UE18CS390B – Capstone Project Phase – 2

SEMESTER - VII

END SEMESTER ASSESSMENT

Project Title : Automated Parallelization of Source Code Using Program Comprehension

Project ID : PW22NSK03

Project Guide : Prof. N S Kumar

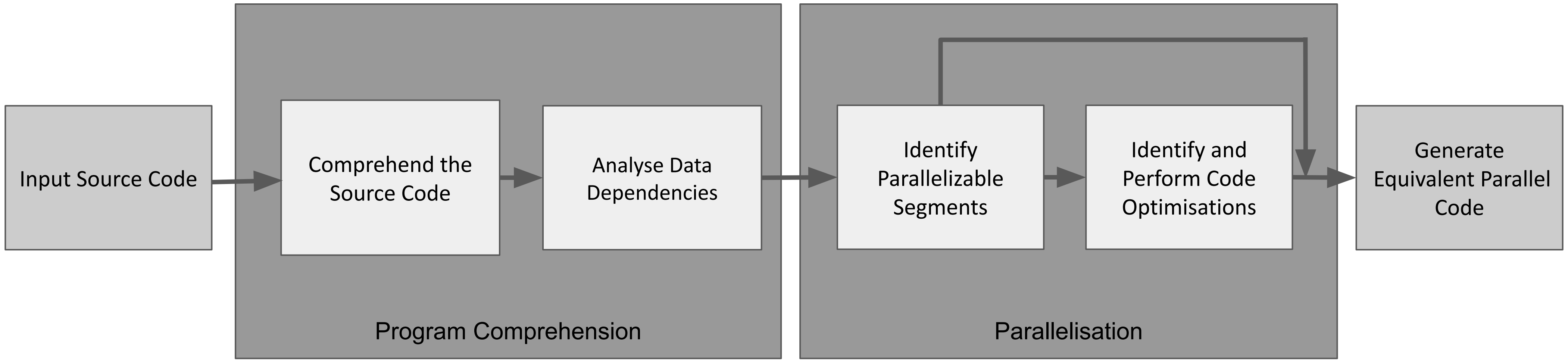
Project Team : Darshan D (PES1201801456)
Karan Kumar G (PES1201801883)
Manu M Bhat (PES1201801452)
Mayur P L (PES1201801439)

Outline

Abstract

Automated Parallelization of Source Code using Program Comprehension:

- Generate the parallel equivalent of given sequential source code automatically.
- Maximise the utilization of the available computational resources.


Input Source Code
Comprehend the Source Code
Analyse Data Dependencies
Program Comprehension
Identify Parallelizable Segments
Identify and Perform Code Optimisations
Parallelisation
Generate Equivalent Parallel Code

Abstract

- Hardware has been improving at a rapid pace recently:
  - Multi - threaded systems
  - Multi - processor systems
  - Multi - core systems

- Performance gain is limited by software and programs written

- Sequential programs only exploit the clock speed improvements

Abstract

- Parallel computing enables:
    - Efficient use of available hardware
    - Faster execution
    - Better cost-effectiveness

- Problems with Parallel Coding:
    - Requires highly skilled programmers
    - Requires additional development time and increases cost
    - Requires maintaining correctness of code
    - Increases testing and debugging complexity

Abstract

- Auto parallelization techniques help in mitigating costs incurred by manual parallelization

- Scope of current tools covers only loops and other minor optimisations

- Research and goal of the project
  - Enable parallelisation for entire programs
  - Support for a wide variety of programs
  - Achieve maximum possible parallelism

Outline

Literature and Product Survey

We present the following points regarding the forthcoming literature and product survey:

- The Paper title, authors and year of publication
- A brief introduction and explanation of the paper/product
- The Pros concerning the paper/product
- The Cons concerning the paper/product
- The availability of a tool/implementation
- Relation of the paper/product to our Capstone Project

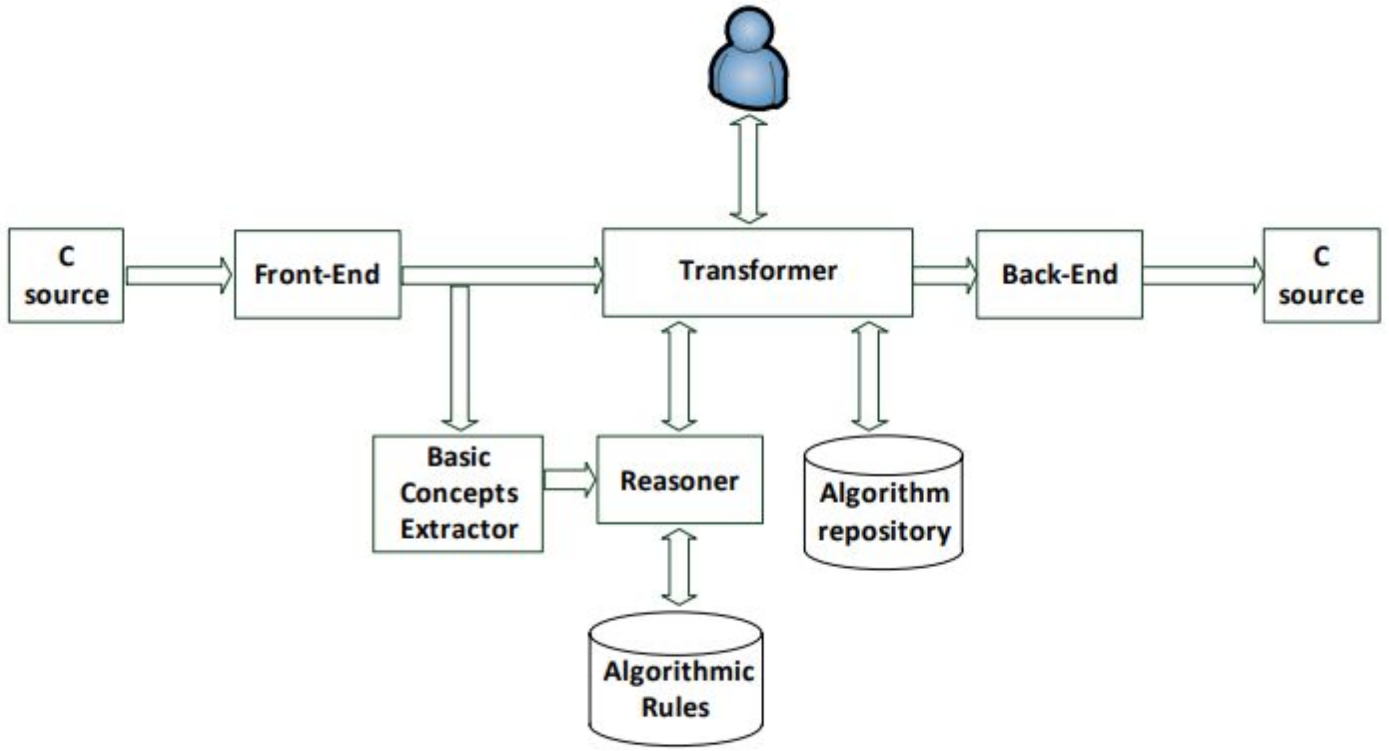
Literature and Product Survey

1. Cantiello, P., & Di Martino, B. (2012). Automatic Source Code Transformation for GPUs Based on Program Comprehension

   - Performs program comprehension using PAP Recognizer (Static Analyser)
   - Implements an “Extractor” based on Prolog facts to identify algorithms (paradigms)
   - Modifies the program’s AST to add necessary sub-tree with parallel version of code

Literature and Product Survey

1. Cantiello, P., & Di Martino, B. (2012). Automatic Source Code Transformation for GPUs Based on Program Comprehension



Working pipeline of method proposed

Literature and Product Survey

1. Cantiello, P., & Di Martino, B. (2012). Automatic Source Code Transformation for GPUs Based on Program Comprehension

   Pros:

   - Unique approach to utilise Program Comprehension to assimilate code
   - Effective approach to use AST to handle section-wise code
   - Flexible algorithm recognition with rule-based matching

   Cons:

   - Difficult generalisation for all algorithms, requires for extensive rule writing
   - Long execution time for Recognition phase, implies scalability issues

Literature and Product Survey

1. Cantiello, P., & Di Martino, B. (2012). Automatic Source Code Transformation for GPUs Based on Program Comprehension

   Tool availability:

   - Tool built according to authors, but not publicly available

   Relation to our work:

   - Paper provides basis for our approach to algorithm recognition (Program comprehension)

Literature and Product Survey

2. Martino, B. D., & Iannello, G. (n.d.) (1991). Towards automated code parallelization through program comprehension

- Presents Program comprehension as a “Concept Assigning Problem”

- Defines two “Programming Paradigms”:
    - Tree computation: Problems which can be divided into a representation of parents-children tasks
    - Master-Worker based: Problems represented to have a Master task instructing other worker tasks

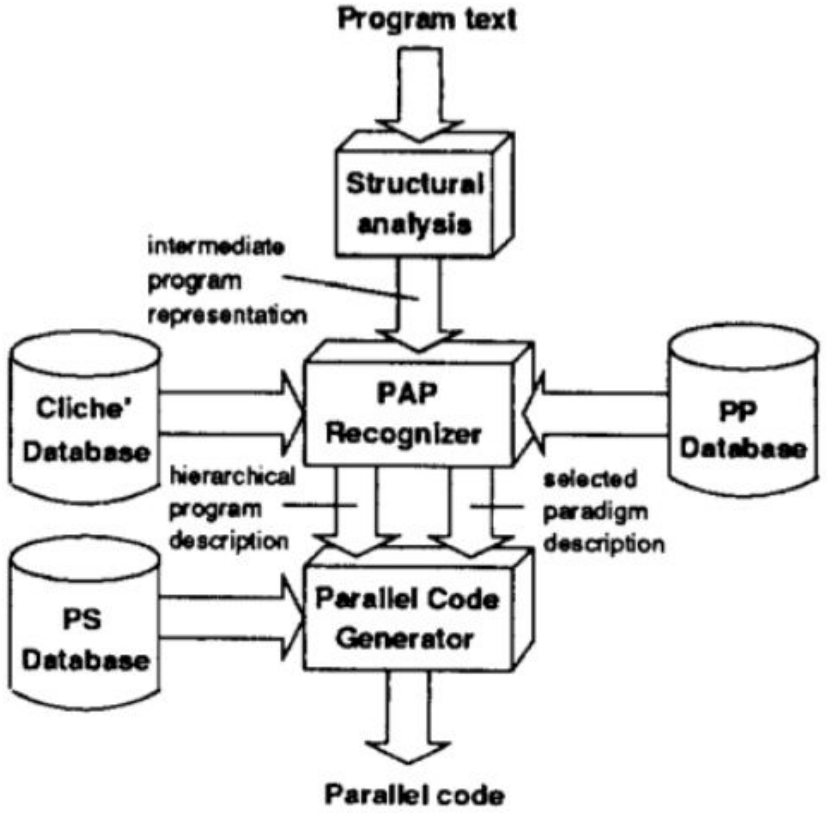
Literature and Product Survey

2. Martino, B. D., & Iannello, G. (n.d.) (1991). Towards automated code parallelization through program comprehension

- Recognizes the paradigm by the use of concept called “cliches”
- Defines the Parallel Skeleton code for the selected paradigm, required for replacement

Literature and Product Survey

2. Martino, B. D., & Iannello, G. (n.d.) (1991). Towards automated code parallelization through program comprehension



Working pipeline of method proposed

Literature and Product Survey

2. Martino, B. D., & Iannello, G. (n.d.) (1991). Towards automated code parallelization through program comprehension

Pros:

- Method aims to generalise process of parallelization for all kinds of programs
- Scalable with addition of pre-defined “Programming Paradigms”

Cons:

- Method might not work for a large number of algorithms
- Requires creating and updating databases to support additional paradigms

Literature and Product Survey

2. Martino, B. D., & Iannello, G. (n.d.) (1991). Towards automated code parallelization through program comprehension

Tool availability:

- Method proposed is only theoretical in nature, no tool exists

Relation to our work:

- Paper provides basis for our approach to algorithm recognition (Program Comprehension)
- Concept of paradigms and cliches helps us develop on a generalised approach of parallelization

Literature and Product Survey

3. Uday Bondhugula, J. Ramanujam, P. Sadayappan (2007). PLuTo: A Practical and Fully Automatic Polyhedral Program Optimization System

- Implements a S2S compiler that performs loop parallelisation
- Uses the concept of Polyhedral modelling
- Applies transformations based on dependencies on Affine Loops
- Performs various other transformation techniques on Non-affine Loops

Literature and Product Survey

3. Uday Bondhugula, J. Ramanujam, P. Sadayappan (2007). PLuTo: A Practical and Fully Automatic Polyhedral Program Optimization System

   Pros :

   - Correctness of transformed program is verified mathematically
   - Accuracy with respect to loop parallelisation and optimisations is high

   Cons :

   - Installation process is quite cumbersome
   - Expensive nature of Integer Linear programming for Polyhedral modeling
   - Limited nature of loop parallelisation

Literature and Product Survey

3. Uday Bondhugula, J. Ramanujam, P. Sadayappan (2007). PLuTo: A Practical and Fully Automatic Polyhedral Program Optimization System

Tool Availability and implementation :

- Available as an open-source tool
- All major loop parallelisation techniques are implemented

Relation to our work :

- Tool performs loop-based parallelization and optimization which can be integrated with our task-level parallelism to increase generality

Literature and Product Survey

4. ParaWise – Widening Accessibility to Efficient and Scalable Parallel Code (White Paper)

- Implementation not available in detail since tool is commercial and paid
- Customisation of type of parallelisation achieved
- Usage of OpenMP directives at appropriate positions using code analysis
- Message Passing optimizations

Literature and Product Survey

4. ParaWise – Widening Accessibility to Efficient and Scalable Parallel Code (White Paper)

Pros :

- Designed for different end users, i.e expert, non-expert and serial code users
- Provides state of the art features to enable parallelization

Cons :

- Requires user intervention to choose parallelisation settings, hence not fully automated
- Does not cover all possibilities of parallelisation possible

Literature and Product Survey

4. ParaWise – Widening Accessibility to Efficient and Scalable Parallel Code (White Paper)

Tool Availability and implementation :

- Commercial tool, not available freely
- Analysis of requirements of users in the domain of HPC and accordingly design their product

Relation to our work :

- Provides for a reference point with respect to the possible parallelisation of a program

Literature and Product Survey

5. Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav. (2019). code2vec: Learning Distributed Representations of Code

   - Converts source code to vector embeddings representation using a neural model
   - Represents source code by capturing the meaning, intent and structure
   - Converts code to its AST initially, extracting path-based representations
   - Captures relative importance of sections of code and combines importance metrics using a neural attention model - enables identifying subtle differences

Literature and Product Survey

5. Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav. (2019). code2vec: Learning Distributed Representations of Code

Pros :

- SOTA model to obtain numerical representations of source code
- Neural attention model produces different vector embeddings for similar programs, capturing the subtle differences

Cons :

- Requires large dataset to train the model to obtain decent results
- Applications such as code labelling might not be fully accurate due to lack of categories represented in an inadequate dataset

Literature and Product Survey

5. Uri Alon, Meital Zilberstein, Omer Levy, Eran Yahav. (2019). code2vec: Learning Distributed Representations of Code

   Tool availability:

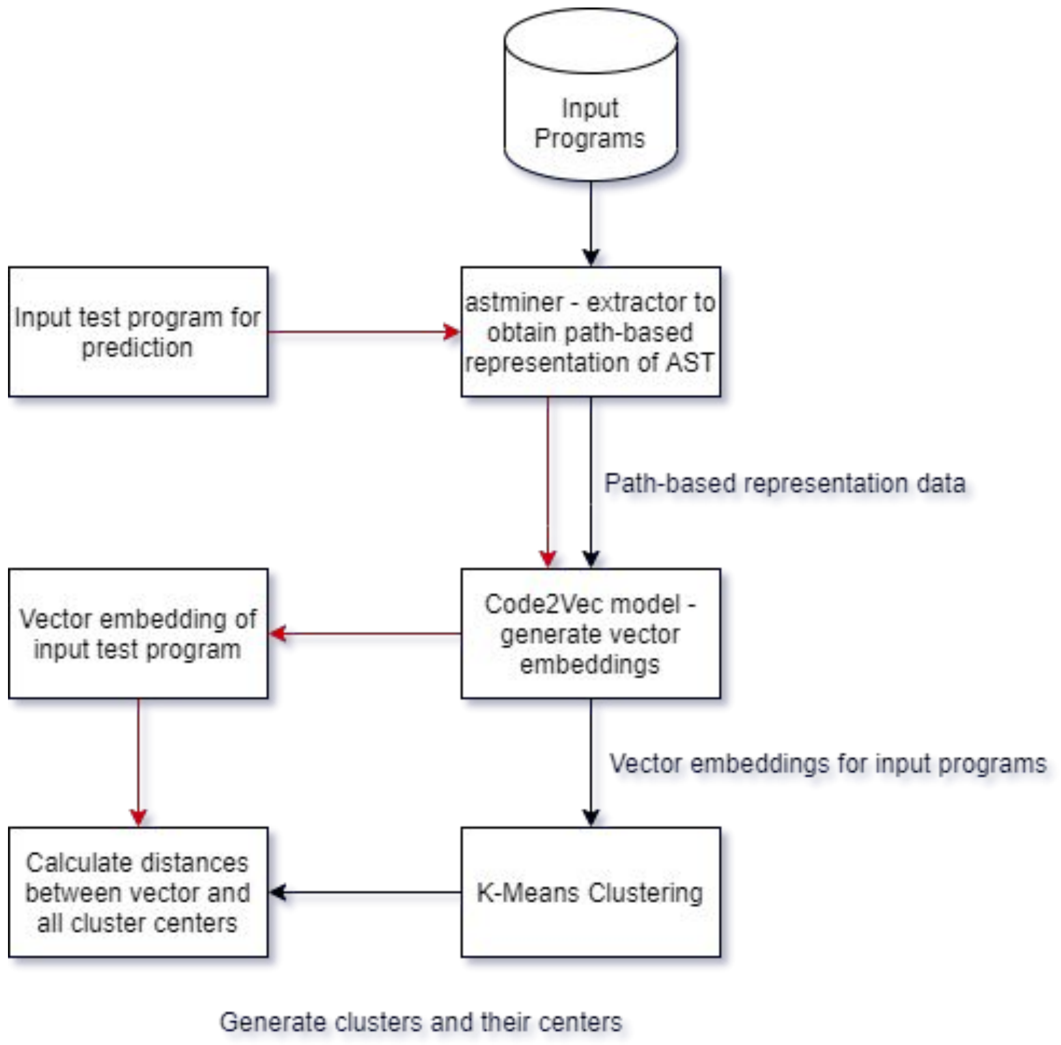
   - Web applications available to test out any programs and check their labelling at code2vec.org

Relation to our work:

- Vector embeddings obtained from model helps us in grouping similar vectors together and perform program comprehension

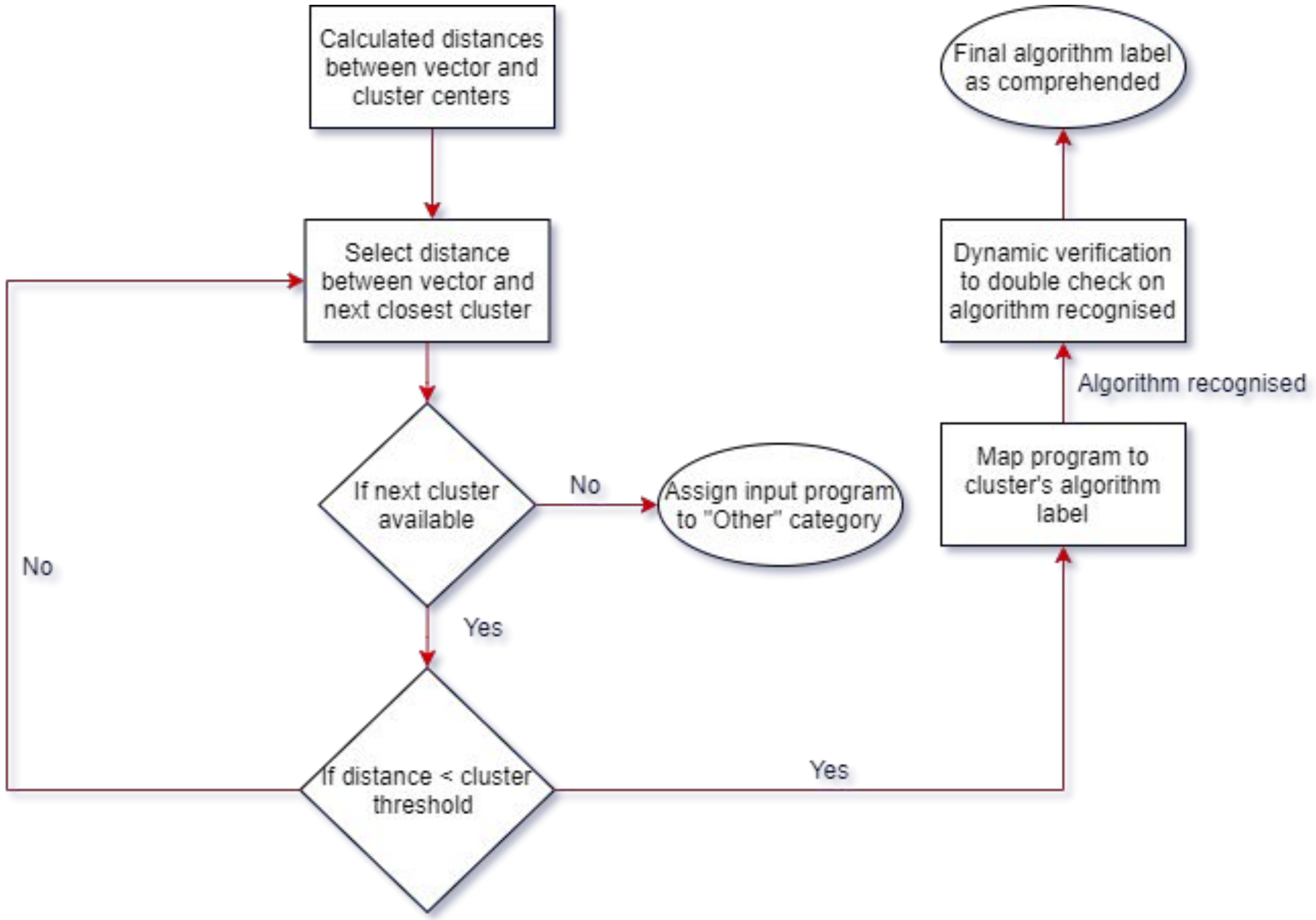
Outline

Program Comprehension Implementation


Input Programs
astminer - extractor to obtain path-based representation of AST
Input test program for prediction
Path-based representation data
Code2Vec model - generate vector embeddings
Vector embedding of input test program
Vector embeddings for input programs
K-Means Clustering
Calculate distances between vector and all cluster centers
Generate clusters and their centers

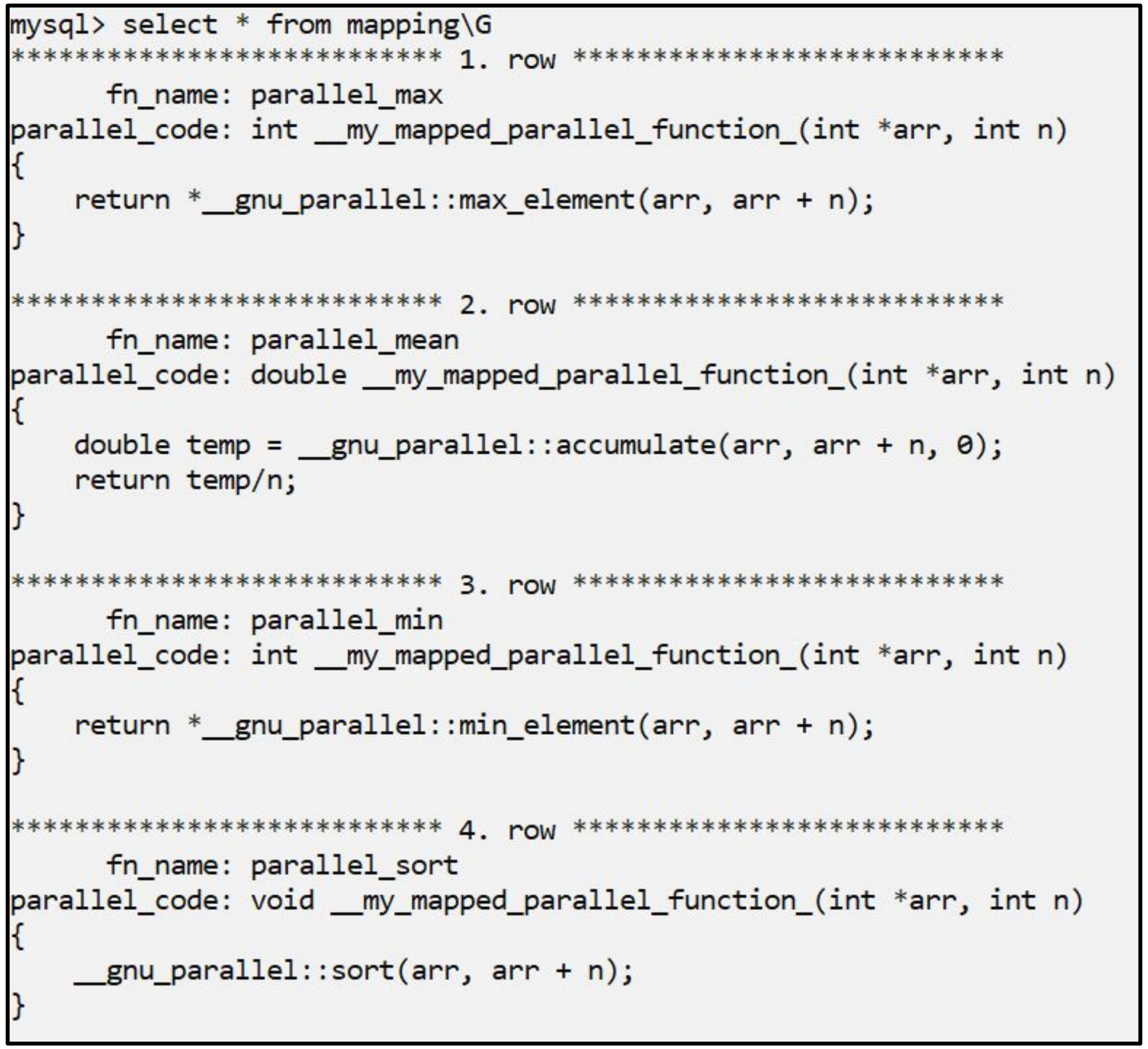
→	Flow of Input Program for Training
→	Flow of Input Test Program for Prediction

Program Comprehension Implementation


Calculated distances between vector and cluster centers
Select distance between vector and next closest cluster
If next cluster available
No
Assign input program to "Other" category
Yes
If distance < cluster threshold
No
Yes
Map program to cluster's algorithm label
Algorithm recognised
Dynamic verification to double check on algorithm recognised
Final algorithm label as comprehended

→	Flow of Input Program for Training
→	Flow of Input Test Program for Prediction

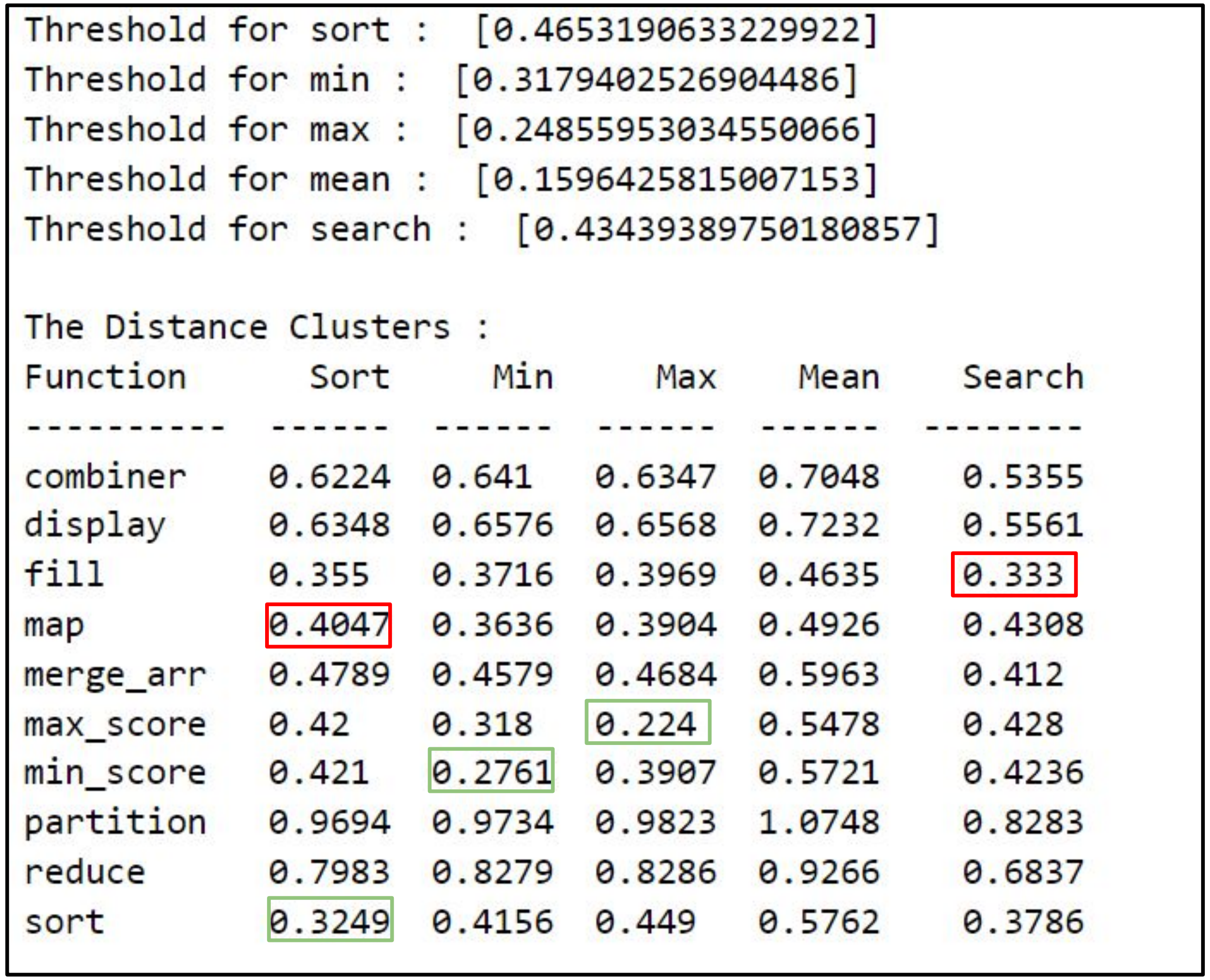
Program Comprehension Implementation



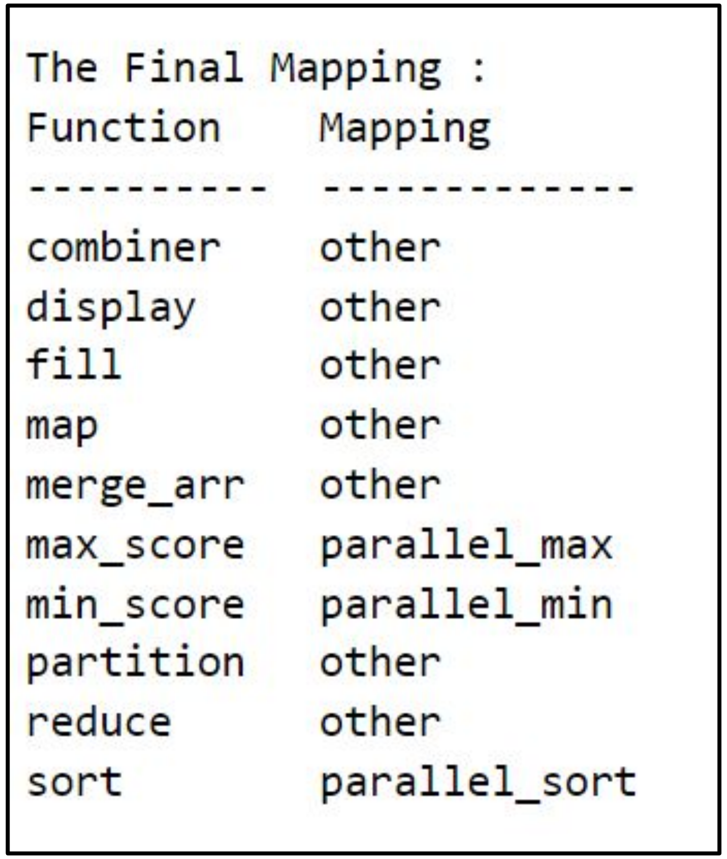
```
mysql> select * from mapping\G
*************************** 1. row ***************************
      fn_name: parallel_max
parallel_code: int __my_mapped_parallel_function_(int *arr, int n)
{
    return *__gnu_parallel::max_element(arr, arr + n);
}

*************************** 2. row ***************************
      fn_name: parallel_mean
parallel_code: double __my_mapped_parallel_function_(int *arr, int n)
{
    double temp = __gnu_parallel::accumulate(arr, arr + n, 0);
    return temp/n;
}

*************************** 3. row ***************************
      fn_name: parallel_min
parallel_code: int __my_mapped_parallel_function_(int *arr, int n)
{
    return *__gnu_parallel::min_element(arr, arr + n);
}

*************************** 4. row ***************************
      fn_name: parallel_sort
parallel_code: void __my_mapped_parallel_function_(int *arr, int n)
{
    __gnu_parallel::sort(arr, arr + n);
}
```

A few rows in the “algorithm label to parallel code” mapping database

Program Comprehension Implementation



```
Threshold for sort :  [0.4653190633229922]
Threshold for min :  [0.3179402526904486]
Threshold for max :  [0.2485595303455006]
Threshold for mean :  [0.1596425815007153]
Threshold for search :  [0.4343938975018085]

The Distance Clusters :
Function        Sort     Min     Max     Mean     Search
----------     ------   ------  ------  ------   --------
combiner       0.6224   0.641   0.6347  0.7048    0.5355
display        0.6348   0.6576  0.6568  0.7232    0.5561
fill           0.355    0.3716  0.3969  0.4635    0.333
map            0.4047   0.3636  0.3904  0.4926    0.4308
merge_arr      0.4789   0.4579  0.4684  0.5963    0.412
max_score      0.42     0.318   0.224   0.5478    0.428
min_score      0.421    0.2761  0.3907  0.5721    0.4236
partition      0.9694   0.9734  0.9823  1.0748    0.8283
reduce         0.7983   0.8279  0.8286  0.9266    0.6837
sort           0.3249   0.4156  0.449   0.5762    0.3786
```



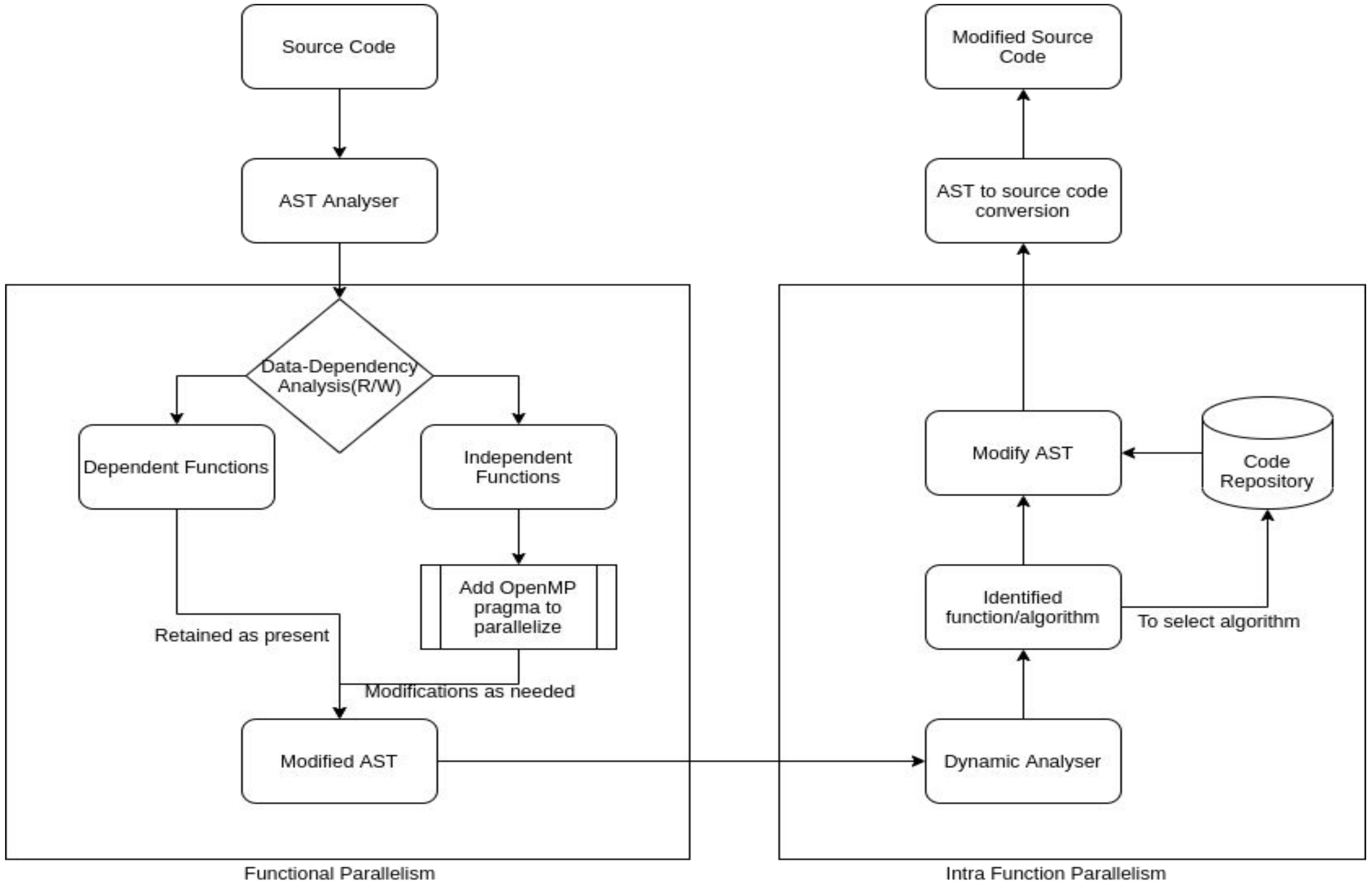
```
The Final Mapping :
Function     Mapping
----------   -------------
combiner     other
display      other
fill         other
map          other
merge_arr    other
max_score    parallel_max
min_score    parallel_min
partition    other
reduce       other
sort         parallel_sort
```

Output of Program Comprehension phase on a Map Reduce Program

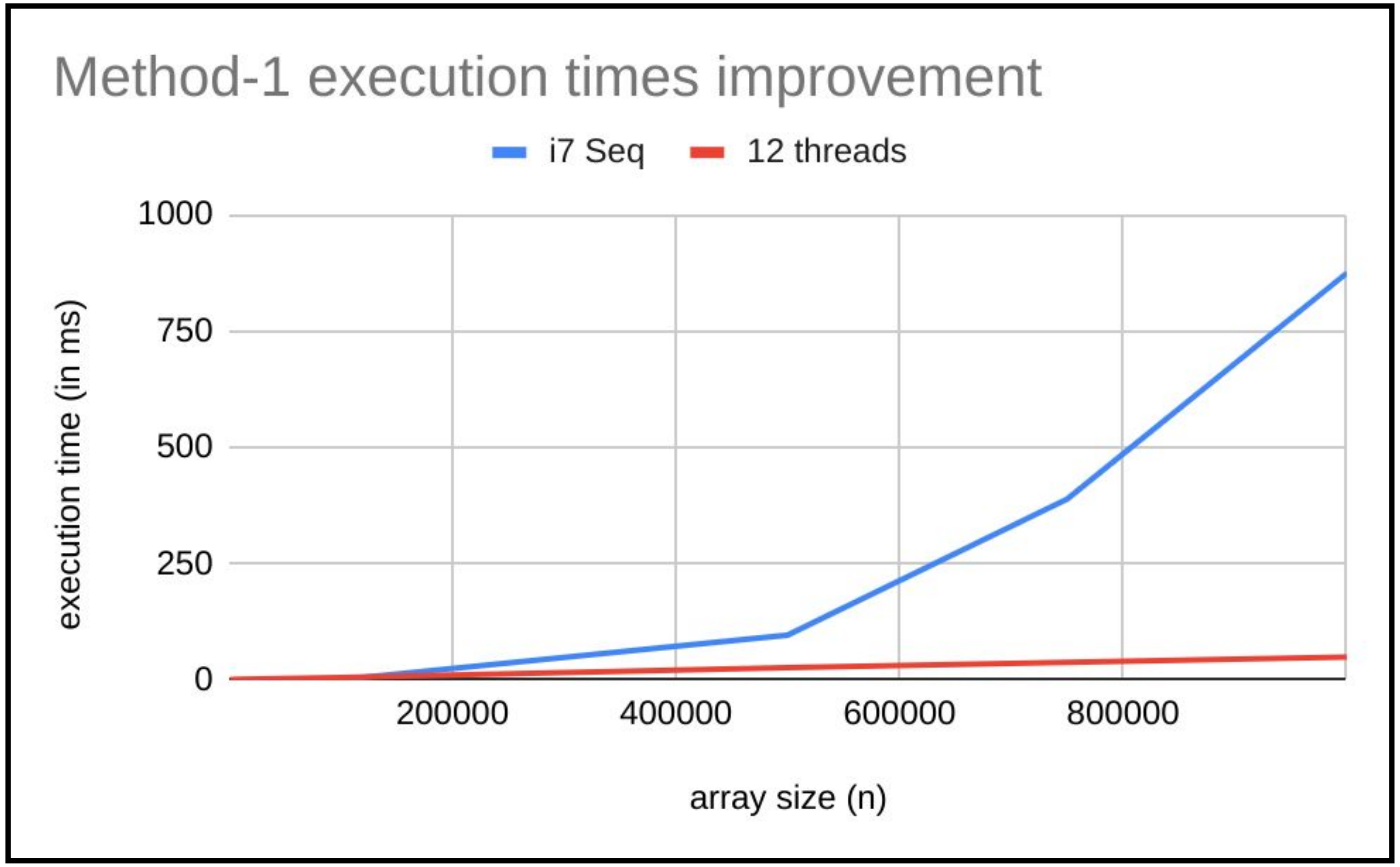
Outline

- Abstract
- Literature and Product Survey
- Implementation Details of Program Comprehension Phase
- **Implementation Details of Parallelization Phase:**
  - ➡ **Method 1: Parallelism by AST Querying & OpenMP Directives**
  - ➡ Method 2: Naive Thread Scheduling using C++ concepts of Promises and Futures
  - ➡ Method 3: Master-Worker based Optimised Thread Scheduling
  - ➡ Method 4: Non Master-Worker based Optimised Thread Scheduling
- Project Demonstration
- Technologies Used
- Documentation
- Team Roles and Responsibilities
- Lessons Learnt
- Conclusion and Future Work
- References

Parallelization Implementation - Method 1



Method 1 Results



Comparison of execution times for Sequential vs Parallel environments

Method 1 Drawbacks

- Example program for which Method-1 fails:

Input:

```
fn_A(arr1, n)
fn_B(arr1, n)
fn_C(arr1, n)
fn_A(arr2, n)
fn_B(arr2, n)
fn_C(arr2, n)
```

Output:

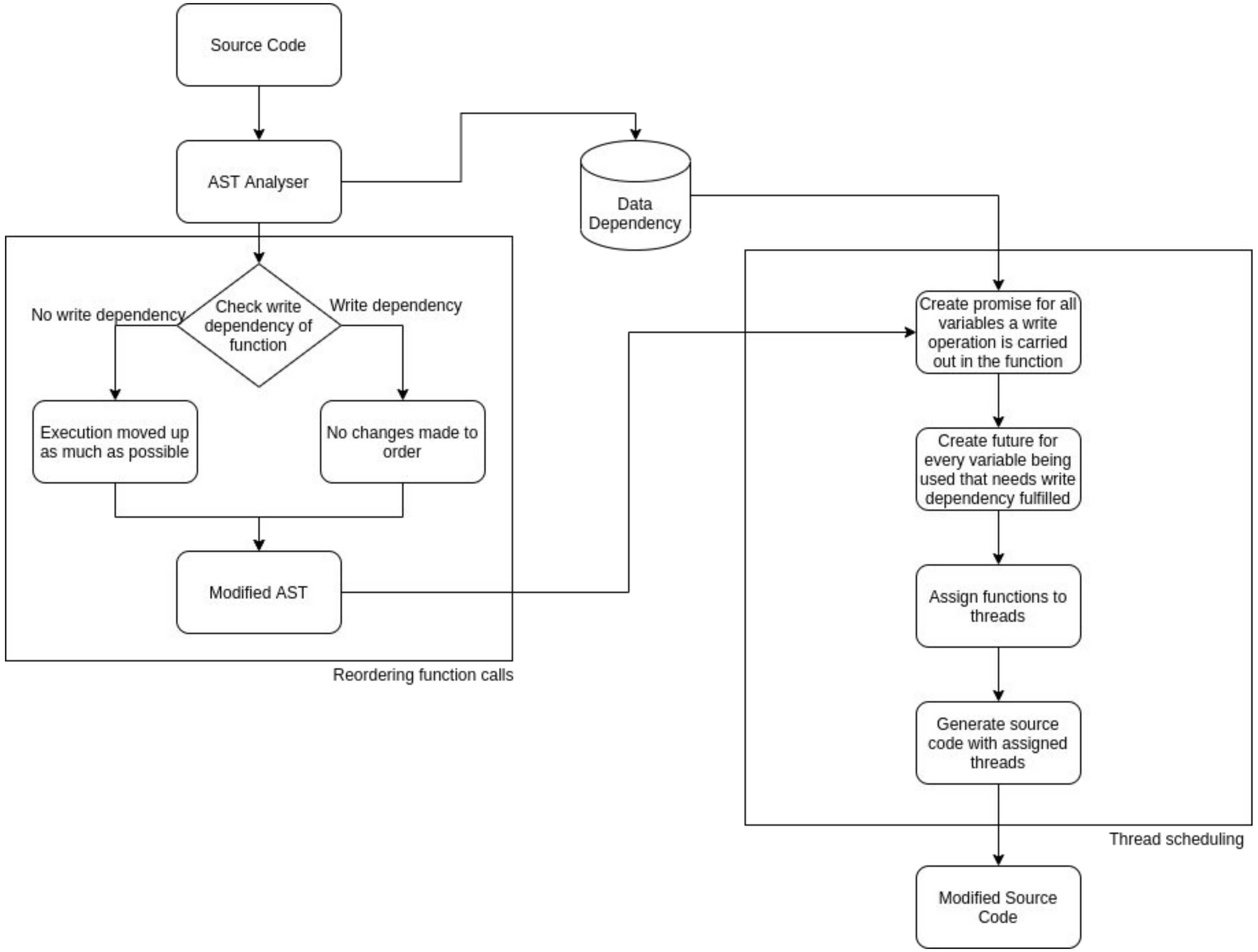
```
fn_A(arr1, n);
  #pragma omp parallel sections
  {
    #pragma omp section
    fn_B(arr1, n);
    #pragma omp section
    fn_C(arr1, n,);
    fn_A(arr2, n);
    #pragma omp section
    fn_B(arr2, n);
    #pragma omp section
    fn_C(arr2, n);
  }
```

Method 1 Drawbacks

- Limited degree of parallelism
- Failed to support grouping of function under one section for OpenMP pragma
- Fine control of execution not possible, leading to Method-2

Outline

Parallelization Implementation - Method 2


Source Code
AST Analyser
Data Dependency
No write dependency
Check write dependency of function
Write dependency
Execution moved up as much as possible
No changes made to order
Modified AST
Reordering function calls
Create promise for all variables a write operation is carried out in the function
Create future for every variable being used that needs write dependency fulfilled
Assign functions to threads
Generate source code with assigned threads
Thread scheduling
Modified Source Code

Method 2 Drawbacks

- Method-2 is able to handle the test case which failed in Method-1, but it is not completely optimised.

Input function calls:

```
fn_A(arr1, n)
fn_B(arr1, n)
fn_C(arr1, n)
fn_A(arr2, n)
fn_B(arr2, n)
fn_C(arr2, n)
```

Output:

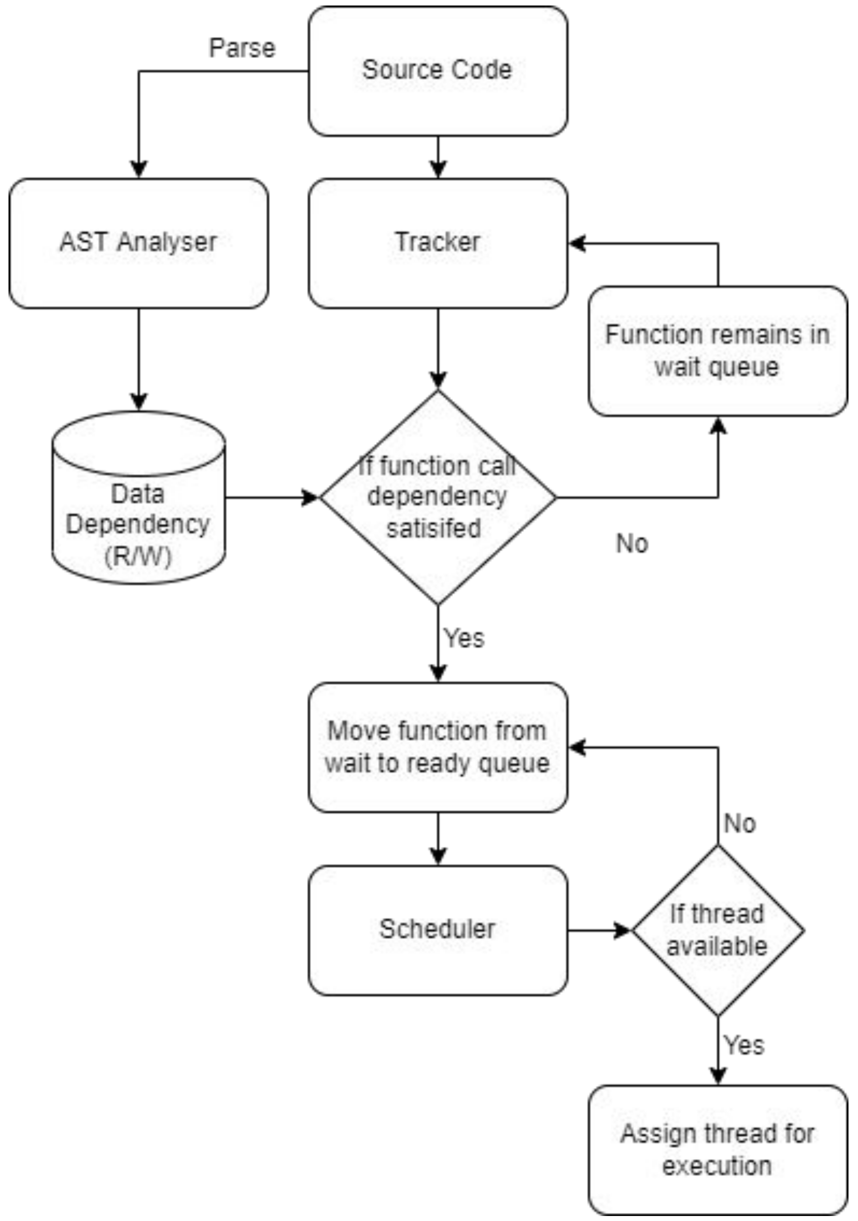
```
std::promise<void> p_arr1_0;
thread t1(fn_A, arr1, n, p_arr1_0);
std::promise<void> p_arr2_0;
thread t2(fn_A, arr2, n, p_arr2_0);
std::future<void> f_arr1_1= p_arr1_0.get_future().wait();
thread t3(fn_B, arr1, n);
thread t4(fn_C, arr1, n);
std::future<void> f_arr2_1= p_arr2_0.get_future().wait();
thread t5(fn_B, arr2, n);
thread t6(fn_C, arr2, n);
```

Method 2 Drawbacks

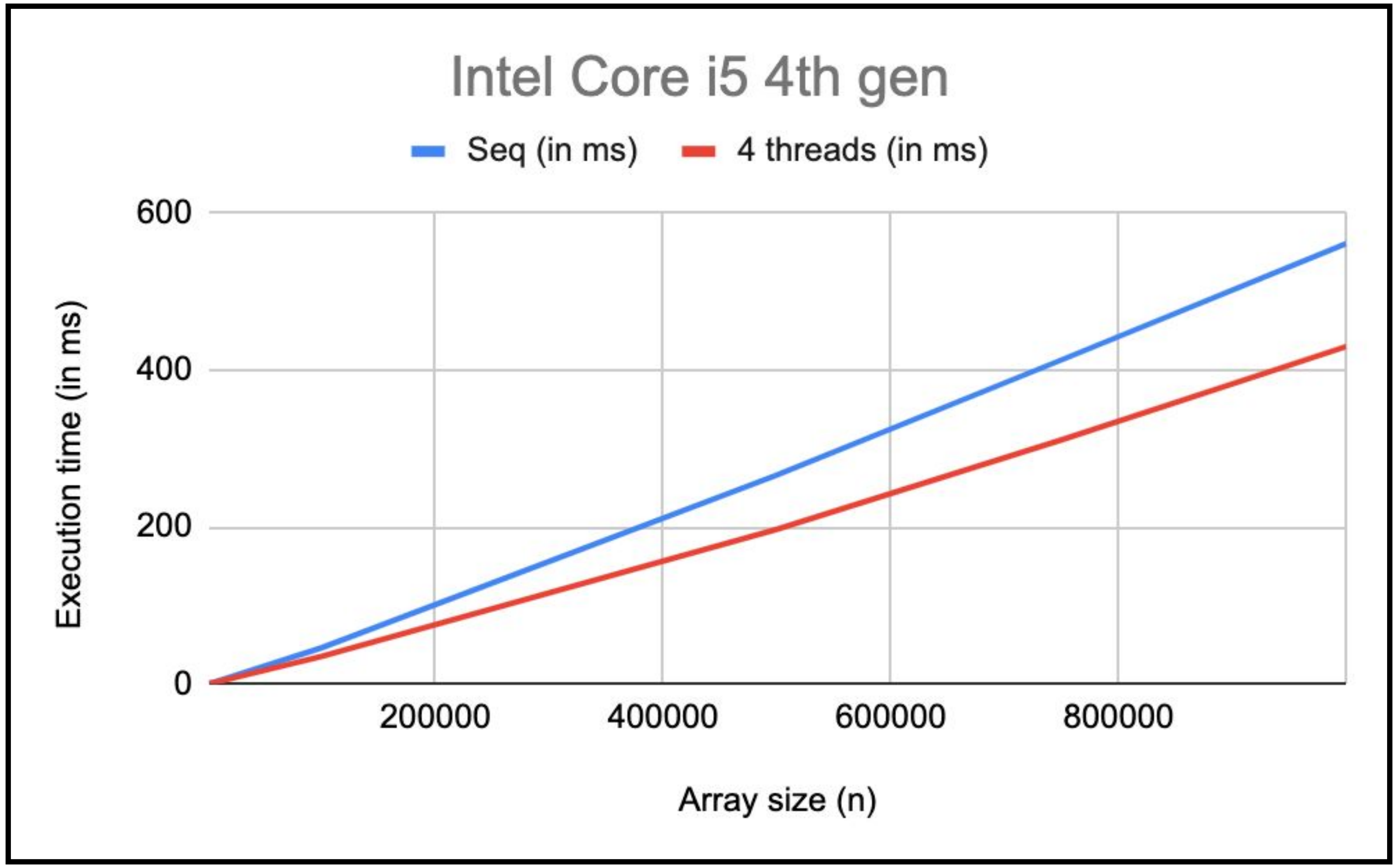
- Failed to support grouping of functions in a generalised manner
- Increased execution time for certain cases as discussed
- Required more fine grained control of execution for achieving better parallelism

Outline

Parallelization Implementation - Method 3

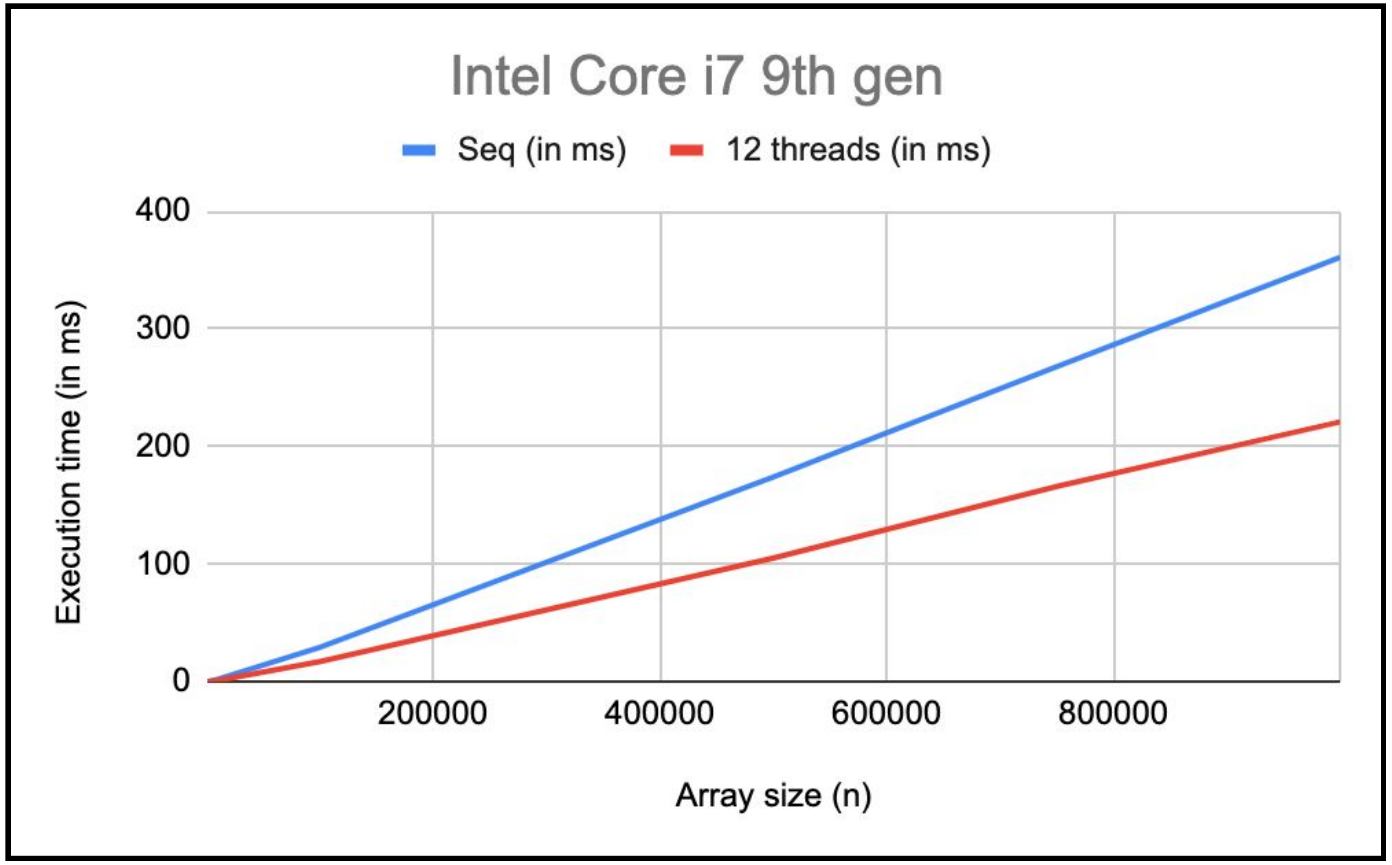

Parse
Source Code
AST Analyser
Tracker
Function remains in wait queue
Data Dependency (R/W)
If function call dependency satisifed
No
Yes
Move function from wait to ready queue
No
Scheduler
If thread available
Yes
Assign thread for execution

Method 3 Results



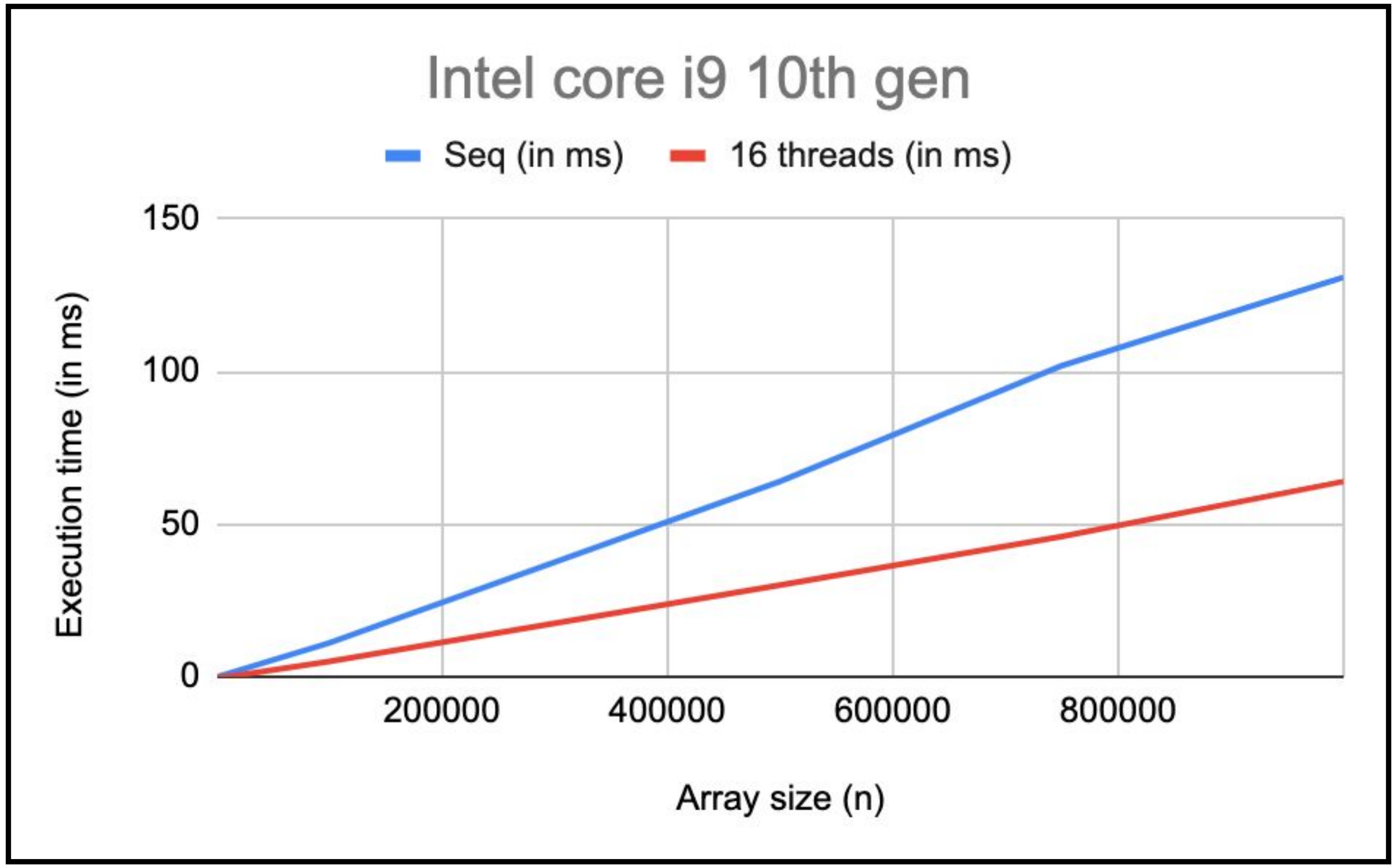
Comparison of execution times for Sequential vs Parallel environments (only inter-function parallelism)

Method 3 Results



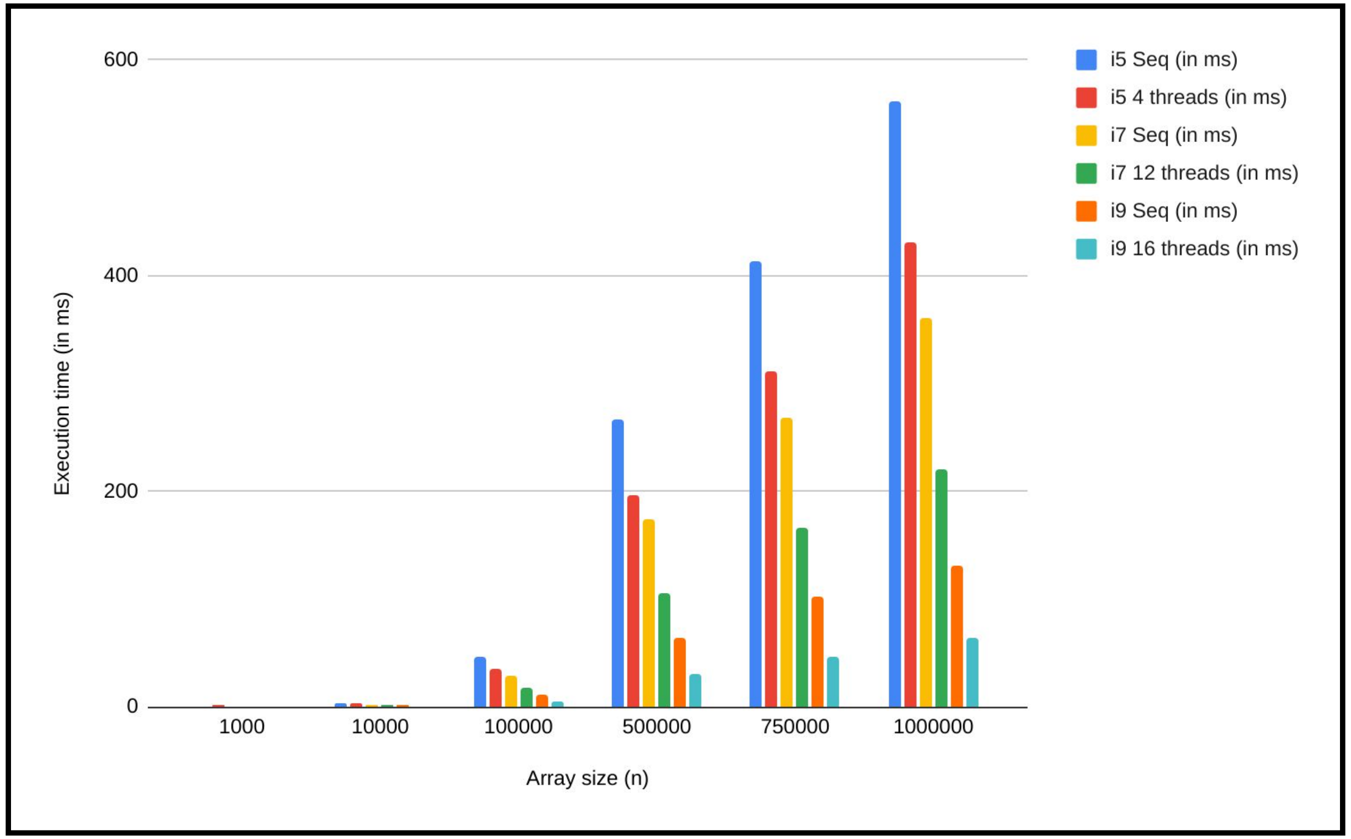
Comparison of execution times for Sequential vs Parallel environments (only inter-function parallelism)

Method 3 Results



Comparison of execution times for Sequential vs Parallel environments (only inter-function parallelism)

Method 3 Results



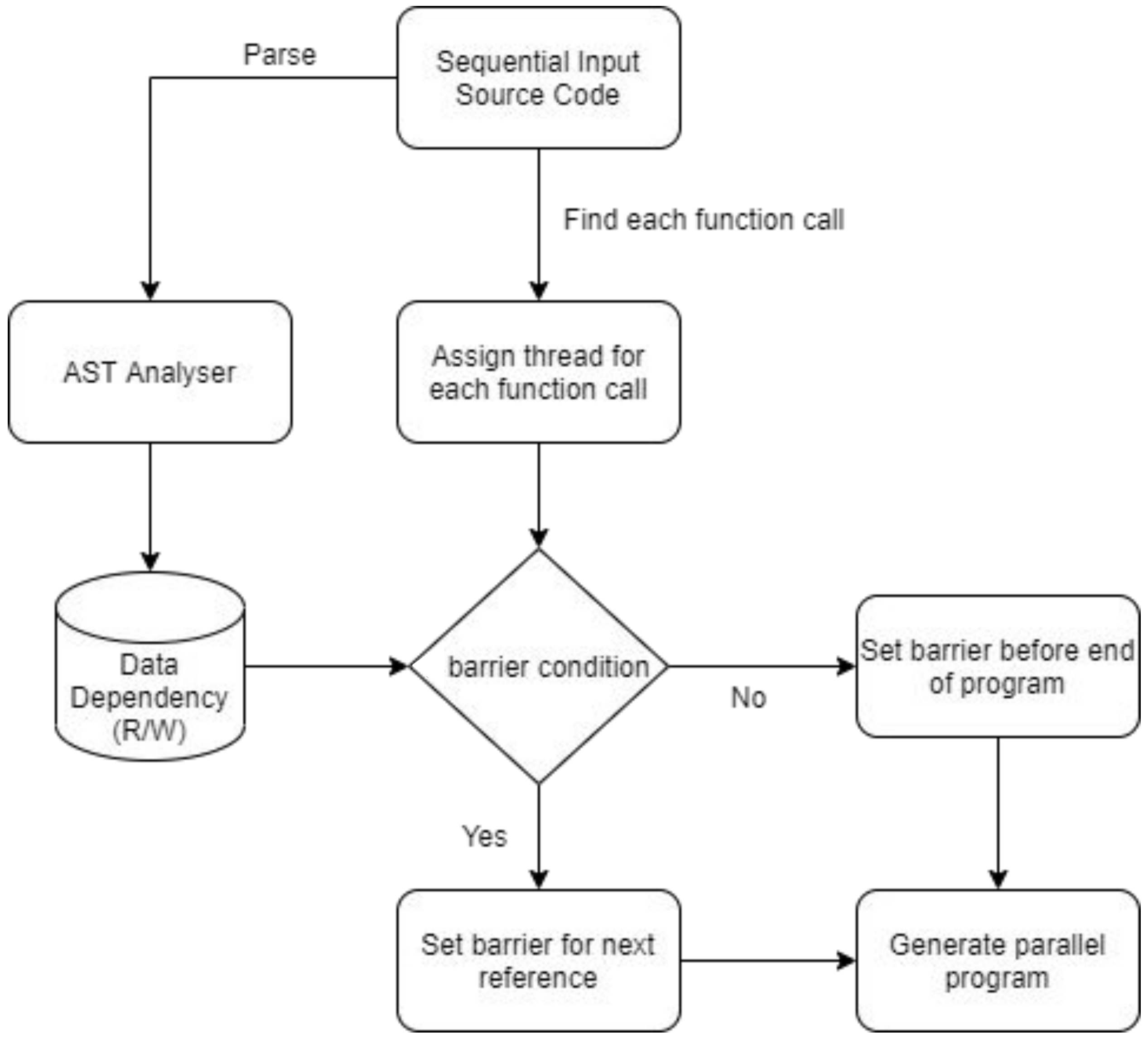
Comparison of execution times on different machines, sequential and parallel (only inter-function parallelism)

Method 3 Drawbacks

- 2 master threads always used for tracking and scheduling
- Large number of mutex locks leads to considerable busy waiting
- Complex generated program
- Large number of selection statements in the generated program leads to frequent invalidation of instruction cache
- Higher complexity of program due to ready and wait queues

Outline

Parallelization Implementation - Method 4



Sequential Input Source Code
Parse
Find each function call
AST Analyser
Assign thread for each function call
Data Dependency (R/W)
barrier condition
No
Set barrier before end of program
Yes
Set barrier for next reference
Generate parallel program

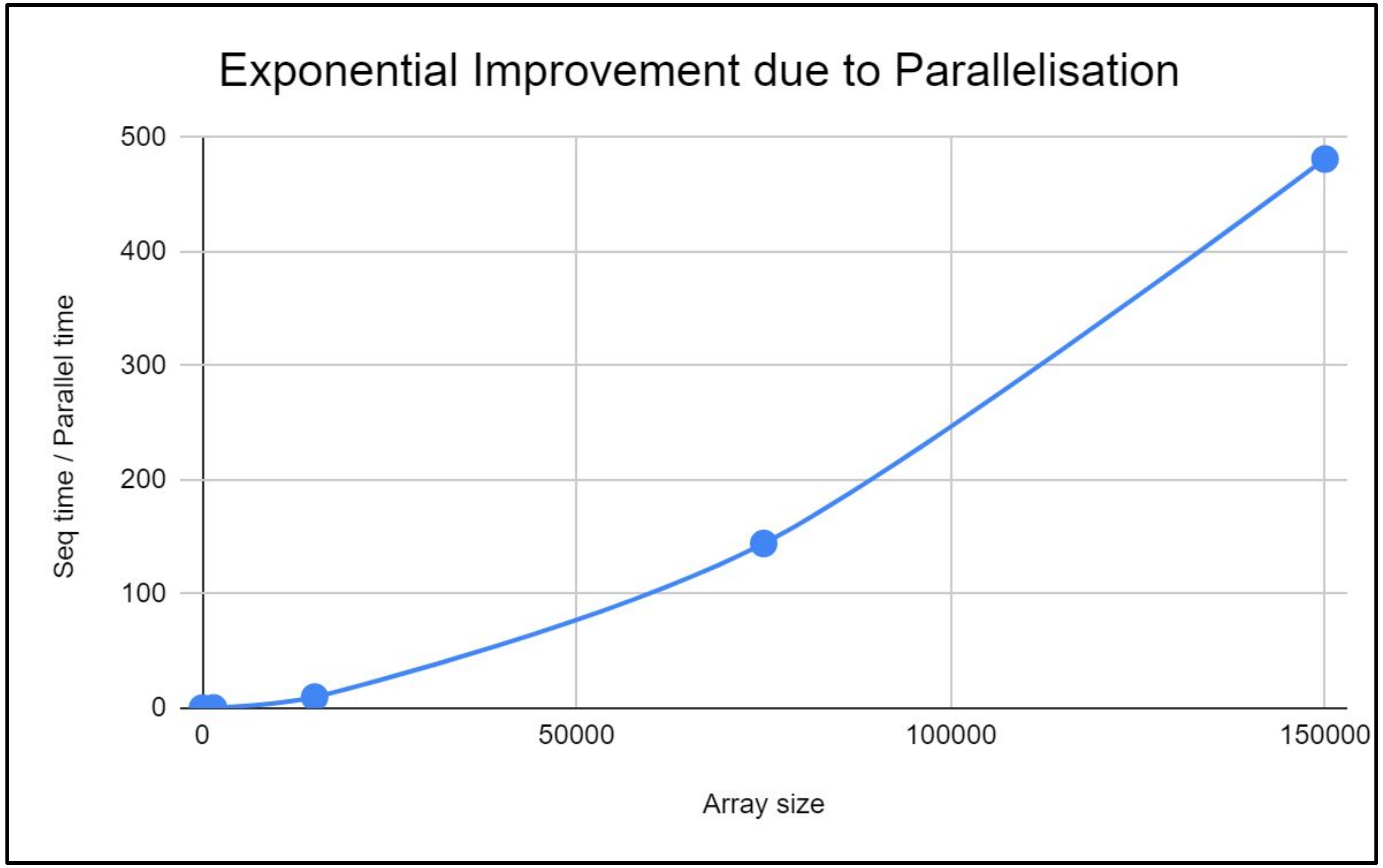
We define ***Barrier Condition*** as a check of whether

- **return value** of the function call is used at a later point in the program or
- if any of the **arguments are modified** inside the function and used at a later point.

Project Demonstration

DEMO

Method 4 Results

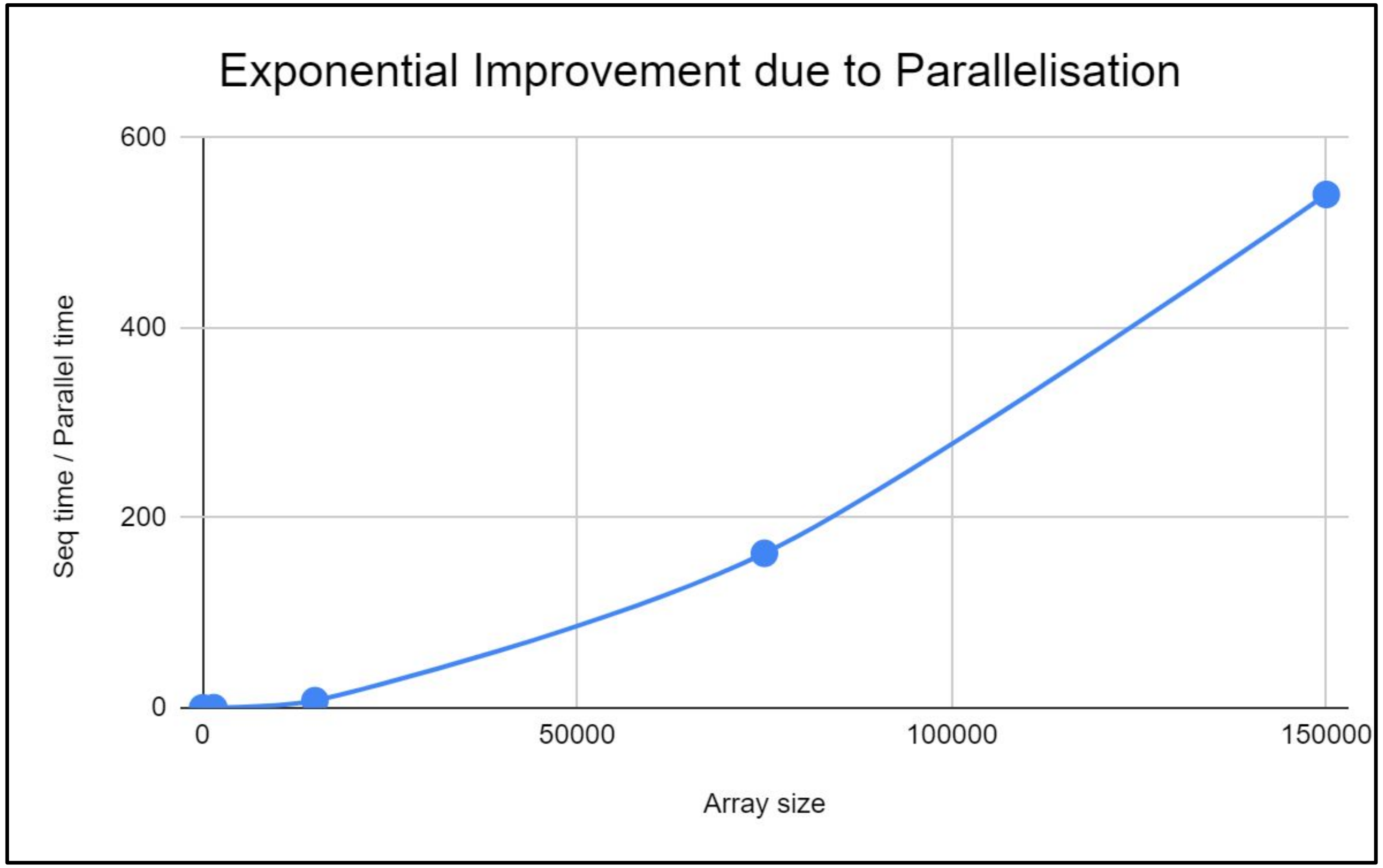


Nearly **500** times faster for larger array sizes!!

Hardware setup : Core i5 - 2nd gen - 2 core machine

Ratio of **Sequential** execution times to **Parallel** execution times

Method 4 Results

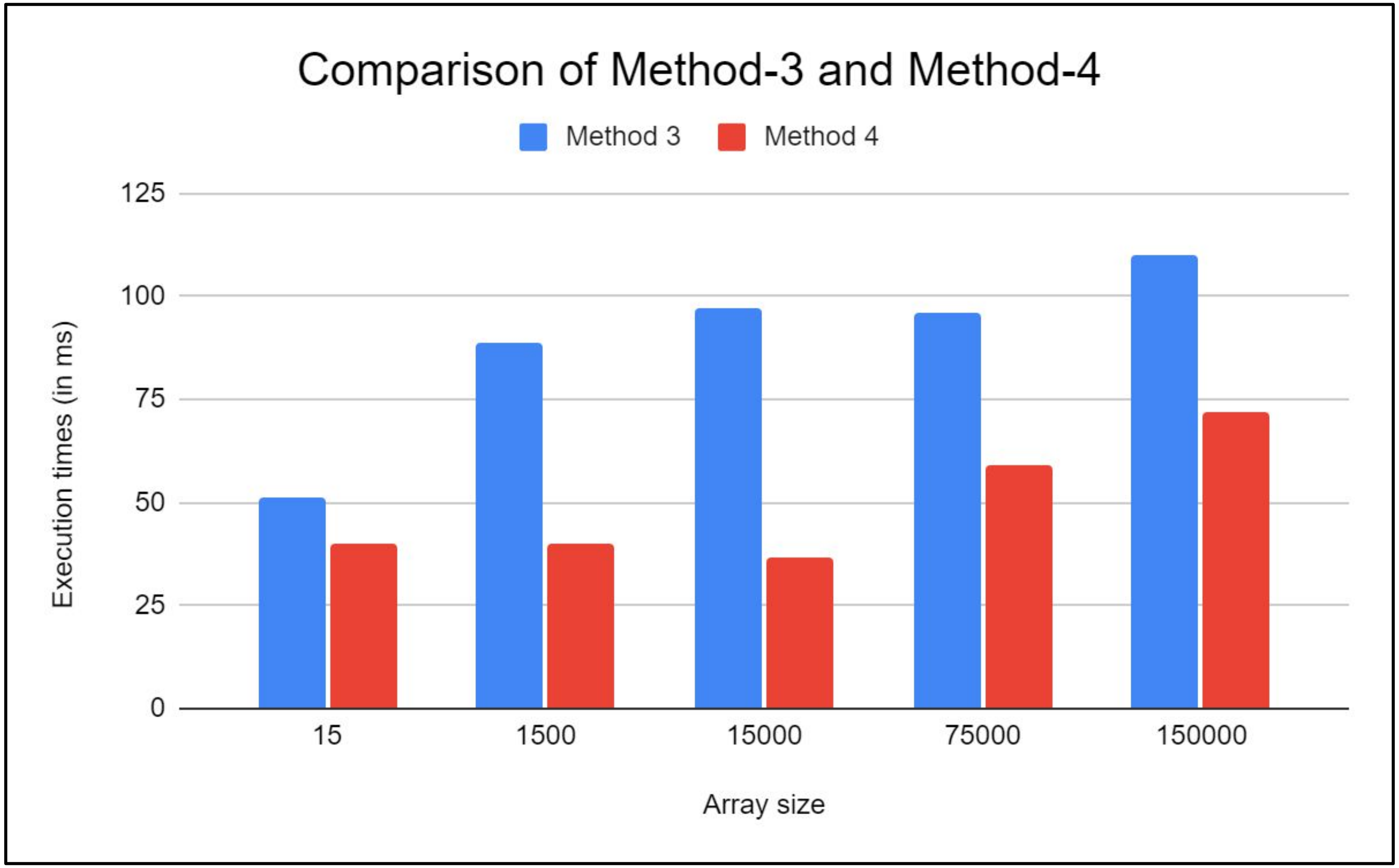


Hardware setup : Core i7 - 9th gen - 6 core machine

Nearly 550 times faster for larger array sizes!!

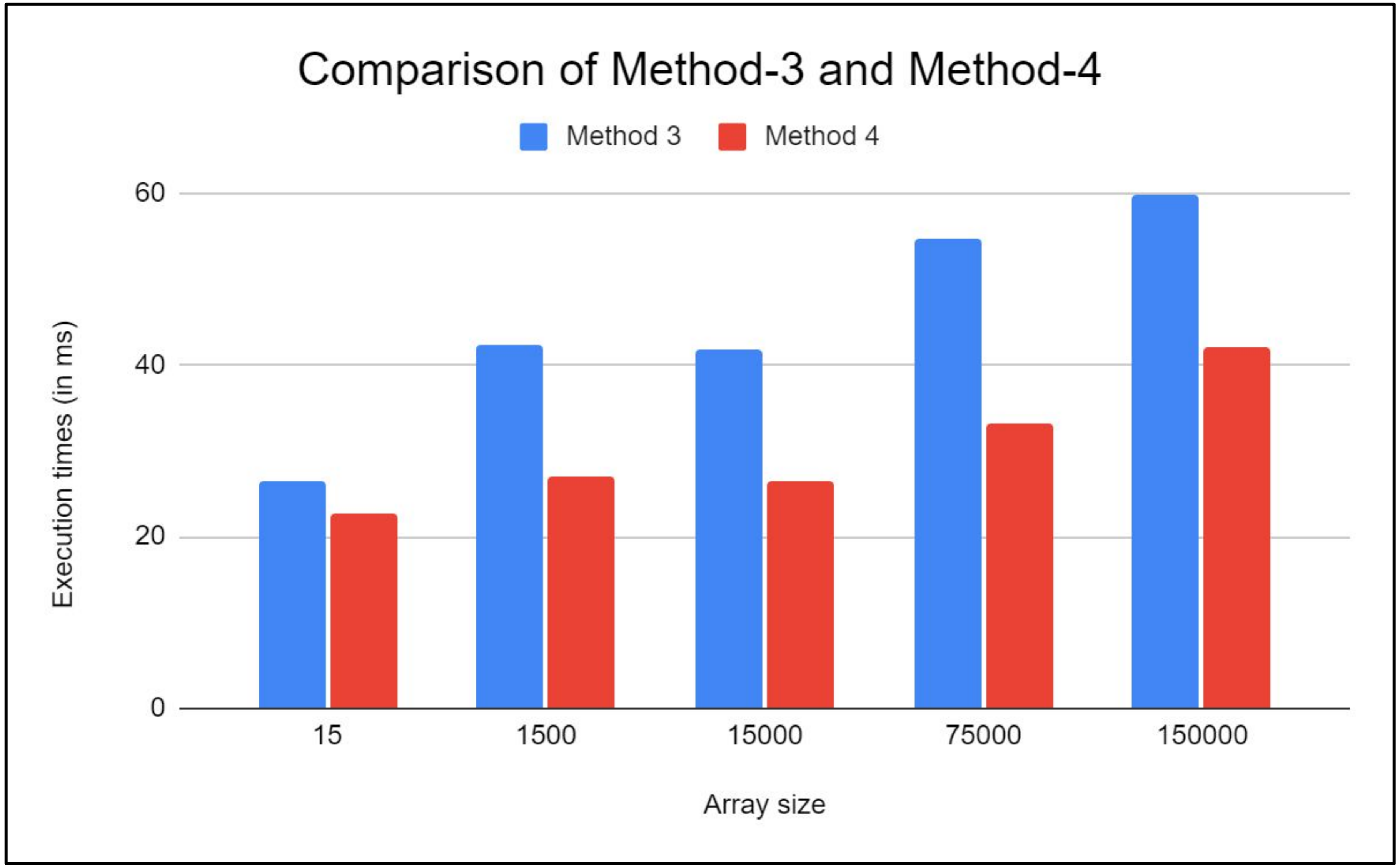
Ratio of **Sequential** execution times to **Parallel** execution times

Method 3 vs Method 4



Hardware setup : Core i5 - 2nd gen - 2 core machine

Method 3 vs Method 4



Hardware setup : Core i7 - 9th gen - 6 core machine

Outline

Technologies Used

- **OpenMP** : For usage of parallel OpenMP directives to segment independent code segments and execute them in parallel.

- **C++ thread library** (based on Pthreads) : For creation and execution of threads for individual functions.

  - Thread Pools
  - Future and Promises
  - Variant and Visit
  - Mutex and Lock guards

- **CLAVA/LARA** : For generation of an enriched Abstract Syntax Tree to parse and obtain data dependencies.

- **JS and PHP** : For creating a user interface to upload sequential code and generate the parallel equivalent.

Outline

Documentation

- Project Report
- Plagiarism Report
- IEEE Paper Draft
- A3 size Poster
- Github repository: https://github.com/Spielerr/Capstone_Project

IEEE Paper Draft

- We intend on separating these ideas into two papers respectively

- Our capstone project entails two main ideas:
  - Inter-functional parallelism - Functional parallelism
  - Intra-functional parallelism - Program Comprehension

- Our intention for separation is to allow better clarity on each idea

- Submission to conference:
  - Our aim is for International conferences, so as to have better reach for our paper
  - We intend to focus on possible journal publications as well

List of targeted Conferences / Journals

Sl. No.	Conference Name	Deadline for Submission of Paper	Conference Date
1.	International Conference on Parallel Programming and Computing, ICPPC, Rome Italy	April 2nd, 2022	May 03-04, 2022
2.	International Conference on Distributed and Parallel Computing, ICDPC in Sydney, Australia	April 16th, 2022	May 17-18, 2022
3.	International Conference on Parallel and Distributed Computing Systems, ICPDCS in Amsterdam, Netherlands	April 6th, 2022	August 05-06, 2022
4.	International Conference on Network and Parallel Computing ICNPC in Paris, France	April 16th, 2022	May 17-18, 2022
5.	International Conference on Parallel and Distributed Computing and Systems, ICPDCS in Montreal, Canada	May 14th, 2022	June 14-15, 2022

List of targeted Conferences / Journals

Sl. No.	Conference Name	Deadline for Submission of Paper	Conference Date
6.	6th International Conference on High Performance Compilation, Computing and Communications (HP3C 2022)	Feb 1, 2022	Jun 23, 2022 - Jun 25, 2022
7.	International Conference on Parallel and Distributed Computing Systems, ICPDCS in Dubai, United Arab Emirates	April 08th, 2022	May 09-10, 2022
8.	International Conference on Computational Mathematics, Parallel and Distributed Computing ,ICMPDC in Vancouver, Canada	April 19th, 2022	May 20-21, 2022
9.	International Conference on Distributed Systems and Parallel Computing ICDSPC in Dubai, United Arab Emirates	May 27th, 2022	June 29-30, 2022
10.	International Conference on Advances in Distributed and Parallel Computing, ICADPC in Tokyo, Japan	June 22nd, 2022	July 22-23, 2022

List of targeted Conferences / Journals

Sl. No.	Journal Name	Link to Journal
1.	International Journal of Parallel Programming	https://link.springer.com/journal/10766/volumes-and-issues
2.	Parallel Computing	https://www.journals.elsevier.com/parallel-computing

Outline

Team Roles and Responsibilities

PHASE 1

Team Roles and Responsibilities

Week	Task	Description	Assignee
1-2	Literature Survey on parallelization techniques	An in depth literature survey into different parallelization techniques developed and used so far. To better understand existing techniques and their advantages and pitfalls, so as to consider those during our development	Karan & Manu
1-2	Literature Survey on parallelising tools	To understand how different tools work, their scalability, domain of application and the impact. Also to better understand the tools and the driving principles behind them, hence allowing us to be in a better position for our development	Darshan & Mayur
3-5	Experimentation with existing libraries	We experimented with existing libraries such as openMP, c++ thread library, openMPI, OpenCL, CUDA etc. This allowed us to gain a better understanding of the target code we intend to generate	Darshan & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
3-5	Experimentation with different parallelising compilers	Tried out different parallelising compilers such as pluto, parawise, parafrase-2 etc. This was done to better understand what exactly is the generated code, and how useful it was to the case of parallelisation in the general case	Karan & Manu
6	Discussed Dynamic analyser with senior(Skanda)	Had a meet with seniors to discuss their capstone project on dynamic analysis, to better understand how program comprehension can be carried out	Darshan, Karan, Manu & Mayur
7	Naive GDB approach	First attempt at automating parallelisation, used GDB to identify relevant code and replace it with parallel versions of code. Specifically tried it out with a sequential sort function	Darshan, Karan, Manu & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
6-8	Initial parallelisation technique	Assuming we have knowledge about the underlying algorithm, proceeded with the implementation to replace relevant code with parallel version	Darshan, Karan, Manu & Mayur
9-10	Understanding CLAVA/LARA	CLAVA/LARA tool offered us the ability to query AST. This could help us in identifying relevant sections of code and replacing with equivalent parallel versions	Darshan & Mayur
9	Review 2 preparation	Completed required documents such as SRS, PPT etc. Submitted the same to guide and prepared for initial demo	Karan & Manu

Team Roles and Responsibilities

Week	Task	Description	Assignee
9-10	LARA based tool development	Used LARA to query AST for the information needed, for parallelising and replacing code as needed. This was used as the foundation for later improvements in all our proposed methodology thus far.	Darshan & Mayur
11	Check feasibility of proposed method 1	Check on the feasibility and potential pitfalls of method 1. This allowed us to better understand the changes necessary and how to implement them	Karan & Manu
11	Development of Method 1	We pursued an implementation of our proposed methodology, and possible changes to the initial design	Darshan, Karan, Manu & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
9-10	Implement suggestions from review 2	We pursued the changes recommended by guide during our demo in review 2. This involved adding features to handle functions that were not being parallelised and any other code in the main function	Darshan & Mayur
12	Design of Method 2	We set upon a new proposed methodology, where we tried to design based on reordering functions while maintaining their dependency. This involved studying the usage of future and promise	Karan & Manu
12	Implementation of Method 2	We refined our ideas. We learnt from Method 1, Review 2 and literature surveys, and used the concept of futures and promise, along with reordering of functions, to build a naive scheduling algorithm.	Darshan, Karan, Manu & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
12-13	Carried out experimentation on Method 2	We tried various test cases on method 2. And tried to find the edge cases. While it handled all the cases handled by Method 1, and did so in an optimised manner, there were a few issues in method 2. This made us understand the need for fine grained control	Karan & Manu
13	Design of Method 3	From our understanding of Method 2 drawbacks, we realised there needs to be more fine grained scheduling. We designed the same, and generated code that schedules, based on data dependency and other required conditions	Darshan, Karan, Manu & Mayur
13	Modifications to LARA code, for information needed for scheduling	The information required to do fine grained scheduling was extracted from the AST. This made scheduling more feasible and robust.	Darshan & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
13-15	Implementation of Method 3	We carried out the implementation of Method 3, where we undertook activities to put our proposed methodology of fine grained scheduling into action. Our results seemed promising	Darshan, Karan, Manu & Mayur
16	GUI for demo	Upon the suggestion of our guide, we built a GUI, a web interface for the tool. This was carried out in PHP and Javascript	Darshan & Mayur
15-16	Review 3 preparation	Began the preparation of documents and ppt for review 3 held by college. This also involved the documentation of work done and other related activities	Karan & Manu

Team Roles and Responsibilities

Week	Task	Description	Assignee
17-18	Completion of report	We completed a comprehensive report of the Phase 1 of our capstone work. This involved drafting the report, preparing the required graphs and results, Also involved sending the same for plagiarism check	Darshan, Karan, Manu & Mayur

Team Roles and Responsibilities

PHASE 2

Team Roles and Responsibilities

Week	Task	Description	Assignee
1-2	Literature Survey on Program comprehension	An in depth literature survey into different program comprehension techniques developed and used so far. To better understand existing techniques and their advantages and pitfalls, so as to consider those during our development	Mayur & Manu
1-2	Extended handling of client code	Implementation to include any type of client code apart from function calls (no optimisation for loops or control statements), along with continued refinement to the generator program	Darshan & Karan
3-5	Review 1 Preparation	Completing PPT as required, Addition of separate futures to run our two threads (thread_track and scheduling_fn), separate from void_futures meant for void functions from client code. Additional re-engineering along with fixing certain bugs to handle return value variables storing return values of function calls	Darshan, Karan, Manu & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
3-5	Development and refinement of generation program	To support functions with return value. Perform dependency analysis to find next usage of return value and modified parameter.	Darshan & Karan
5	Discussion with Skanda and team regarding their capstone project of “Code Semantic Detection and Optimization” carried out in the previous year	Had a collaborative talk discussing mutual projects, understanding the constraints of their project, feasibility of usage of their project in our project’s pipeline, and received link to their github project	Darshan, Karan, Manu & Mayur
5-6	Setting up and experimenting with the tool “logic detector” (Skanda’s team)	Tried to identify use cases and involved constraints through a preliminary experimentation	Manu & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
6	Preparation for Review 2	Started working on presentations, interface and preparing demo for Review 2 with our mentor	Darshan, Karan, Manu & Mayur
6-8	Continued refactoring of the generator program	Continued fixing bugs and handling more cases in the generator program	Darshan & Karan
7-8	Tested out various program comprehension models	Analysed different models such as code2vec, code2sec, TBCNN, etc. Identified the advantages and shortcomings for each model.	Darshan, Karan, Manu & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
8-9	Data collection for ML model	Started scraping data for different categories as decided. Using a web scraper to extract data from different code repositories such as Github, Leetcode, etc.	Mayur & Karan
9-10	Fully integrated a basic Program comprehension model into pipeline	Run astminer on the created dataset and train the code2vec model to output the vector embeddings. Created an independent PC module and fully integrated it into the complete tool pipeline. Now we have a fully functioning pipeline from start to finish.	Darshan, Karan, Manu & Mayur
9-10	Method-4 implementation	Design and implementation of a new method to handle parallelization - without the use of master threads, but by making use of only information made available using data dependency analysis.	Darshan & Karan

Team Roles and Responsibilities

Week	Task	Description	Assignee
9-10	Clustering ideas to categorize algorithms	Discussed and designed a clustering technique to categorize input programs into different algorithm categories.Experimentation and designing changes to improve the clustering technique - usage of threshold values, ensemble of binary models, etc.	Mayur & Manu
10-18	Research paper	Decided on splitting project into two papers - inter-function parallelism and program comprehension based parallelism. Worked on first paper of inter-functional parallelism, with continuous modification based on suggestions from our guide	Darshan, Karan, Manu & Mayur
10-11	Further refactoring and bug fixes to generator program	Further refinement of generator program to handle all programs with no exceptions. Necessary modifications and implementation for extended functionality of the new generator program for Method 4	Karan & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
10	Review 3 preparation and presentation	Started working on presentations, interface and preparing demo for Review 3 with our mentor.	Darshan, Karan, Manu & Mayur
11	Review 3	A comprehensive review where we were able to present our new method (Method 4) and show implementations of the Program Comprehension phase. Received very good feedback from the panel members	Darshan, Karan, Manu & Mayur
12	Map reduce program	As suggested by our mentor, a better example which captures the requirement of parallelization must be shown as part of demo in Review 3. Verified all features for this program, analysed the performance gain. Generated results by running it on different architectures and different array sizes. Represented concisely with various graphs in our report	Darshan & Mayur

Team Roles and Responsibilities

Week	Task	Description	Assignee
11-12	Experimentation with other Parallelization tools	Specifically checked out Pluto since it was the only tool with an implementation, rest of them were just concepts. Made a comprehensive list of limitations wrt Pluto and figured out that our tool handles more cases for functional parallelism	Darshan & Karan
13-15	Fixes to randomised nature of the astminer	Due to inconsistencies in the outputs of the astminer, the model received inconsistent path-based representation data. Fixed this by sending any new program through the beginning of the pipeline	Mayur & Darshan
13-15	More fixes to find_future function in generator program	Handled more cases by fixing some logic in the generator program	Karan & Manu

Team Roles and Responsibilities

Week	Task	Description	Assignee
16-17	Refactoring and restructuring of code to improve efficiency of our pre-processing stage of pipeline	Organizing code repository and minor fixes to both parallelization and program comprehension phase	Darshan & Manu
16-18	Final ESA Review preparation	Started working on presentations, interface and preparing demo for ESA review with our mentor. Prepared and finalised report, draft of IEEE paper, ppt and other necessary documents.	Darshan, Karan, Manu & Mayur

Outline

Lessons Learnt

- Learnt and experimented with a multitude of new domains and technologies:
  - Enriched Abstract Syntax Tree generation and querying
  - Data Dependency Analysis
  - Concept of Parallelization
  - C++ Promises and Futures
  - OpenMP
  - C++ Thread library
  - Thread Pools
  - Mutex locks and guards

Lessons Learnt

- C++ Variant and Visit
- Concept of Generating programs for different test cases
- Concept of Program Comprehension
- Path based representation of programs
- Vector embedding for programs
- Dynamic verification

Lessons Learnt

- Overview of the issues that have been overcome in this project:
  - Handling of possible complications due to parallelization:
    - Deadlock
    - Race Conditions
    - Starvation
  - Generalizing the parallelization to handle variety of test cases
  - Handling different edge cases possible
  - Reengineering and Refactoring of generator program to handle different cases

Outline

Conclusion

- Proposed pipeline implemented in its entirety
- Assumptions made in Phase 1 have been eliminated.
- Parallelization Phase:
  - Simplified generated program using Method 4
  - Master threads removed, freeing two threads for function execution
  - Ready and wait queues have been removed
  - Reduced busy waiting for acquiring locks and number of mutex locks
  - Selection and Iterative Statements are handled

Conclusion

- Program Comprehension Phase:
    - Represent the input source code as vector embeddings
    - Find similarities between these embeddings to group them into clusters
    - Additionally verify if the predicted label is accurate using a dynamic verification process
    - “Others” category introduced to ensure correctness of the program is maintained all the time
    - Program Comprehension phase implemented with pipeline along with Parallelisation phase, fully complete

Future Work

- Refinement of both parallelization and program comprehension phases for any gains possible
- Parallelization Phase:
    - Extend the cases handled by generator program for better optimised code
    - Refactor the code to improve efficiency
- Program Comprehension Phase:
    - Extend support to more cases by training on larger datasets
    - Experiment with any new improvements in the area of Program Comprehension

Outline

References

[1] Cantiello, P., & Di Martino, B. (2012). Automatic Source Code Transformation for GPUs Based on Program Comprehension. Lecture Notes in Computer Science, 188–197. doi:10.1007/978-3-642-29740-3_22

[2] Martino, B. D., & Iannello, G. (n.d.); Towards automated code parallelization through program comprehension; 1994

[3] Di Martino, B., & Iannello, G. (n.d.). PAP Recognizer: a tool for automatic recognition of parallelizable patterns. WPC '96. 4th Workshop on Program Comprehension. doi:10.1109/wpc.1996.501131

References

[4] Di Martino, B., & Kessler, C. W. (2000). Two program comprehension tools for automatic parallelization. IEEE Concurrency, 8(1), 37–47. doi:10.1109/4434.824311

[5] Peter Kraft, Amos Waterland, Daniel Y Fu Anitha Gollamudi, Shai Szulanski, Margo Seltzer. (2018). Automatic Parallelisation of Sequential programs

[6] Cristian Ramon-Cortes, Ramon Amela, Jorge Ejarque, Philippe Clauss, Rosa M. Badia. (2018). AutoParallel: A Python module for automatic parallelization and distributed execution of affine loop nests

References

[7] Wim Vanderbauwhede, Gavin Davidson. (2017). Domain-Specific Acceleration and Auto-Parallelization of Legacy Scientific Code in FORTRAN 77 using source-to-source Compilation

[8] Idan Mosseri, Lee-or Alon, Re’em Harel, and Gal Oren. (2020). ComPar: Optimized Multi-Compiler for Automatic OpenMP S2S Parallelization

[9] Hamid Arabnejad, João Bispo, Jorge G. Barbosa, João M.P. Cardoso. (2018). AutoPar-Clava: An Automatic Parallelization source-to-source tool for C code applications

References

[10] João Bispo, João M.P. Cardoso. (2020). Clava: C/C++ Source-to-Source compilation using LARA

[11] Sean Rul, Hans Vandierendonck, Koen De Bosschere. (2007). Function Level Parallelism Driven by Data Dependencies

[12] Di Martino, B.: Algorithmic concept recognition to support high performance code reengineering. Special Issue on Hardware/Software Support for High Performance Scientific and Engineering Computing of IEICE Transaction on Information and Systems E87-D, 1743–1750

References

[13] Di Martino, B., Zima, H.P.: Support of automatic parallelization with concept comprehension. Journal of Systems Architecture 45(6-7), 427–439 (1999)

[14] Gabel, M., Jiang, L., Su, Z.: Scalable detection of semantic clones. In: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, pp. 321–330. ACM, New York (2008)

[15] "Convolutional Neural Networks over Tree Structures for Programming Language Processing" Lili Mou, et al.

References

[16] Bilateral Dependency Neural Networks for Cross-Language Algorithm Classification, by Nghi D. Q. BUI, Yijun YU, Lingxiao JIANG

[17] Uri Alon, Meital Zilberstein, Omer Levy and Eran Yahav, "code2vec: Learning Distributed Representations of Code", POPL'2019

[18] Uri Alon, Shaked Brody, Omer Levy and Eran Yahav, "code2seq: Generating Sequences from Structured Representations of Code"

References

[19] “A Novel Neural Source Code Representation based on Abstract Syntax Tree” Zhang, Jian et. al

Thank
You