn), min(UF)) (2)

B2=(max(R1),max(R2)…….max(Rn),max(UF)) (3)

B3= (…. (min(Rn) + max(Rn))/2, (min(UF) + max(UF))/2) (4)

Where R1,R2…,Rn are various resource types and UF is the loop unrolling factor.

4.2 Encoding of the Auxiliary Bacterium

Once the initial population of datapath bacterium is created as described above, the unrolled untimed CDFG corresponding to the UF value (specified in the datapath bacterium) is constructed for each parent. This indicates that for B1, B2 and B3 the corresponding unrolled CDFGs are constructed for UFmin, UFmax and UFMV. An auxiliary bacterium is then generated using a novel encoding technique corresponding to each unrolled untimed CDFG of bacterium population (B1, B2 & B3). This indicates that in general an auxiliary bacterium exists for each datapath bacterium. This auxiliary bacterium acts a priority resolver for operations competing during scheduling. The priority resolution (E) of operation ‘oi’ is performed through proposed encoding scheme given as :

$$E\,(oi) = W1 * (LV) + W2 * (UV) \qquad (5)$$

Where LV is the load value and UV is the utilization value for each node (operation).W1 and W2 are designer specified weights assumed as 0.5 each. The load value of an operation is defined as the summation of the load factor (delay) of each successor operations including the operation itself. The utilization value of an operation

is its number of child branches. An example of encoding value of operations (for auxiliary bacterium)for an example CDFG (Fig.7)is provided in Table 1.

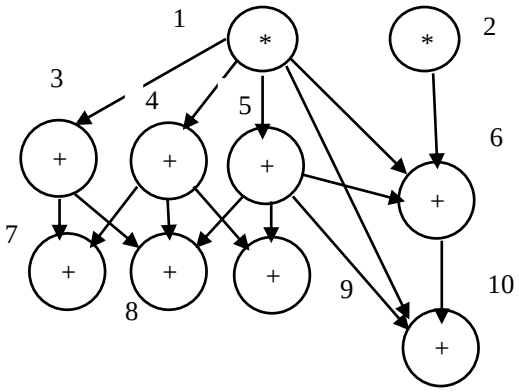
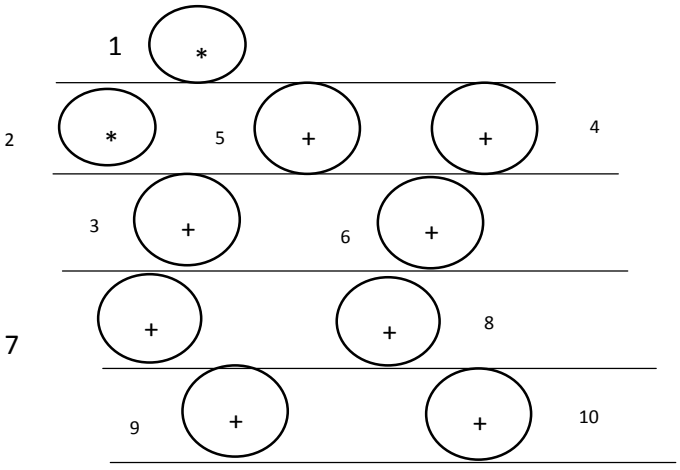


Figure 7 A CDFG example

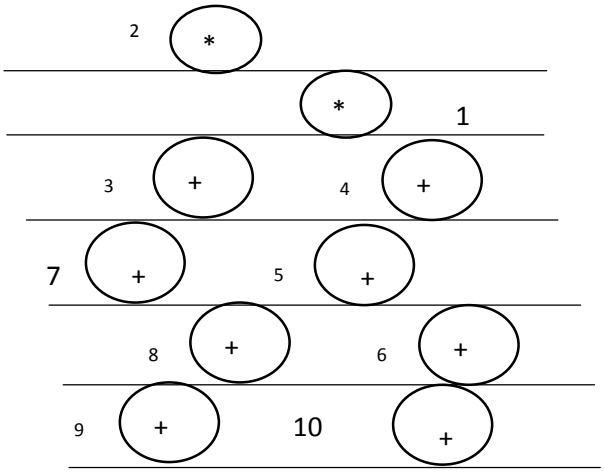
o1	o2	o3	o4	o5	o6	o7	o8	o9	o10
4	2	2	2.5	3	1.5	0.5	0.5	0.5	0.5

Table 1 . Encoded values for auxiliary bacterium for CDFG in figure 6

Further, a motivational example on this encoding strategy, which integrates utilization value besides load value, is described below. *Scenario 1*: assume two encoded operations with load value (and having same load values), whereby the tie is broken by randomly selecting the operations during scheduling. *Scenario 2*: the two encoded operations have same load value. However, instead of randomly breaking, the tie is broken based on the proposed encoding strategy (function of load value and utilization value) during scheduling. Example, for a resource configuration, 2(+) and 1(*), the schedule in scenario 1 (for CDFG in Fig.7) uses only load values to resolve operation priority during scheduling and thereby consumes 6 control steps compared to schedule in scenario 2 which uses proposed encoding to resolve operation conflict during scheduling and consumes only 5 control steps.



Scenario 2: Scheduling with encoding



Scenario 1: Scheduling with load values

Figure 8 Schedules for comparing two encoding schemes of auxiliary bacterium

4.3 Proposed Movement of Bacterium

A bacterium moves through chemotactic movement in every step (j), the proposed DSE mechanism explores new feasible solutions is given in figure 5. However, after a designer specified periodic intervals ('y^{th}' iteration step), the process of elimination dispersal (ED) occurs. The ED algorithm is repeated for N_{ed} times (a temporary counter '*l*' is initialized with the values of N_{ed} which decrements after every corresponding occurrence of ED operation; where 'N_{ed}' is a designer specified variable that defines the number for times, ED occurs. Therefore, it tracks the number of times ED is further allowed). Further, arrays (*Ed [j-]*) is created for ED process, each to store the outcome, checking whether ED has been performed in last iterative step. This storage structures are necessary to determine whether variables '*y*' needs up gradation. If *Ed [j--]* has taken place, therefore, the 'y' needs to be updated.

j (Step/iteration) = 1
Repeat
{
If $j = (n*N_c/N_{ed})$ // $1 <= n <= N_{ed}$
Then perform Elimination-Dispersal mechanism
Tumble: Generate a random vector $\Delta_m(i)$, whose each component is in the range of [-1,1].
Tumble-count=1
Swim-count=1
1.Generate a tumble vector $\Delta_m(i)$.
2.$B_i^{new} = B_i^{current} + C(i) * (\frac{\Delta_m(i)}{||\Delta_m(i)||})$ (6)
3.Go to cost function based on the decoded CDFG and store it in F_i^{new}.
4. If $F_i^{old} > F_i^{new}$ and tumble-count<=5
Tumble-count=tumble-count+1
Goto step 1.
5.Else if Swim-count < 5
$B_i^{current} = B_i^{new}$
$F_i^{current} = F_i^{new}$
Goto step 2.

j =j + 1
}

Figure 9 A bacterium's locomotion process

The chemotactic function (eqn. (6)) incorporates a behavior of tumble/swim in order to explore the new positions; where *C(i)* is the step size taken in random direction specified by the tumble and Δ is a random

vector whose elements lie in [-1, 1]. A substantially large value of *C(i)* is required for DSE, in order to avoid redundant solutions.

Elimination-Dispersal Mechanism

In order to implement the elimination dispersal mechanism, new replacements are randomly initialized over the search space (between the least fit and best fit bacterium position but beyond their midpoint, but closer to the best fit bacterium) by eliminating the least fit bacterium. If the new replacement obtained is already found to be explored, and then dispersal is repeated. Further, if the new cost of the dispersed bacterium position is found to be higher than the replaced bacterium, then it is not accepted. The ED mechanism has been adopted from [9].

Identifying and discarding unproductive unroll factors before start of algorithm [1]

The unroll factors are screened to reduce the execution time, by unrolling the ineffective ones. The pseudo-code for the algorithm that is used to screen the unrolling factors is given in figure 6.

Pre-processing of unrolling factor

Input – value of 'I' (Total no. of loop iteration)

Output – screened set of unrolling factor (UF)

```
1. Begin
// Screening of UF
2. For UF =2 to I
Do
        2.1 IF ((I mod UF < UF/2 ) &&  (UF <= I/2))Then
        //Add UF into the accepted UF list
                2.2 Accepted UF[k] = UF
                2.3 k++
                2.4 End IF
        2.5 End For
3. End
```

Figure 10 pre-processing of unrolling factors

4.4 Evaluation Models

There are two evaluation models each to estimate the Power and Execution time of each design specification. These weighted sum of these two form the cost function which is used to evaluate the points in the design space.

Evaluation model for estimation of execution time

The model used for evaluating the execution time taken by each point in the design space is as follows :

$$\mathbf{T_E = (C_{first} + (UF - 1)C_{II})*alpha + (I\%UF)*C_{first}} \qquad (7)$$

- 'C_{first}' is time required to execute first iteration
- 'I' is the maximum number of iteration (loop count)
- 'αlpha' is floor(I/UF)
- 'C_{II}' is the difference in time between outputs of consecutive iteration in ns.

This is an estimation model for T_E, where the necessity of tediously unrolling the CDFG is not required to calculate T_E, unless the number of of independent operations required to be performed in parallel due to unrolling exceeds available resources (specified in bacterium).

Evaluation model for estimation of power

The total power(P^T) represented by each point in the design space is a combination of two components namely static power(P^S) and dynamic power(P^D).

The total power consumed is given by: $\mathbf{P^T = P^S + P^D}$ (8)

The equation for P^S is given by:

$$\mathbf{P_S} = \left(\sum_{i=1}^{v}(N_{Ri} \cdot K_{Ri} + N_{mux} \cdot K_{mux})\right)\mathbf{*P_C} \qquad (9)$$

- 'N_{Ri}' represents the number of instance of resource R_i
- 'K_{Ri}' represents the area occupied by resource R_i
- 'v' is the number of resource types
- 'N_{MUX}' is number of the multiplexer

- 'K_{MUX}' is area occupied by the multiplexer
- 'P_C' denotes the power dissipated per area unit (e.g. transistors)

The equation for P^D is given by:

$$\mathbf{P_D} = \frac{\boldsymbol{alpha} * (\boldsymbol{E_{FU}} + \boldsymbol{E_{mux}})}{\mathbf{(Cfirst + (UF - 1)\ CII) * alpha + (I\%UF) * Cfirst}} \qquad (10)$$

- 'P_C' denotes the power dissipated per area unit (e.g. transistors)
- E_{FU} and E_{mux} are the energy consumed by resources and mux respectively.

Cost Function

The cost function is a weighted sum of the power and execution time compared to the maximum power and maximum execution time of the whole design space. The maximum power is obtained for the configuration of most parallel implementation, while maximum execution time is obtained for the most serial configuration. The cost function of each design space configuration is given by the equation:

$$\boldsymbol{C_f^b} = \boldsymbol{W_T} \frac{\boldsymbol{T_E}}{\boldsymbol{T_{max}}} + \boldsymbol{W_P} \frac{\boldsymbol{P_T}}{\boldsymbol{P_{max}}} \qquad (11)$$

- $\boldsymbol{C_f^b}$ = Fitness of bacterium b
- $\boldsymbol{W_T}$, $\boldsymbol{W_P}$ = User defined weights for power and execution time
- $\boldsymbol{T_{max}}$, $\boldsymbol{P_{max}}$ = maximum Time and power of any schedule

Chapter 5: Implementation

To test the theoretical conclusion that BFOA designed by us is a better approach compared to the previous approaches we have implemented the BFOA and GA algorithms. The GA approach used for comparison has been adopted from [11]

5.1 BFOA.exe

This is a program written in java to implement the BFOA as discussed in the previous sections. It's very user friendly. A screen shot of this is shown in the figure 7 below.

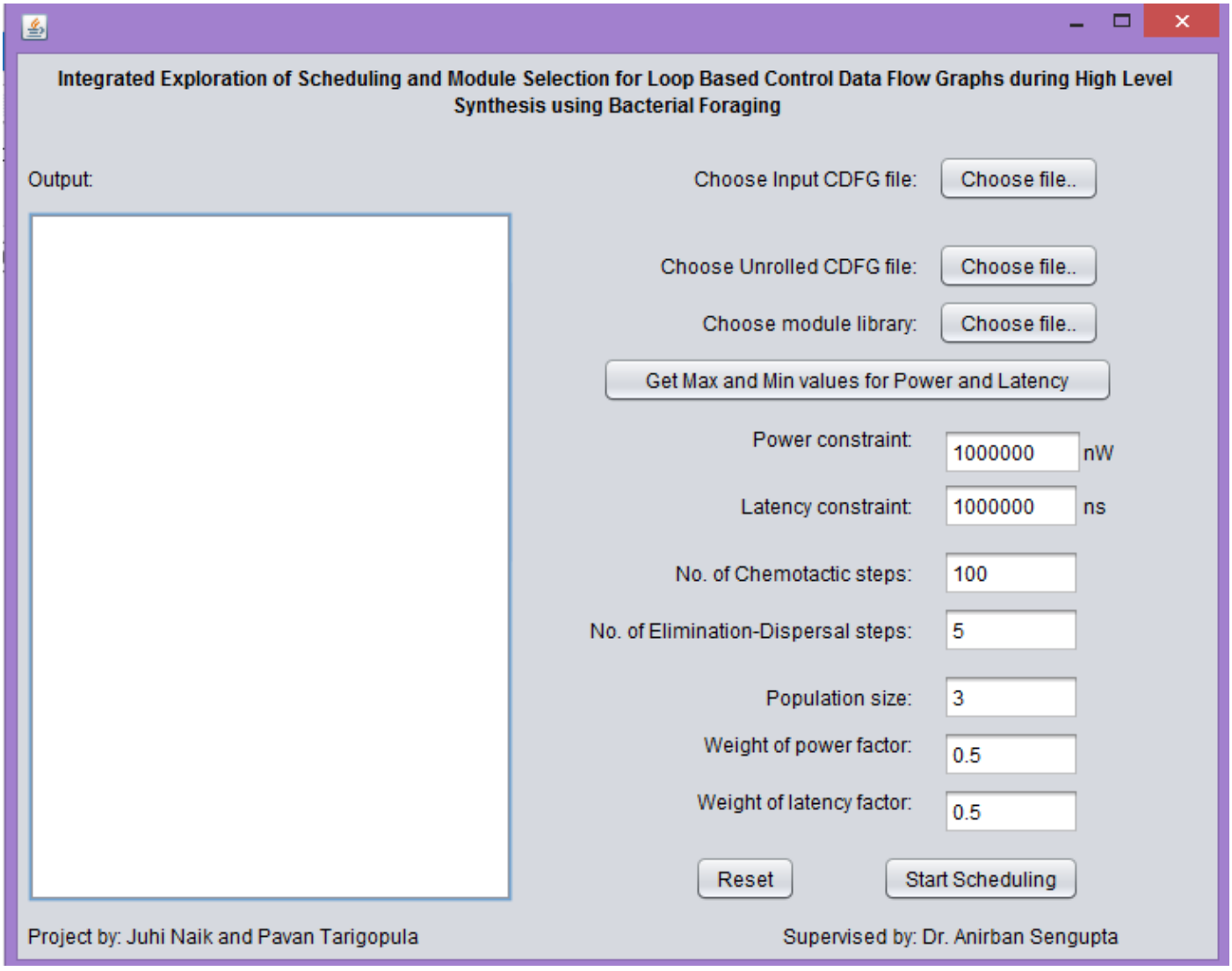


Figure 11 Screen shot of BFOA.exe

The inputs are same as that of any normal HLS problem. In addition the user can choose the parameters such as No.of chemotactic steps, elimination dispersal steps, Population size, and respective weights of power and latency during the evaluation of cost function. The 'get Max and Min values for Power and Latency' button gives the maximum power and maximum latency of all the design space points and outputs it as shown in figure 8.

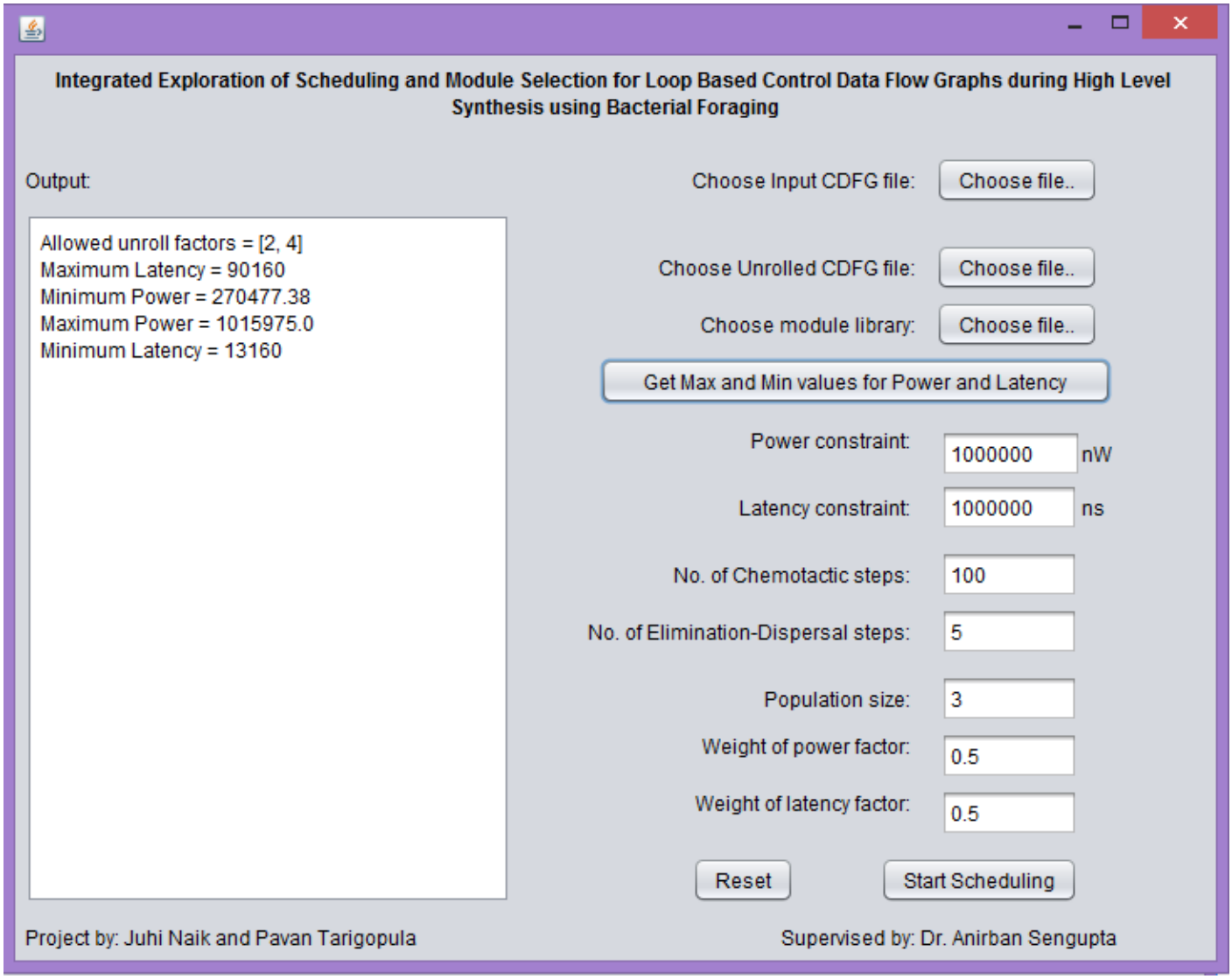


Figure 12 Screen shot of BFOA.exe giving maximum power and latency and the unrolled factors allowed for the FIR filter.

After filling all the user defined fields now click on start scheduling. Then it runs the BFOA as presented in the previous sections and in the output field present on the right side of the layout presents the iteration of the convergence, the final configuration of the global best found, the cost function value, power and latency of the global best, the time taken for the execution of the algorithm. It also presents the solution found out by the brute force approach to cross check if we have the reached the golden solution, which is the global optimum solution or not. The screen shot showing the outputs in shown in figure 9.

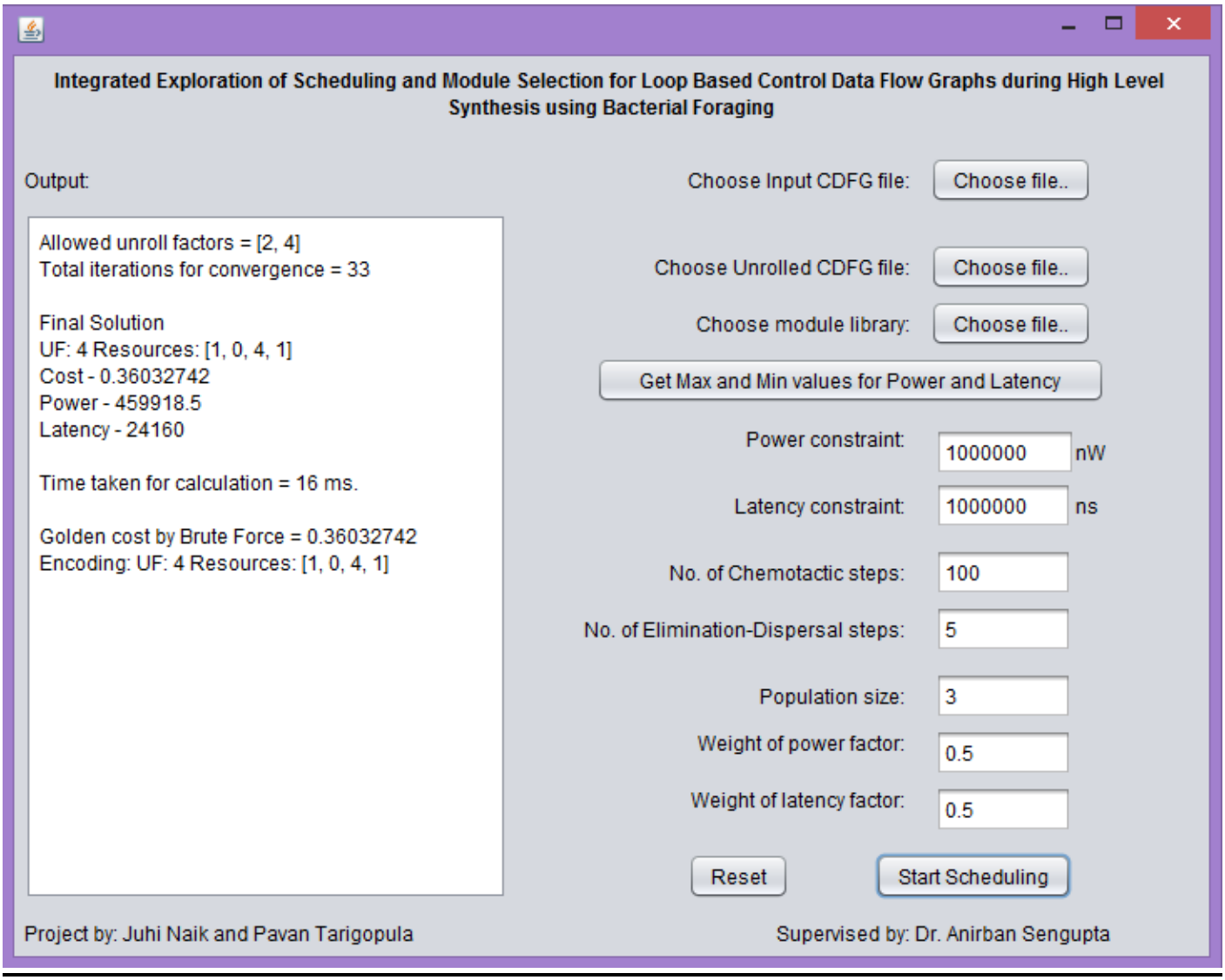


Figure 13 Screen shot of BFOA.exe showing the final output.

In addition to the output shown on the layout of the program, another file is generated in the folder from where the program is run with the file name in the format 'BFO-<timestamp>', the timestamp of the time of execution of the program is appended each time the program is run. This contains all the local best configurations after each iteration. The reset button resets all the input fields and a new configuration can be run.

5.2 GA.exe

This is a program written in java to implement the GA present in [11]. A screen shot of this is shown in the figure 10 below.

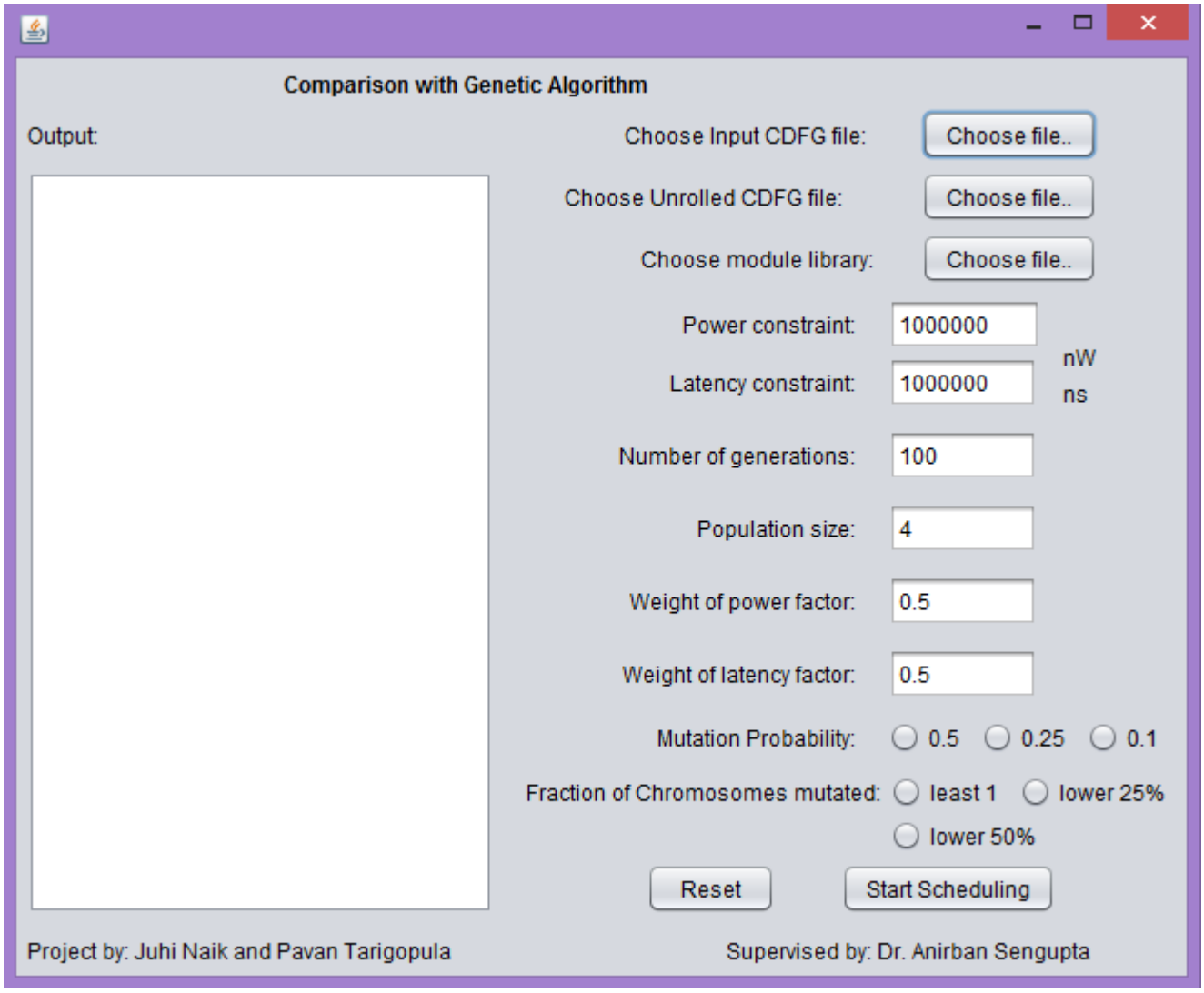


Figure 14 Screen shot of GA.exe

The only difference between GA.exe and BFOA.exe is the genetic algorithm specifications the user needs to supply. The Mutation probability can be varied between 0.5,0.25 and 0.1. The fraction of chromosomes mutated can be varied between three options. It can be as small as 1 or 25% or 50% of the total size of the population. After giving all the inputs and the constraints on power and latency, press the start scheduling button for the program to run.

After the execution of the program and a solution is reached the output is displaced in the right hand side of the layout. This contains the total no.of iterations taken for convergence, the configuration of the final solution and the time taken for convergence. A screenshot of the layout showing the output is shown in figure 11.

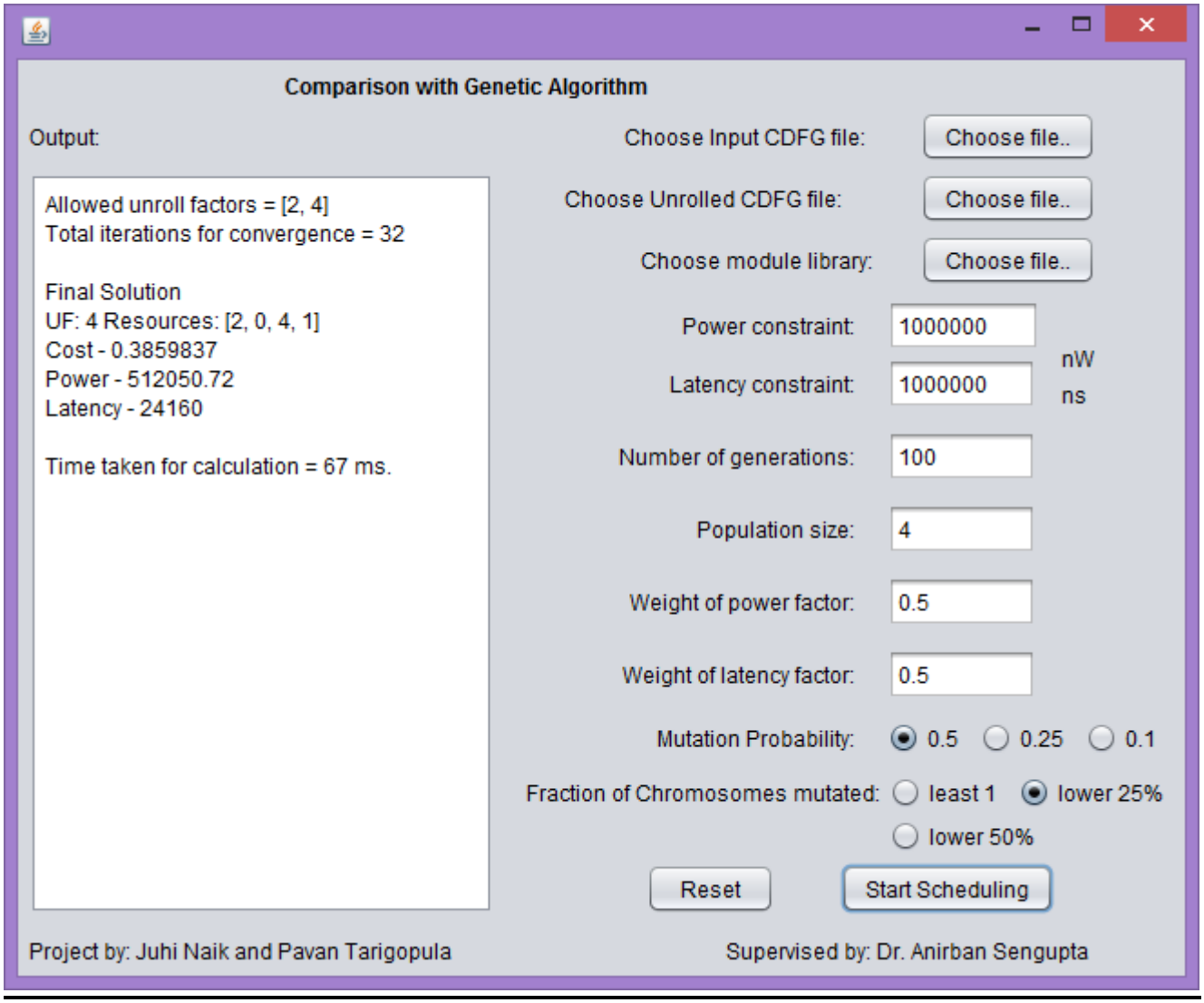


Figure 15 Screen shot of GA.exe showing the output

As in the case of BFOA.exe, GA.exe also generates a file with file name in the form of 'GA-<timestamp>' which has the configurations and cost function values of the local best after each generation.

Chapter 6: Testing and Results

The proposed approach has been implemented in Java and run on Intel core i5-2450M processor, 2.5 GHz with 3 MB L3 cache memory and 4 GB DDR3 RAM. The results are presented in three phases:

6.1 Sensitivity Analysis on Cost and Convergence Iteration due to Population Size

Benchmark	Population	Cost	Exploration time (ms)	Iteration at which convergence is achieved
FIR	3	0.3603	62	29
	5	0.3603	81	26
	7	0.3603	91	25
FFT	3	0.2245	197	36
	5	0.2245	253	35
	7	0.2245	300	34
Differential Equation	3	0.3013	165	38
	5	0.3013	184	37
	7	0.3013	247	34
Test Case	3	0.3735	176	34
	5	0.3735	208	33
	7	0.3735	274	31
IIR Butterworth	3	0.505	39	24
	5	0.505	58	23
	7	0.505	69	23
MPEG Motion Vector	3	0.2991	47	26
	5	0.2991	56	25
	7	0.2991	68	24
JPEG downsample	3	0.6514	63	25
	5	0.6514	83	23
	7	0.6514	109	23

Table 2 Sensitivity Analysis of the impact of Population Size on Cost and Convergence Iteration

- As evident in table 2, for all benchmarks, as the population size increases, the convergence iteration decreases (however, the exploration time increases).
- This is because the computational complexity per iteration is more for larger population size.

- Nevertheless the cost of final solution (quality of results: QoR) remains same for all benchmarks regardless of population size.

6.2 Results of Proposed approach

Benchmark	Resources and UF	Execution Time(us)		Power (mW)	
		Constraint	Proposed Solution	Constraint	Proposed Solution
FIR	UF = 4, 1(+), 4(*), 1(<)	60	24.16	0.5	0.4599
FFT	UF = 2, 1(+), 1(-), 4(*), 1(<)	800	292.28	2	0.6519
Differential Equation	UF = 4, 1(+), 1(-), 4(*), 1(<)	600	267.24	1.2	0.6377
Test Case	UF = 2, 1(+), 3(*), 1(<)	500	400.86	1.5	0.3859
IIR Butterworth	1(+), 3(*)	30	22.54	0.35	0.3119
MPEG Motion Vector	1(+), 5(*)	36	33.27	1	0.5511
JPEG downsample	2(+), 1(*)	26	24.97	0.6	0.3188

Table 3 Results of proposed approach.

- As evident in Table 3, the solution found indicates an optimal combination of resource array and loop unrolling factor CDFG's, where the final solution comprehensively meets the user constraints of power and delay (execution time) as well as minimizes the final cost (As per eqn. (11)).
- It is worthwhile to mention that for all benchmarks we have achieved the real global best after comparing with the golden solution obtained through brute force method.

6.3 Comparison of QoR and Exploration Runtime with previous approaches

Benchmark	Final Solution			Exploration Run time			QoR(Cost)		
	Proposed	[5]	[3]	Proposed(ms)	[5]	[3]secs	Proposed	[5]	[3]
FIR	4(*), 1(+), 1(<),UF = 4	3(*),1(+), 1(<),UF= 8	4(-), 1(+), 1(<),UF= 8	62	4.31min	5.03	0.36	0.41	0.38
FFT	4(*), 1(+), 1(-) ,1(<), UF = 2	3(*), 2(+), 1(-), 1(<), UF = 16	2(*), 1(+), 1(-), 1(<), UF = 16	197	>1 hr	141	0.22	0.6	0.7
Differential Equation	4(*), 1(+), 1(-),1(<), UF = 4	4(*), 1(+), 2(-), 1(<), UF = 16	3(*), 1(+), 1(-), 1(<), UF = 16	165	>1 hr	436	0.30	0.52	0.51
Test case	3(*),1(+),1(<), UF = 2	2(*), 4(+), 1(<), UF = 36	2(*), 1(+), 1(<), UF = 36	176	>1 hr	351	0.37	0.78	0.87
MPEG MMV	5(*),1(+)	3(*),1(+)	5(*),1(+)	47	5.45min	6.63	0.29	0.39	0.36
JPEG DS	1(*),2(+)	1(*),2(+)	1(*),1(+)	63	2.5 min	8.21	0.65	0.65	0.77

Table 4 Comparison of QoR and Exploration Runtime

The Proposed approach is compared with two other approaches in [3] and [5] and the comparison is given above in table 4.

- As evident in table 4, the QoR of the proposed approach is significantly better than [3] and [5].
- This is because optimization of loop unrolling was not performed simultaneously with datapath resource configuration in them.
- Additionally, adaptive features such as tumbling which assists in changing direction when a search path is found ineffective does not exist in genetic based approaches.
- Besides above, the encoding of the individuals did not include utilization metric concept in resolving priority during scheduling which assists in reducing latency. Therefore two different schedules (different latency) are possible for same resource configuration (this has been established in section 3.2).
- The exploration runtime of [3 and [5] were higher because, both the previous approaches being driven through genetic algorithm induces greater computational complexity than proposed fast bacterial foraging driven DSE process.

6.4 Comparison of QoR and Exploration Runtime with GA of [11]

Benchmark	Mutation Factor (Fm)	Cost	Exploration time (ms)	teration at which convergence is achieve	Configuration
FIR	single least fit	0.385984	99	33	UF: 4 2(+), 4(*), 1(<)
	lower 25%	0.360327	237	44	UF: 4 1(+), 4(*), 1(<)
FFT	single least fit	0.263173	275	50	UF: 5 1(+), 1(-), 8(*), 2(<)
	lower 25%	0.253643	169	50	UF: 4 1(+), 3(-), 4(*), 1(<)
Differential Equation	single least fit	0.312822	141	33	UF: 5 1(+), 1(-), 4(*), 1(<)
	lower 25%	0.310825	153	45	UF: 5 1(+), 1(-), 5(*), 1(<)
Test Case	single least fit	0.41585	121	34	UF: 2 1(+), 4(*), 2(<)
	lower 25%	0.41585	158	34	UF: 2 1(+), 4(*), 2(<)
IIR Butterworth	single least fit	0.535144	206	33	1(+), 2(*)
	lower 25%	0.504985	273	45	1(+), 3(*)
JPEG downsample	single least fit	0.768476	71	33	1(+), 4(*)
	lower 25%	0.651423	77	33	2(+), 1(*)
JPEG Motion Vecto	single least fit	0.330782	408	33	2(+), 7(*)
	lower 25%	0.330782	453	33	2(+), 7(*)
Legend:	Premature Convergence	Golden Solution		Reaching maximum iterations without convergence	
Note: Pm = 0.25, Max population size = 100					

Table 5 Comparison of QoR and Exploration Runtime with GA using GA.exe

As you can see from the table 5 above that GA reaches to the golden solution only for two benchmarks and the execution time is also very very high compared to the proposed approach.

Chapter 7: Conclusion

An automated exploration of optimal datapath configuration and loop unrolling factor for integrated scheduling and module selection of CDFG's using bacterial foraging is proposed. The proposed approach has been able to attain improvement in QoR and reduction in exploration runtime compared to [3], [5]. We have also implemented the GA in [11] and found out that it's reaching to the golden solution only for three benchmarks. Out of the seven bench marks tested three converged prematurely and one has not converged at all within the limit of the maximum number of iterations. For all the bench marks the execution time of the GA is much more than the proposed BFOA.

Chapter 8: Future Work

- Better approximations to evaluate design spaces points like considering registers and latches also.
- Using better encoding scheme for encoding the bacteria.

References

[1] A Sengupta, VK Mishra, “Automated exploration of datapath and unrolling factor during power–performance tradeoff in architectural synthesis using multi-dimensional PSO algorithm”, Elsevier Journal on Expert Systems,Vol. 41, Issue 10, 2014, pp. no:4691-4703.

[2] Holzer, M.; Knerr, B.; Rupp, M., "Design Space Exploration with Evolutionary Multi-Objective Optimisation," Proc. International Symposium on Industrial Embedded Systems, 2007, pp.126 – 133.

[3] V Krishnan, S Katkoori, A Genetic Algorithm For The Design Space Exploration Of Data Paths Using High-Level Synthesis,IEEE Transactions On Evolutionary Computation, Vol. 10, No. 3, June 2006.

[4] C. Mandal, P. P. Chakrabarti, and S. Ghose, “GABIND: A GA approach to allocation and binding for the high-level synthesis of data paths,” IEEE Trans. on VLSI, vol. 8, no. 5, pp.747–750, 2000.

[5] A Sengupta, R Sedaghat, “Integrated Scheduling, Allocation and Binding in High Level Synthesis using Multi Structure Genetic Algorithm based Design Space Exploration System”, Proc. of 12th IEEE Intl Symposium on Quality Electronic Design (ISQED), California, 2011, pp. 486-494.

[6] A Sengupta, S Bhadauria, “Automated Exploration of Datapath in High Level Synthesis using Temperature Dependent Bacterial Foraging Optimization Algorithm”, Proc. of 27th IEEE Canadian Conference on Electrical & Computer Engineering, Toronto, 2014, pp. 1- 5.

[7] Gupta, S.; Dutt, N.; Gupta, R.; Nicolau, A., "Loop shifting and compaction for the high-level synthesis of designs with complex control flow," in Proc. of DATE, 2004, pp.114-119.

[8] Express Benchmark Suite, http://express.ece.ucsb.edu/benchmark/.

[9] Anirban Sengupta, Reza Sedaghat , Zhipeng Zeng, “A high level synthesis design flow with a novel approach for efficient design space exploration in case of multi-parametric optimization objective”, Microelectronics Reliability,31/3/2010,volume 50,issue 3,pages 424-437.

[10] Professor M. Bolic, “Design space exploration: comparaitive study of simulated annealing and particle swarm optimization”, http://carg.site.uottawa.ca/doc/ELG6158VishalThareja.pdf, December 3, 2007.

[11] Anirban Sengupta, Reza Sedaghat, Pallabi Sarkar, “A multi structure genetic algorithm for integrated design space exploration of scheduling and allocation in high level synthesis for DSP kernels”, journal of Swarm and Evolutionary Computation published by Elsilvier dated 31/12/2012,volume 7, pages 35-46.

[12] Methods for Evaluating and Covering the Design Space during Early Design Development, Matthias Gries, Technical Memorandum UCB/ERL M03/32, August 12, 2003

[13] Modeling Loop Unrolling Approaches and Open Issues, João M. P. Cardoso and Pedro C. Diniz, in International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS IV), Samos, Greece, July 19-21, 2004

[14] M. Holzer, B. Knerr, and M. Rupp, “Design Space Exploration with Evolutionary Multi-Objective Optimization” Proc. Industrial Embedded Systems, p.125-133, Lisbon, Portugal, 2007