“ACR - ADAPTIVE CODE REFINEMENT”

TECHNICAL CODEATHON REPORT

SUBMITTED TO

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute, Affiliated to VTU)

Bangalore – 560054

**SUBMITTED BY**

|  |  |
| --- | --- |
| **Name** | **USN** |
| M V S Viswanadh | 1MS15CS074 |
| Prabhuling | 1MS15CS090 |
| Anusha Prabhudev | 1MS15CS144 |
| Amrita Behera | 1MS15CS148 |

As part of the Course **Compiler Design- CS61**

SUPERVISED BY

Faculty

**Parkavi.A**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAMAIAH INSTITUTE OF TECHNOLOGY

Jan-May 2018

Department of Computer Science and Engineering

Ramaiah Institute of Technology

(Autonomous Institute, Affiliated to VTU)

Bangalore – 560054



**CERTIFICATE**

This is to certify that **M V S VISWANADH (1MS15CS074)**, **PRABHULING (1MS15CS090), ANUSHA PRABHUDEV (1MS15CS144), AMRITA BEHERA (1MS15CS148)** have completed the **“ACR -ADAPTIVE CODE REFINEMENT”** as part of Technical Codeathon. We declare that the entire content embodied in this B.E. 6th Semester report contents are not copied.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Submitted by     |  |  | | --- | --- | | MVS Viswanadh | 1MS15CS074 | | Prabhuling | 1MS15CS090 | | Anusha Prabhudev | 1MS15CS144 | | Amrita Behera | 1MS15CS148 | | Guided by  Prof. Parkavi  (Dept of CSE, RIT)  (Assistant Professor, Dept. of CSE, RIT) |

Department of Computer Science and Engineering

Ramaiah Institute of Technology

(Autonomous Institute, Affiliated to VTU)

Bangalore – 560054



**Evaluation Sheet**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sl. No** | **USN** | **Name** | **Report Content**  **and Demonstration**  **(11)** | **Speaking Skills**  **(1)** | **Team-**  **work**  **(1)** | **Neatness and care**  **(1)** | **Effectiveness**  **& Productivity (1)** | **Total Marks**  **(15)** |
| 1 | 1MS15CS074 | MVS Viswanadh |  |  |  |  |  |  |
| 2 | 1MS15CS090 | Prabhuling |  |  |  |  |  |  |
| 3 | 1MS15CS144 | Anusha Prabhudev |  |  |  |  |  |  |
| 4 | 1MS15CS148 | Amrita Behera |  |  |  |  |  |  |

Evaluated By

Name: Parkavi.A

Designation: Assistant Professor

Department: Computer Science & Engineering, RIT

Signature:

HOD, CSE

**Table of Contents**

|  |  |  |
| --- | --- | --- |
|  | **Title** | **Page No.** |
|  | Abstract | 4 |
|  | Introduction | 5 |
|  | Literature Survey | 6 |
|  | System Architecture | 8 |
|  | Implementation | 9 |
|  | Results and Discussions | 13 |
|  | Conclusion | 14 |
|  | References | 15 |

**ABSTRACT**

In this research, an adaptive capability is provided to compute-intensive codes. Heavy computations are carried out only when it is necessary. Thereby, increasing the performance of the pre-existing code and provides with an efficient and refined result of the code. The adaptive capabilities of the codes are carried out using the State Grid model. Programmers provide special pragmas to the code that helps regenerate a code at runtime that performs complex computations only where it is necessary. By doing so, the computational complexity as well as time complexity are reduced by significant amount. The paper also aims at incorporating the parallelization technique in compilation of code.

**CHAPTER 1**

**INTRODUCTION**

Many computations in various applications are using approximations in order to achieve the desired result with the best time complexity possible. To be able to reach these specifications of an ideal and optimised method of computation it is required that such a methodology be devised wherein the procedure to implement computation intensive application is optimised and pertains to real-world objects.

State-of-the-art automatic optimization and parallelization compiler techniques depend on the polyhedral model to manipulate computation-intensive kernels. Some techniques have been designed to carry out approximations for some computations automatically. Ignoring some dependencies, providing alternative implementations, skipping some computations are some of the layman techniques used. The polyhedral model structures the application into a net of computations each dealt with as cells of a polyhedron. Each cell is simultaneously worked on to avail the computation in the application. This polyhedral model is encompassed as a State Grid. This poses as a static-dynamic grid for which code is to be generated.

ACR uses a polyhedral code generation technique which approximates results for static code and computes dynamic code with on-the-fly discriminating abilities. The ACR generates the grid and moderates the computations in the grid based on predetermined values and current status.This optimises code, ensures an energy efficient methodology, and real-time problem-solving.

This is a simple basic implementation of the concept of ACR, by taking three different computations involving static or dynamic code generation and observing how with the passage of iterative steps the computations are optimized using approximations where necessary.

**CHAPTER 2**

**LITERATURE SURVEY**

The review starts off with understanding the techniques like Relaxing Program Semantics [2], Code Perforation [3], Green [4] and Code Generation [5]. *Relaxing Program Semantics* is a method in which semantics (conditions) are relaxed while computation of the code is being performed, such that there is not much distortion in the output. It also aims at improving the robustness of the system in case of failures. The distortion bounds are set to the output so that any relaxation in the semantics produces a distortion that is within the bound and the end result is not affected. It aims at combining approximate computing with parallelization which is done automatically by the system in order to provide users with a bound for the relaxation in the semantics.

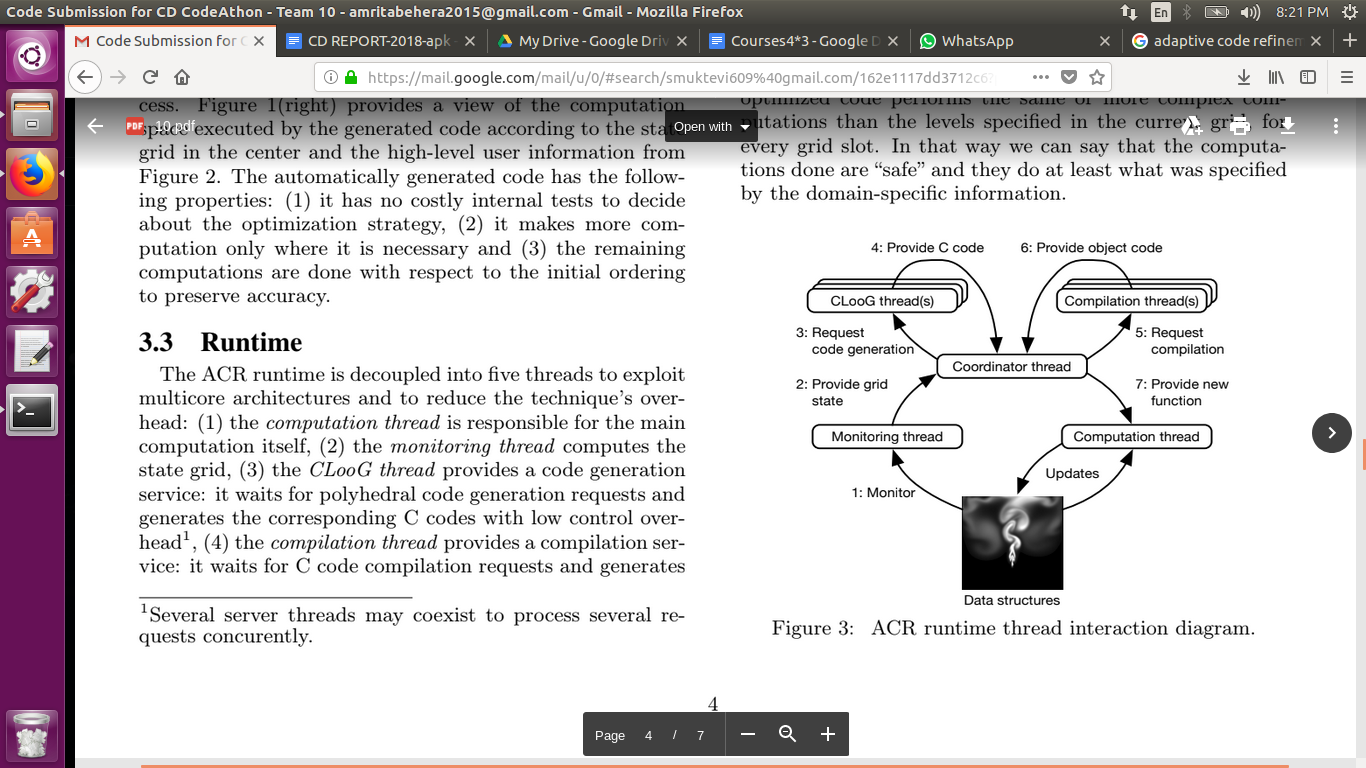
*Code Perforation* [3] is a technique to enhance the performance of the applications by trading it with accuracy. It has a bound that is set for the distortion in the output which is similar to the previously discussed technique. It automatically enhances the performance without the developer’s intervention. It detects the part of the program automatically where the computation has very less or negligent effect on the output. For example, to compress an audio file, it detects the part of the file where it can compress it and that has a very negligent amount of effect in the end result and will be unnoticed by the user. In this way, it compromises on the accuracy thereby increasing the performance without having much distortion in the end result.

*Green* [4] is a framework for supporting energy-conscious programming using a controlled approximation. This paper deals with energy-efficient computing which is required for any scale of devices for any execution of required task materials. The quality of computing should not affect the importance given to the energy consumption of the computations hence we must find optimised methods of such calculations performed. In order to achieve such an optimised approach, a system called Green is proposed wherein programming methodology and concepts are used keeping in mind energy consumption. Along with such ideologies, it also ensures Quality of Service rendered in situ with the provided optimised framework. It operates in two phases namely; the Calibration phase and operational phase. The former deals with building a model of QoS loss caused due to the choice of approximation and the latter deals with making decisions based on the former’s model.

Code Generation in the Polyhedral Model is made easy [5]. Polyhedral representation and scheme has been proved very useful in code optimisation but has also been recognized as belonging to a tedious or difficult category under implementable computational models. Code generators find a hard time coping with code size and overheads. This also takes up a large amount of processing time due to its structured iterative nature. The paper [5] tackles the problem by providing models to enable efficient coding exploring generation of code and code size. Hence approached by separation of selection of an optimising transformation and its application to the source code. The paper provides a flexible transformation framework.

**CHAPTER 3**

**SYSTEM ARCHITECTURE**



*Figure : ACR runtime thread interaction diagram[1]*

The ACR is reduced into five threads, *computation* thread which is responsible for the computation, *monitoring* thread which computes the state grid, *CLoog* thread that provides code generation service, *compilation* thread which provides a compilation service and *coordinator* thread that creates and manages all the other threads. The original code is executed during the first iteration as there is no optimized code available at the beginning of the computation. In every iteration, the monitoring thread watches the monitored data and updates the state grid whenever necessary. When there is a new state grid, the coordinator thread tries to obtain an optimized code which corresponds to the current functionality code. The coordinator thread fetches the new optimized code from CLoog and compilation threads. It checks whether the generated code fits the state grid or not. If it does, the code is updated. If it doesn’t, the new optimized code is ignored and the computation thread continues with the original code.

**CHAPTER 4**

**IMPLEMENTATION**

#include<stdio.h>

#include<math.h>

#include<time.h>

#define PI 3.1415

//for time calculations

clock\_t start, end;

double cpu\_time\_used;

//Euler

//y(n+1) = y(n) + h\*f(x,y)

//where f(x,y) = x+y

void euler()

{

float x[6],y[6];

x[0]=0.0;

//initializing the x array

for(int i=1;i<6;i++)

{

x[i]=x[i-1]+0.2;

}

y[0]=1.0;

float h=0.2; // where h is the step division

float f[5]; // array to store the function values

printf("Values are \n");

printf("%f\n",y[0]);

for(int i=0;i<5;i++)

{

f[i]=x[i]+y[i]; // stores the addition of (x+y)

y[i+1]=y[i]+(h\*f[i]); //computes the y value

printf("%f\n",y[i+1]);

}

}

//dynamic

void eulermodified()

{

float x[6],y[6][4];

x[0]=0.0;

for(int i=1,j=0;i<6,j<4;i++,j++)

{

x[i]=x[i-1]+0.2; //initializing x array

y[0][j]=1.0; //initializing the first row of the y array

}

float h=0.2;

//printing the first row of the array

for(int i=0;i<4;i++)

{

printf("%f ",y[0][i]);

}

printf("\n");

for(int i=1;i<6;i++)

{

y[i][0] = y[i-1][3]+h\*(x[i-1]+y[i-1][3]);

printf("%f ",y[i][0]);

y[i][1] = y[i-1][3]+(h/2)\*((x[i-1]+y[i-1][3])+(x[i]+y[i][0]));

printf("%f ",y[i][1]);

y[i][2] = y[i-1][3]+(h/2)\*((x[i-1]+y[i-1][3])+(x[i]+y[i][1]));

printf("%f ",y[i][2]);

y[i][3] = y[i-1][3]+(h/2)\*((x[i-1]+y[i-1][3])+(x[i]+y[i][2]));

printf("%f ",y[i][3]);

printf("\n");

}

}

void eulermodified2()

{

float x[6],y[6][4];

x[0]=0.0;

for(int i=1,j=0;i<6,j<4;i++,j++)

{

x[i]=x[i-1]+0.2;

y[0][j]=1.0;

}

float h=0.2;

for(int i=1;i<6;i++)

{

y[i][0]=y[i-1][3]+h\*(x[i-1]+y[i-1][3]);

y[i][1]=y[i-1][3]+(h/2)\*((x[i-1]+y[i-1][3])+(x[i]+y[i][0]));

y[i][2]=y[i-1][3]+(h/2)\*((x[i-1]+y[i-1][3])+(x[i]+y[i][1]));

y[i][3]=y[i-1][3]+(h/2)\*((x[i-1]+y[i-1][3])+(x[i]+y[i][2]));

}

}

//calculates the area of a circle

void Area\_of\_Circle(int radius)

{

float area=PI\*radius\*radius;

printf("\n\nArea of the circle is : %f\n",area);

}

//calculate the Arithmetic\_Progression

//using the first term of series is a and common difference is d

void Arithmetic\_Progression (int a, int d, int n)

{

int ai=a;

int sum=0;

for(int i=1;i<=n;i++)

{

sum=sum+ai;

ai=ai+d;

}

printf("Sum of the AP is : %d\n",sum);

}

//calculate the Optimized Arithmetic Progression

void Optimized\_Arithmetic\_Progression(int a, int d, float n)

{

int Optimized\_sum;

Optimized\_sum=(n/2)\*((2\*a)+((n-1)\*d));

printf("Sum of the AP is (using approximation) : %d\n",Optimized\_sum);

}

int main()

{

int count=0; // Counts iterations simulating code generator

double stateGrid[3]; //Simulates state Grid.

int flag1=0; // Flag variables simulate uncovering of approximation methods

int flag2=0; // in order to optimize given cell in state grid.

int i;

//Outer Loop simulates continuous code generation of scanner that selects which parts of code to optimize from polyhedral..

printf("\nThe State Grid representation (time taken for each computation):");

for(i=0;i<3;i++)

{

stateGrid[i]=0;

printf(" \t %f|",stateGrid[i]);

}

while(count!=2)

{

//circle

start = clock();

Area\_of\_Circle(5);

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

stateGrid[0]=cpu\_time\_used;

printf("Time used for computation : %f\n\n\n",stateGrid[0]);

//Oap

if(flag1==1)

{

start = clock();

Optimized\_Arithmetic\_Progression(10,5,15);

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

stateGrid[1]=cpu\_time\_used;

printf("Time used for computation : %f\n\n\n",stateGrid[1]);

}

//Arithmetic Progression

if(flag1==0)

{

start = clock();

Arithmetic\_Progression(10,5,15);

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

stateGrid[1]=cpu\_time\_used;

printf("Time used for computation : %f\n\n\n",stateGrid[1]);

flag1=1;

}

//Euler Modified

if(flag2==1)

{

start = clock();

eulermodified2();

end = clock();

eulermodified();

cpu\_time\_used = (((double) (end - start)) / CLOCKS\_PER\_SEC)\*10000;

stateGrid[2]=cpu\_time\_used;

printf("Time used for computation : %f\n\n\n",stateGrid[2]);

}

//Euler

if(flag2==0)

{

start = clock();

euler();

end = clock();

cpu\_time\_used = (((double) (end - start)) / CLOCKS\_PER\_SEC)\*10000;

stateGrid[2]=cpu\_time\_used;

printf("Time used for computation : %f\n\n\n",stateGrid[2]);

flag2=1;

}

printf("The State Grid representation (time taken for each computation):");

for(i=0;i<3;i++)

{

printf(" \t %f|",stateGrid[i]);

}

count++;

}

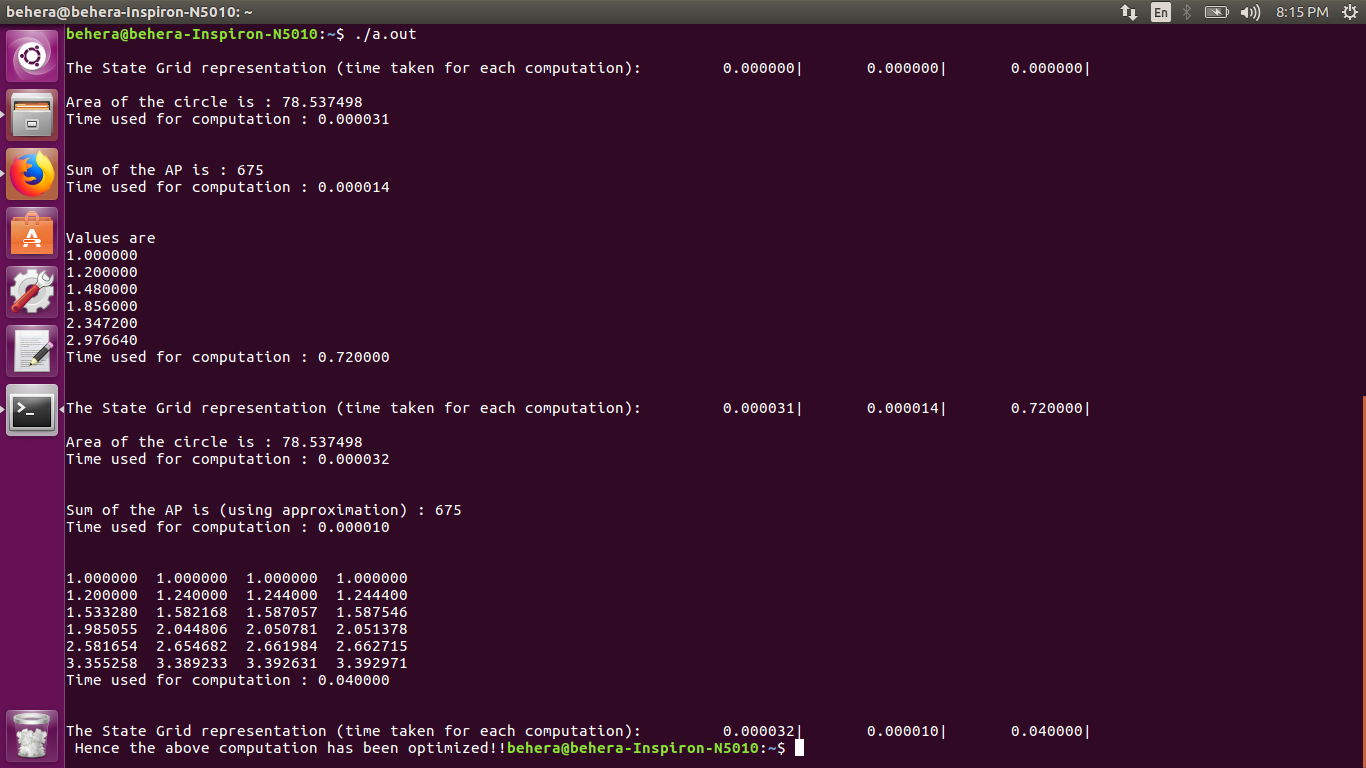
printf("\n Hence the above computation has been optimized!!");

return 0;

}

**CHAPTER 5**

**RESULTS AND DISCUSSION**



The above snapshot shows the output of the implemented code. It is observed that the computation time for three functions is displayed thrice. In the first iteration, all the functions are analyzed and various decisions are taken care of. Issues related to code modification, code alteration, approximation of the functions which improve the performance are looked after. The second iteration executes the functions and displays the computation time for the original codes. The third iteration executes the same functions but with an alternative approach (approximation) and displays the computation time for the same. As it is seen, the execution of the approximated methods shows a significant decrease in the computation time. On similar terms, the algorithm scans through the functions in each iteration, modifies and optimizes the code in order to reduce the computation time.

**CHAPTER 6**

**CONCLUSION**

This paper provides a basic understanding and a view towards the implementation of ACR (Adaptive Code Refinement) [1]. The three different functions namely area of a circle (static), dynamic arithmetic progression and Euler’s method calculations demonstrate the various scenarios the ACR can come across. The first function is a static function, it doesn’t need an alternative. While the other two functions are dynamic in nature. These functions need optimization. The same has been implemented. As it can be seen from the results, the performance is being improved alongside reducing the time complexity. Hence, Adaptive Code Refinement can be used in order to improve the performance of a system.

**CHAPTER 7**

**REFERENCES**

1. Maxime Schmitt, César Sabater, and Cédric Bastoul: “Semi-automatic Generation Of Adaptive Codes”. Stockholm, Sweden, Jan. 2017.
2. S. Campanoni, G. Holloway, G.-Y. Wei, and D. M. Brooks. “Helix-up: Relaxing Program Semantics To Unleash Parallelization”. San Francisco, USA, Feb. 2015.
3. H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. “Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures”. Technical Report MIT-CSAIL-TR-2009-042, MIT, Sept. 2009.
4. W. Baek and T. M. Chilimbi. Green: “A Framework for Supporting Energy-Conscious Programming using Controlled Approximation”. In PLDI’10 Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 198–209, Toronto, Canada, June 2010.
5. Cédric Bastoul. “Code Generation in the Polyhedral Model Is Easier Than You Think”. 2004, IEEE Computer Society, pp.7–16, 2004, 29 September - 3 October 2004, Antibes Juan-les-Pins, France.
6. Maxime Schmitt, Philippe Helluy, Cédric Bastoul. Adaptive Code Refinement: “A Compiler Technique and Extensions to Generate Self-Tuning Applications”.
7. Information regarding arithmetic progression, calculation of the sum of the arithmetic series. <https://en.wikipedia.org/wiki/Arithmetic_progression>.
8. Information regarding Modified Euler’s method. <https://mat.iitm.ac.in/home/sryedida/public_html/caimna/ode/euler/ie.html>
9. Information regarding Euler’s method. <http://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx>