Degree thesis

Modeling Intel® Cilk™ Plus Programs with Unified Modeling Languages

Authors: Ata-Ul-Nasar, Mansoor
Supervisor: Dr Sabri Plana
Examiner: Mathias Hedenberg
Semester: VT 2014
Subject: Computer Science
Abstract
Recently multi-core processors have become very popular in computer systems. It allows multiple threads to be executed simultaneously. The advantage of multi-core comes by parallelizing codes to expand the work across hardware. Furthermore, this can be done by using a parallel environment developed by M.I.T. called Intel Cilk Plus, which is design to provide an easy and well-structured parallel programming approach.

Intel Cilk Plus is an extension of C and C++ programming languages that describes data parallelism. This extension is very helpful and easy to use among other languages in this field. It has different features including keywords, reducers and array notations etc. In general, this article describes Intel Cilk Plus and its features. In addition, Unified Modelling Language, activity diagrams are used in term of graphical modelling of Intel Cilk Plus by describing the process of a system, capturing the dynamic behaviour of it and representing the flow from one activity to another using control flow. Later on Intel Cilk Plus keywords and UML diagrams tools will be evaluated, a comparison of different UML modelling tools will also be provided.

Preface

First of all I will like to thank my supervisor, Prof. Dr. Sabri Pllana, for his continuous support, encouragement and patience. His continuous good mood and humour has always been encouraging and inspiring for me.

I wish to thank my sisters, brothers and my parents, especially my brother for being so kind and loving to me. I am also thankful to all of those who supported me during the completion of the thesis.
Contents

1. Introduction ...................................................................................................................... 1
   1.1 Problem definition .................................................................................................. 1
   1.2 Scope ......................................................................................................................... 1
   1.3 Related work ............................................................................................................ 1
   1.4 Outline ....................................................................................................................... 1
   1.5 Social considerations ............................................................................................... 2

2 Background ....................................................................................................................... 3
   2.1 Intel Cilk Plus ........................................................................................................... 3
      2.1.1 Keywords .......................................................................................................... 3
   2.2 UML .......................................................................................................................... 4
      2.2.1 UML Activity Diagram ..................................................................................... 5

3 Tools ................................................................................................................................... 13
   3.1 Modelling tools ......................................................................................................... 13
      3.1.1 Microsoft Visio .................................................................................................. 13
      3.1.2 Visual Paradigm ............................................................................................... 14
   3.2 Integrated development environment ......................................................................... 15
      3.2.1 Microsoft Visual Studio .................................................................................. 16
   3.3 System ....................................................................................................................... 16
   3.4 Survey tool ............................................................................................................... 16

4 Results ................................................................................................................................ 18
   4.1 Test programs .......................................................................................................... 18
   4.2 Modeling Intel Cilk Plus keywords ......................................................................... 19
      4.2.1 cilk_spawn and cilk_sync .................................................................................. 19
      4.2.2 cilk_for ............................................................................................................. 33
   4.3 Survey ...................................................................................................................... 39
      4.3.1 Participants ........................................................................................................ 39
      4.3.2 Questions .......................................................................................................... 40
      4.3.3 Survey results ................................................................................................... 41
5 Discussion............................................................................................................. 42
  5.1 Pros and Cons................................................................................................. 42
  5.2 Comparison of feature................................................................................... 43
6 Conclusions ......................................................................................................... 44
  6.1 Intel Cilk Plus with UML activity ................................................................. 44
  6.2 UML tools .................................................................................................... 44
7 Future work .......................................................................................................... 45
  7.1 UML tools .................................................................................................... 45
  7.2 Intel Cilk Plus ............................................................................................... 45
Reference.................................................................................................................. 46
Appendix .................................................................................................................. a
  Appendix A: Survey participants......................................................................... a
  Appendix B: Survey............................................................................................... c
  Appendix C: Execution time of programs that use cilk_sync, cilk_spawn and cilk_for. ...... e
List of Figures

Figure 2.1 UML Representation of Activity .................................................. 7
Figure 2.2. UML representation of Action ...................................................... 7
Figure 2.3. UML representation of Activity Edge ........................................... 8
Figure 2.4. UML representation of Object flow ............................................. 8
Figure 2.5. UML representation of Control flow ........................................... 8
Figure 2.6. UML representation of Activity final .......................................... 8
Figure 2.7. UML representation of Initial Node ............................................ 9
Figure 2.8. UML representation of Decision node ......................................... 10
Figure 2.9. UML representation of Fork node ............................................. 10
Figure 2.10. UML representation of Join node ........................................... 11
Figure 2.11. UML representation of Note .................................................... 11
Figure 2.12. UML representation of Pin ....................................................... 11
Figure 2.13. UML representation of Expansion Region .................................. 12
Figure 2.14. UML representation of Call Behavior Action ............................. 12
Figure 3.1. Working environment of Microsoft Visio .................................... 14
Figure 3.2. Working environment of visual paradigm .................................. 15
Figure 3.3. Working environment of Microsoft Visual Studio ....................... 16
Figure 3.4. Working environment of freeOnlineSurvey tool .......................... 17
Figure 4.1. An example of Pascal triangle ..................................................... 18
Figure 4.2. UML Activity diagram for Fibonacci method by Microsoft Visio ... 20
Figure 4.3. UML Activity diagram for Fibonacci method by Visual Paradigm .. 21
Figure 4.4. UML Activity diagram for Fibonacci method by Papyrus ........... 22
Figure 4.5. UML Activity diagram for pascalTriangleRow program by Microsoft Visio .......................... 24
Figure 4.6. UML Activity diagram for pascalTriangleRow program by Papyrus .......................... 25
Figure 4.7. UML Activity diagram for pascalTriangleRow program by Visual paradigm ...... 26
Figure 4.8. UML Activity diagram for Karatsuba multiplication program by Microsoft Visio .......................... 28
Figure 4.9. UML Activity diagram for Karatsuba multiplication program by Visual paradigm .......................... 30
Figure 4.10. UML Activity diagram for Karatsuba multiplication program by Papyrus .................. 32
Figure 4.11. UML Activity diagram for intSum function by Visual Paradigm .......................... 33
Figure 4.12. UML Activity diagram for intSum function by Papyrus .................. 34
Figure 4.13. UML Activity diagram for bubble sort function by visual paradigm .................. 35
Figure 4.14. UML Activity diagram for bubble sort function by Papyrus .................. 36
Figure 4.15. UML Activity diagram for primeNumber function by Visual Paradigm .................. 38
Figure 4.16. UML Activity diagram for primeNumber function by Papyrus .................. 39
Figure 4.17. An example question used in the survey ................................... 40
Figure 4.18. Results of our survey in bar chart ......................................... 41
1. Introduction
Nowadays multi-core processors become popular in Desktops, servers and in laptops, it enable multithreading to be executed simultaneously. To take advantage of our multi-core hardware we can parallelize our code, and expand our work across our hardware. To do this we can use parallel environments. There are several parallel environments available like μC++, OpenCL, OpenACC, Intel Cilk Plus etc. In my case I will use Intel Cilk Plus; it is an extension of C and C++, and will model it using UML activity diagrams. The selection of this parallel environment will be based on its less number of keywords or directives, and easy in understanding.

1.1 Problem definition
Intel Cilk Plus is an extension for C and C++ programming languages; it supports data and task parallelism. Sometimes, it is hard to understand how parallelism works in Intel Cilk Plus. The proposed research is aimed to model Intel Cilk Plus using UML activity diagrams to develop a better understanding for Intel Cilk Plus codes. A survey will be conduct among the researchers and students of computer science to evaluate the understanding of these models. Different tools are available to draw UML activity diagrams, but not all of them provide full support and are easy to use. A comparison of some well-known UML drawing tools for activity diagrams will be developed in this research, too.

1.2 Scope
Multiprocessing is very important and widely used in multi core processors. Parallel programming is the best way to utilize those multi core processors. C and C++ programming languages provide support for parallel programming via their Intel Cilk Plus extension. It is hard to understand the flow of parallel programs in Intel Cilk Plus. That is why, this thesis attempts to provide a better understanding of Intel Cilk Plus, and other parallel programming environments also.

1.3 Related work
In general modeling parallel programming with UML has been addressed before. Previous research focuses on high performance computing that includes MPI and OpenMP parallel programs [1].

However, to our best knowledge the UML modelling of Intel Cilk Plus programs has not been investigated so far. In this thesis we address suitability of the UML activity diagrams for modelling Intel Cilk Plus programs. In addition we evaluate existing popular UML tools with respect to modelling Intel Cilk Plus programs.

1.4 Outline
This thesis provides a background information and introduction to Intel Cilk Plus keywords and UML activity classes. Later on in chapter 3, a short introduction of each used tool will be provided. In chapter 4 in results part, three programs for each Intel Cilk Plus keyword are graphically model and explained using UML activity diagrams, the execution time of each program is also provided on different core with different input values. In chapter 5 in discussion, we are going to discuss advantages and
disadvantages of different used tools. In addition, we have made a comparison of different features in different tools. In chapter 6, we have concluded various facts regarding our research. Furthermore, a discussion about different possibilities for extending this work in different directions is provided.

1.5 Social considerations
This section provides a discussion regarding the social considerations and their ethical issues, and impact of my work in society.

The thesis work is involved with an online survey with computer science students. During this we consider all ethical issues e.g.

**Handling personal data:** We only request student’s name while completing

**Personal identification:** We did not show any personal identification of student at any stage including its name, email, age etc.

**Specific requirements:**

**Privacy policy:** There was nothing confidential from our perspective in the survey.

**Recommended content:** All contents in the survey were recommended to fill.

**Permission:** We had permission from survey participants to use the survey data for research purpose. [21]

This thesis provides more understanding of Intel Cilk Plus keywords and helps to choose suitable modelling tools for computer science students. This good understanding will help them to contribute in the field of UML modelling and in parallel computing, and that is the need of current the age.
2 Background
In this section we provide background information that is relevant for this thesis. We will introduce the Intel Cilk Plus programming language, and thereafter we give an overview of the UML.

2.1 Intel Cilk Plus
Intel Cilk Plus is an extension of C and C++ programming languages. It is used for data parallelism and task parallelism. Intel Cilk Plus is designed to provide simple and well-structured codes. It is easy to learn and simple to use; it supports both C and C++ languages. It was developed at Massachusetts Institute of Technology (M.I.T.) and first released in 2010. It has two major elements keywords and reducers [2].

2.1.1 Keywords
Intel Cilk Plus has three keywords: cilk_sync, cilk_spawn and cilk_for. These names were chosen to minimize the conflicting with keywords used in already existing programs and they are difficult to type [3, 27].

**cilk_spawn**
The Intel Cilk Plus keyword cilk_spawn allows parallel operations. In cilk_spawn we can execute more than one instruction in parallel without waiting for previous instruction to finish and return. In other words, cilk_spawn allows next instruction to run in parallel with the current instruction.

**Syntax**
The syntax of cilk_spawn is as follows:

```
spawning-expression:
  cilk_spawn function-or-functor ( expression-list_opt )

sync-statement:
  cilk_sync ;
```

According to the above syntax, we can use normal function call, a member function call or the function call operation of a function object as an expression following the cilk_spawn keywords [3, 27].

**cilk_sync**
The Intel Cilk Plus keyword cilk_sync specifies that all child functions spawned from this function must be complete before execution continues. In order to cilk_sync to be in progress, the call of cilk_spawn in the function must be executed. In other words, we placed cilk_sync after all spawned function in order to get the results. The use of cilk_sync alone is useless; it should be used with cilk_spawn in order to get its advantage.
**Syntax**
The syntax of cilk_sync is as follow:

spawning-expression:
- cilk_spawn function-or-functor (expression-list_opt )

sync-statement:
- cilk_sync;

The cilk_sync does not affect children of other functions. It only synchronises the children spawned by function where it is used [3, 27].

**cilk_for**
The Intel Cilk Plus keyword cilk_for replace traditional iterative for loop into a parallel for loop [5]. In other word, cilk_for allows us to execute iteration of the loops body in parallel.

**Syntax:**
Here is the syntax of cilk_for loop:

parallel-loop
- cilk_for (init-clause; condition; increment-expr)
- statement;

In the code above, cilk_for looks like a normal C++ for loop. The only difference is that in Intel Cilk Plus we are using cilk_for instead of for, used in C++. [3, 27]

**2.2 UML**
Unified Modelling Language (UML) is a standard modelling language. One of the main characteristics of UML is, its process independence, thus, it can be used in different processes. UML was initially created and developed by Grady Booch, Ivar Jacobson, and James Rumbaugh at Rational Software in 1994-1995, and adopted as a standard by Object Management Group (OMG) in 1997. Later on in 2002, it was accepted by International Organization for Standardization (ISO). Since then, it has been constantly updated and developed. In this thesis, UML 2.4 released in March 2011 by OMG will be used. [4]

The scope of UML is very broad. UML deals with a large set of application domains. It provides tools for analysis, design, and implementation of software based systems. It is also capable of modelling business and business related.

UML consists of different kinds of diagrams that provide two types of view - static and dynamic. Static view shows the static structure of the system using objects, attributes, operations and relationships. Static consists of Class diagrams, Object diagrams, Package diagrams, Component diagrams etc. Dynamic view shows the dynamic structure of the system by showing the series of changes to the system over time. Dynamic view consists of Use Case diagrams, Activity diagrams, State Machine diagrams, Interaction diagrams. [4]
2.2.1 UML Activity Diagram

Unified Modelling Language, Activity diagram is an important diagram in UML. It helps the user to understand the flow of control of the system from activity to activity. UML Activity diagram is similar to flow charts. In UML Activity diagram, focus is on sequence and condition for coordinating lower-level behaviors [5, 6].

Activity diagrams have different levels. The following classes of UML Activities might be distinguished: Fundamental activities, Basic activities, Complete activities, Structured activities, Complete Structured activities and Extra Structured activities. [6]

Graphic nodes included in activity diagrams
In this section, graphical nodes, paths and graphic elements for containment are listed and activity diagrams describing some of them will be provided later on.

AcceptEventAction
This is an action that waits for the occurrence of an event meeting specified condition. [24]

Action
For details about action please see Action.

ControlNode
It is an abstract activity node; it includes Decision node, Fork node, join node, Initial node, activity final, Flow final and Merge node [37]

DataStore
It is a central buffer node; it is use for non-transient information. [38]

DecisionNode
For details about DecisionNode please see Decision Node

FinalNode
For details about FinalNode please see Activity Final Node

FlowFinal
For details about FlowFinal please see Activity Final Node

ForkNode
For details about ForkNode please see Fork Node

InitialNode
For details about InitialNode please see Initial Node

JoinNode
For details about JoinNode please see Join Node

MergeNode
It has multiple incoming and single outgoing edges.[39]

SendSignalAction
It creates a signal from its input and sends it to a particular target.[40]

Graphic paths included in activity diagrams
In this section, graphical paths of activity diagrams are listed below.
ActivityEdge
For details about ActivityEdge please see Activity Edge.

ControlFlow
For details about ControlFlow please see Control Flow.

ObjectFlow
For details about ObjectFlow please see Object flow.

Graphic elements for containment in activity diagrams
In this section, graphical elements for containment in activity diagrams are listed.

Activity
For details about action please see Activity

ActivityPartition
It is an activity that identifying actions that have some common character. [43]

InterruptibleActivityRegion
It is an activity group that supports termination of tokens. [41]

ExceptionHandler
It’s an element that specifies a body to execute in case if the specified exception occurs [42].

ExpansionRegion
For details about ExpansionRegion please see Expansion Region.

Below is an explanation of activity classes used in this research.

Activity
An activity is a parameterized sequence of behaviours. It is introduced to flow the models that coordinate other behaviours. An activity can be shown as (see Figure 2.1) a rounded-corner rectangle that includes;

- An object flow for sequencing the data produced by one node that is used by another node.
- A control flow that shows the sequence of the activities/node.
- A control node that includes decision nodes.
- Merge nodes.
- Final nodes.
- Fork nodes and join nodes. [7]
Action

Action is a single step inside an activity. That is, a step that has no further decomposition. An activity consists of different actions and those actions can have incoming and outgoing edges. In order for an action to be executed, it must satisfy its input condition. The notation used for action is in the form of a rounded cornered rectangle. We can describe the local pre- and post-conditions of an action by keywords «localPrecondition» and «localPostcondition», as shown in Figure 2.2. [8]

Activity Edge

ActivityEdge is a directed connection between two activity nodes, e.g. between source and target. If a guard evaluates true for a token then that token is allowed to pass through the edge. ActivityEdge covers both control and dataflow edges. The notation used for activity edge is an open arrowhead line connecting two activity nodes. It is also possible to give the name to the ActivityEdge near the arrow as shown in Figure 2.3. [9]
Object flow
It is a generalization of “ActivityEdge” that starts a new activity when the previous activity is done. Through object flow we can pass objects and data. The notation of an Object flow is a simple arrowed line, as shown in Figure 2.4.[10]

Control Flow
It is a generalization of “ActivityEdge” that starts a new activity when the previous activity is done. Through control flow we can only pass control tokens. The notation of control flow is an arrowed line that connects two activities as shown in Figure 2.5. [11]

Activity Final Node
A node that stops all flows in an activity is called Activity Final Node. It is possible to have more than one Activity Final Node in an activity. However, when we reach the first Activity Final Node all flows in the activity will be stopped. In other words, it stops all the executing actions in the activity. The notation for Activity Final Nodes is a filled circle with a hollow circle as shown in Figure 2.6. [12]
**Initial Node**
An Initial Node is a control node. It is a starting point for execution of an activity. It is possible to have more than one initial node in an activity. Notation for initial activity is a filled circle, as indicate in Figure 2.7. In example below, the initial node passes control to do Work action at the beginning of the activity. [13]

![Initial Node](image)

Figure 2.7. UML representation of Initial Node

**Decision Node**
Decision node is a control node. Decision node receives tokens from incoming edge and presents them to outgoing edges. Decision node has one or two incoming edges and at least one outgoing edge. These incoming and outgoing edges must be either all object flows or all control flows. Every token that arrives at an incoming edge is offered to an outgoing edge.

The direction of an edge depends on the guards decision. In other words, an incoming edge will continue to a specific outgoing edge depending on a guards order. In addition, the order of evaluation (Decision nodes) is not usually defined.

The notation of Decision node is a diamond-shape symbol. The decision condition is described by the keyword \(<\text{decisionInput}\>\), and is attached to the appropriate decision node symbol. An example of diagram of Decision node is provided below. However, The determination of the edge that should be traversed is done by the guards of outgoing edges. The tokens that are not accepted by all other outgoing edges will be operated in a predefined “else” guard for one outgoing edge, like the process of an "else" condition in C++. [14]
Fork Node
Fork node is a control node that divides the flow into multiple flows. Fork node is a generalization of “Control node”. Fork node has one incoming edge and multiple outgoing edges. All these edges are either object flow (Object flow edge: edges from Action node to Action node) or control flow (Control flow edge: edges from Object node to Action node or other way around).

Simple line segment is used as a notation for Fork node. Fork node consists of a single activity edge entering it, and two or more edges leaving it as shown in Figure 2.9.[15]

Join Node
Join node is a control node that synchronizes multiple flows. Join node is a generalization of Control node. Join node is used to synchronize the incoming edges and give the control to the next action node using one outgoing edge.

If all tokens at incoming edges are control tokens, then we will get one control token at outgoing edge. If the tokens at incoming edges are control tokens and data tokens, then we will only get data tokens at outgoing edge. Join node consists of one or more activity edges entering it and only one edge leaving from it. The notation of Join node is provided in Figure 2.10. [16]
Figure 2.10. UML representation of Join node.

Note
A note provide the ability to attach different remarks/comments about an elements, it used to give more understanding of element as shown in Figure 2.11.[5]

Figure 2.11. UML representation of Note

Pin
Pin is an Object node for inputs and outputs to actions. In other words, pins are further decomposed into two types - input pins and output pins. Both input and output pins are Object nodes that receive from and deliver to other action through object flows respectively.

The notation of input or output pin is a small rectangle attached to the action. Name of Pins can be shown beside the pins as shown in Figure 2.12.[17]

Figure 2.12. UML representation of Pin

Expansion Region
Expansion region is a structured activity region that executes multiple times. Expansion region is a nested region of an activity that has explicit input and output. Here we have one or more inputs, each input is a collection of values and each collection must be of the same type.
Expansion region has three models: parallel, iterative and stream. In parallel mode all executions happen independently and parallel. Expansion region is drawn as a dashed rounded box, on the upper left corner we mention the keywords parallel, iterative and stream. Inside the dashed boundary, expansion node symbol is placed. This dashed boundary also includes input and output Expansion nodes that are drawn as small rectangles. A sketch of Expansion region is shown in Figure 2.13. [18]

![Figure 2.13. UML representation of Expansion Region](image)

**CallBehaviorAction**

CallBehaviorAction is an activity that calls the behavior directly. We indicate call CallBehaviorAction by placing a rake-style symbol inside our action. The rake symbol indicates that this invocation starts another activity as shown in Figure 2.14. [19]

![Figure 2.14. UML representation of Call Behavior Action.](image)
3 Tools
This section provides a general introduction and an overview of different tools used in this thesis. These tools includes: modelling tools, integrated development environment (IDE) and used operating system.

3.1 Modelling tools
This section provides a general introduction and an overview of different modelling tools used in this thesis. These modelling tools include Microsoft Visio, eclipse papyrus and visual paradigm.

3.1.1 Microsoft Visio
Microsoft Visio is a powerful diagramming application of Microsoft with a rich set of built-in stencils.

Microsoft Visio was first introduced by Shapeware corporation in 1992, and was acquired by Microsoft in 2000. As of April 2014, Visio 2014 is the most up-to-date version of the Visio application. This software is available in two edition: Visio 2014 Standard and Visio 2014 Professional. Even though, it is a commercial software, the free versions are available for students under certain conditions. Also, the most common file format supported by Microsoft Visio to read and write drawing is VSDX.

Microsoft Visio helps to create object oriented models for complex software system. Specifically, it provides stencils for:

- Process Diagrams
- Engineering Drawings and Diagrams
- Architectural Drawings
- Work Flow Diagrams
- Timelines
- Software modeling
- Database Models
- Networks Models
- Business Modeling
- Scheduling
- Maps

Microsoft Visio helps to create and customize professional diagrams in few clicks. In Microsoft Visio, it is easy to find the required stencil with a built-in search. Also, Microsoft Visio’s feature of a new print preview provides customers with a better view of the diagram final look. Like many other UML tools, Microsoft Visio also provides reverse engineering for C++ and C#.[20,22]
Figure 3.1 depicts modeling environment of Microsoft Visio. On the left hand side are modeling elements. On the top are tools and menus. In middle right we have drawing space.

### 3.1.2 Visual Paradigm

Visual Paradigm for UML is a cross platform UML tool that supports UML 2.0 for the Object Management Group (OMG). In the latest UML 2.1 notation it supports 13 different UML diagrams including: [23, 24]

- Class Diagram
- Use Case Diagram
- Sequence Diagram
- Communication Diagram
- State Machine Diagram
- Activity Diagram

Visual Paradigm for UML is a modelling platform for UML Standard Edition (VP-UML SE) that is designed to support developers, system architecture, and UML designers to accelerate the analysis and design processes for different applications. It also supports requirement modelling, database modelling, business process modelling, object-relational mapping, team collaboration and documentation generation.[23]

In additions to modelling, Visual Paradigm for UML supports code engineering capability for Java, C++, PHP and VB. Furthermore, Visual Paradigm supports reverse engineer diagrams. Visual Paradigm is a commercial software. However, the free version is available for students under certain conditions. [23]
Figure 3.2. Working environment of visual paradigm.

Figure 3.2 depicts modeling environment of visual paradigm. On the left hand side are visual paradigm projects, right after this we have modeling elements. In the right hand side we have drawing space. On the top are tools and menus.

3.1.3 Eclipse papyrus
Papyrus is another graphical editing tool for Eclipse-based modelling languages amongst them UML 2 and related Modelling Languages such as MARTE and SysML. Papyrus provides a very advanced support of UML profiles and also supports reverse engineering from Java codes to UML diagrams. It has merged with eclipse and become a new Eclipse project. Now Papyrus is an open source Eclipse Model Development Tools, and we can use it as an eclipse plug-in. Since 2008, the eclipse Papyrus team is working on this tool. Like many other UML tools Papyrus provides support for diagram export as images. The latest version 1.0.X of Papyrus was released in June 2014, which can be installed on Eclipse Luna 4.4. [25, 26]

UML 2
It supports full graphical editors for UML Structure Diagrams UML, Behaviour Diagrams and UML profiles defined by OMG. It targets to implements 100% OMG specification. [25, 26]

SysML
It provides complete support to SysML diagrams identified in the OMG SysML specification. It includes specific graphical editors required for SysML. [25, 26]

3.2 Integrated development environment
In this section we will give an introduction of integrated development environment (IDE) that is used for compilation of Intel Cilk Plus programs.
3.2.1 Microsoft Visual Studio

The tool that is used for compilation of Intel Cilk Plus programs is Intel® C++ Studio XE for Windows, and is integrated into Microsoft Visual Studio.[32]

Figure 3.3. Working environment of Microsoft Visual Studio.

Figure 3.3 depicts programming environment of Microsoft Visual Studio. On the left hand side is a side menu bar of projects. On the top are tools and menus. In middle right we have code editor.

3.3 System

64 bit system is used in the experiments with following specification:

1. Processor: Intel(R) Core(TM) i5-2435M CPU @ 2.4GHz 2.40 GHz
2. Installed memory: 8.00 GM
3. Windows edition: Window 7 home Premium Service Pack 1

3.4 Survey tool

In this section we will give an introduction of survey tool that will be used for collecting feedback for our work.

The tool that is used for creating survey was a web based application provided by Freeonlinesurveys.com. It is commercial software that has different types of subscriptions. It can be used for creating survey, polls and quizzes.
Figure 3.4 depicts online survey tool of freeOnlineSurvey. On the left hand side are different question types. On the top are tools and menus. In right hand side we have place for adding survey questions.
4 Results
In this section I will evaluate Intel Cilk Plus keywords and UML diagrams tools. In evaluation process I will model selected Intel Cilk Plus programs using different UML modeling tools, a survey will be conduct to see the appropriateness of these models. For evaluation I select six different Intel Cilk Plus programs, three for cilk_sync, cilk_spawn and three for cilk_for. I also select three different UML modeling tools. I will try to model each Intel Cilk Plus program by three different selected UML tools to see which tool is good of creating UML activity models for Intel Cilk Plus programs.

4.1 Test programs
Here is a description of all programs that are used in this program.

1. *Fibonacci* method is used to calculate fibonacci number recursively. In order calculate a fibonacci N we need previous two fibonacci numbers (i.e. fibonacci of N-1 and fibonacci of N-2). The key idea here is that the calculation of \(Fibonacci(n-1)\) will be executed in parallel with the calculation of \(Fibonacci(n-2)\) without interference. The parallelism can be express using Intel Cilk Plus keywords [27].

2. PascalTriangleRow is a program that calculate and return nth row of a pascalTriangle, in order to get each member of this row we need to calculate factorial of three different values. By using cilk_spawn and cilk_sync we can calculate these three values in parralel. It is a triangle of numbers, where every row is generated from its previous row. In the pascal triangle every number of nth row is an addition above two numbers (expect first and last). here is an example of pascal triangle;[28].

![Image of Pascal Triangle]

Figure 4.1. An example of Pascal triangle

3. Karatsuba parallel function is using a fast multiplication algorithm that reduces multiplication time of two n-digit numbers. It uses divide and conquer approach [29].

4. Bubble Sort method is used to sort input list of random numbers using bubble sort algorithm.
5. Int Sum method is used to calculate sum of N positive numbers, in order to observe the effect of parallel program there is a delay of 500 ms in every iteration.

6. isPrime method is used to calculate all the prime numbers between 0 to N, in order to observe the effect of parallel program there is a delay of 500 ms in every iteration.

These programs were run on Microsoft visual studio. Each program was executed at least 3 times on different inputs (e.g. N, 2N, 4N) and on different threads (e.g. 1, 2, 4), we chose the middle value as a result value for our data collection. In appendix C you can see the table provides the execution time of these programs used in this thesis.

4.2 Modeling Intel Cilk Plus keywords

In this section we will model Intel Cilk Plus keyword using UML Activity diagrams, we model three different C codes example with Intel Cilk Plus keywords and model three different UML drawing tools.

4.2.1 cilk_spawn and cilk_sync

In this section we will use three different examples to model Intel Cilk Plus keywords cilk_spawn and cilk_sync. In order to model cilk_spawn and cilk_sync we will use following C recursive Fibonacci method.

Fibonacci method

Following is a C Fibonacci method. [27]

```c
double fib(int n)
{
    if (n < 2)
    {
        return n;
    }
    double x = cilk_spawn fib(n - 1);
    double y = fib(n - 2);
    cilk_sync;
    double z = x + y;
    return z;
}
```

In the given piece of code we are using cilk_spawn to make it runs in parallel. As shown in the code cilk_spawn keyword makes sure fib(n-1) is executing in parallel with fib(n-2) and cilk_sync indicates that this function will not continue until all cilk_spawn calls in same function have been completed. In other words, cilk_sync indicates that all spawned children must be completed before proceeding. [3, 27]
**Activity Models**
Since fork node and join node support the parallel flows in UML so we can model this parallel function using Fork node & Join Node in UML Activity. **Activity diagram by Microsoft Visio**

In this section we will provide Activity diagram for Fibonacci method drawn by Microsoft Visio.

![Activity Diagram for Fibonacci Method](image)

Figure 4.2 UML Activity diagram for Fibonacci method by Microsoft Visio.

Figure 4.2 depict diagram is an activity model of fib. The activity start and call fib function, this model consist of fork and Join nodes that shows how we are execution two fib function in parallel to calculate fib values for two different inputs. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all two parallel function calls are executed. After join node we will combine the results and print them.
**Activity diagram by Visual Paradigm**  
In this section we will provide Activity diagram for Fibonacci method drawn by Visual Paradigm.

![Activity diagram for Fibonacci method](image)

**Figure 4.3. UML Activity diagram for Fibonacci method by Visual Paradigm**

Figure 4.3 depict diagram is an activity model of fib. The activity start and call fib function, this model consist of fork and Join nodes that shows how we are execution two fib function in parallel to calculate fib values for two different inputs. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all two parallel function calls are executed. After join node we will combine the results and print them.
Activity diagram by Papyrus
In this section we will provide Activity diagram for Fibonacci method drawn by Papyrus.

![Activity Diagram](image)

Figure 4.4. UML Activity diagram for Fibonacci method by Papyrus

Figure 4.4 depict diagram is an activity model of fib. The activity start and call fib function, this model consist of fork and Join nodes that shows how we are execution two fib function in parallel to calculate fib values for two different inputs. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all two parallel function calls are executed. After join node we will combine the results and print them.
**PascalTriangleRow**

Following is another C function of pascalTriangleRow.

```c
void pascalTriangleRow(double num, double i){
    if (i < 0){
        return;
    }
    else if (i>=0){
        double fact1 = cilk_spawn factorial(num);
        double fact2 = cilk_spawn factorial(num - i);
        double fact3 = factorial(i);
        cilk_sync;
        double result1 = fact1/(fact2*fact3);
        printf("%f ", result1);
        i = i - 1;
        return pascalTriangleRow(num, i);
    }
}
```

In the above piece of code, *cilk_spawn* is used twice. *cilk_sync, cilk_spawn* instructions will make it run in parallel. As shown in the code *cilk_sync* keyword forces all the previous spawned tasks to wait for each other in order to be completed before the program continues. So results will be printed after all factorials will be calculated.

**Activity Model**

Since fork node and join node support the parallel flows in UML so we can model this parallel function using Fork node and Join Node in UML Activity.
Activity diagram by Microsoft Visio
In this section we will provide Activity diagram for “pascalTriangleRow” drawn by Microsoft Visio.

Figure 4.5 UML Activity diagram for pascalTriangleRow program by Microsoft Visio

Figure 4.5 depict diagram is an activity model of pascalTriangleRow. The activity start and call pascalTriangleRow function, this model consist of fork and Join nodes that shows how we are executing factorial functions for calculating three different factorial values fact1, fact2 and fact3 in parallel. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all three parallel function calls are executed. After join node these factorial values are used to calculate ith value of pascalTriangleRow and print the result, after printing the results we will decrements the value of i by one and make a call to the pascalTriangleRow function again.
Activity diagram by Papyrus
In this section we will provide Activity diagram for “pascalTriangleRow” drawn by Papyrus.

Figure 4.6. UML Activity diagram for pascalTriangleRow program by Papyrus

Figure 4.6 depict diagram is an activity model of pascalTriangleRow. The activity start and call pascalTriangleRow function, this model consist of fork and Join nodes that shows how we are executing factorial functions for calculating three different factorial values fact1, fact2 and fact3 in parallel. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all three parallel function calls are executed. After join node these factorial values are used to calculate ith value of pascalTriangleRow and print the result, after printing the results we will decrements the value of i by one and make a call to the pascalTriangleRow function again.
Activity diagram by Visual Paradigm
In this section we will provide Activity diagram for “pascalTriangleRow” drawn by Visual Paradigm.

Figure 4.7 depict diagram is an activity model of pascalTriangleRow. The activity start and call pascalTriangleRow function, this model consist of fork and Join nodes that shows how we are executing factorial functions for calculating three different factorial values fact1, fact2 and fact3 in parallel. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all three parallel function calls are executed. After join node these factorial values are used to calculate ith value of pascalTriangleRow and print the result, after printing the results we will decrements the value of i by one and make a call to the pascalTriangleRow function again.
Karatsuba multiplication program
Following is a C function to calculate multiplication of two numbers using Karatsuba algorithm. [29]

```c
void karatsuba_parallel(int c[], const int a[], const int b[], size_t n) {
    if (n <= Cutoff)
        { simple_mul(c, a, b, n);
            return;
        }
    size_t m = n / 2;

    // Set c[0:n-1] = t_0
    cilk_spawn karatsuba_parallel(c, a, b, m);

    // Set c[2*m:n-1] = t_2
    cilk_spawn karatsuba_parallel(c + 2*m, a + m, b + m, n - m);

temp_space<int> s(4 * (n - m));
int *a_ = s.data(), *b_ = a_ + (n - m), *t = b_ + (n - m);

    // initialize(*a_, *b_, *c_, m,n);
    for (size_t j = 0; j<m; ++j)
        { a_[j] = a[j] + a[m + j];
            b_[j] = b[j] + b[m + j];
        }
    if (n/2==1){
        a_[m] = a[2 * m];
        b_[m] = b[2 * m];
    }

    // Set t = t_1
    karatsuba_parallel(t, a_, b_, n - m);
    cilk_sync;

    // Set t = t_1 - t_0 - t_2
    for (size_t j = 0; j<2 * m - 1; ++j)
        t[j] = c[j] + c[2 * m + j];

    // Add (t_1 - t_0 - t_2) into final product.
    c[2 * m - 1] = 0;
    for (size_t j = 0; j<2 * m - 1; ++j)
        c[m + j] += t[j];
    if (n/2==1){
        for (size_t j = 0; j < 2; ++j)
            c[3 * m - 1 + j] += t[2 * m - 1 + j] - c[4 * m - 1 + j];
    }
}
```

In the piece of code, `cilk_spawn` is used twice. `cilk_sync`, `cilk_spawn` instructions will make it run in parallel. Figure 4.8 is an activity diagram of above Intel Cilk Plus code that provides you a better understanding of this code.
Activity diagram by Microsoft Visio

In this section we will provide Activity diagram for Karatsuba multiplication program drawn by Microsoft Visio.

Figure 4.8. UML Activity diagram for Karatsuba multiplication program by Microsoft Visio
Figure 4.8 depicts a diagram of the activity model of \texttt{karatsubaParallel}. The activity starts by calling the \texttt{karatsubaParallel} function. This model consists of fork and join nodes that show how we are executing three \texttt{karatsubaParallel} functions in parallel. The fork node will spawn all function calls, and the join node will ensure that the flow will not continue until all three parallel function calls are executed.
Activity diagram by Visual Paradigm
In this section we will provide Activity diagram for matrix multiplication program drawn by Visual Paradigm.

Figure 4.9. UML Activity diagram for Karatsuba multiplication program by Visual paradigm
Figure 4.9 depict diagram is an activity model of karatsubaParallel. The activity start and call karatsubaParallel function, this model consist of fork and Join nodes that shows how we are execution three karatsubaParallel function in parallel. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all three parallel function calls are executed.
Activity diagram by Papyrus
In this section we will provide Activity diagram for matrix multiplication program drawn by Papyrus.

Figure 4.10. UML Activity diagram for Karatsuba multiplication program by Papyrus
Figure 4.10 depict diagram is an activity model of karatsubaParallel. The activity start and call karatsubaParallel function, this model consist of fork and Join nodes that shows how we are execution three karatsubaParallel function in parallel. Fork node will spawn all function calls and Join node will make sure that flow will not continue until all three parallel function calls are executed.

4.2.2 cilk_for
By using cilk_for, we convert simple for loop into a parallel for loop [5]. In other word, we can execute iterations of loop bodies in parallel. In order to model this, we use expansion region class with stereotype <<parallel>> from UML activity classes. This class executes cilk_for loop in parallel.

intSum
Here is a small piece of code that calculate sum of N numbers it includes cilk_for keyword. [27]

```
int sum(int number){
    int sum = 0;
    cilk_for(int i = 0; i <= number; i++){
        Sleep(500);
        sum += i;
    }
    return sum;
}
```

We can model this using UML Activity diagram as shown in Figure 4.11.

Activity model by Microsoft Visio
Microsoft did not provide any support for expansion region in UML activity diagrams, and this implies that the system cannot be used when problem needs this type of functionality.

Activity model by Visual paradigm
In this section we will provide Activity diagram for intSum function, drawn by Visual Paradigm.
Figure 4.11 depict diagram is an activity model of intSum. This activity model consists of expansion region that shows how we are calculating sum of N numbers in parallel. Here we get an input as a collection of value, and we return the sum of those values.

**Activity model by Papyrus-eclipse**

In this section we will provide Activity diagram for intSum function, drawn by papyrus.

![Activity Diagram for intSum](image)

**Figure 4.12. UML Activity diagram for intSum function by Papyrus**

Figure 4.12 depict diagram is an activity model of intSum. This activity model consists of expansion region that shows how we are calculating sum of N numbers in parallel. Here we get an input as a collection of value, and we return the sum of those values.

**Bubble Sort**

Here is another C program that sort an array of elements using bubble sort algorithm, it includes cilk_for keyword.[30]

```c
void bubble_sort(int iarr[], int num) {
    int k, temp;
    cilk_for (int i = 1; i < num; i++) {
        cilk_for (int j = 0; j < num - 1; j++) {
            if (iarr[j] > iarr[j + 1]) {
                temp = iarr[j];
                iarr[j] = iarr[j + 1];
                iarr[j + 1] = temp;
            }
        }
    }
}
```

The above piece of code consist of two nested cilk_for loops. In order to model this, we use callBehaviourAction inside an expansion region class with stereotype <<parallel>>. This callBehaviourAction indicates that this invocation starts another activity class. This callBehaviourAction is having another expansion region class. The inside expansion region class will also execute in parallel.
Activity model by Microsoft Visio
Microsoft did not provide any support for expansion region in UML activity diagrams, and this implies that the system cannot be used when problem needs this type of functionality.

Activity model by Visual paradigm
In this section we will provide Activity diagram for bubble sort function, drawn by Visual Paradigm.

Figure 4.13. UML Activity diagram for bubble sort function by visual paradigm

Figure 4.13 depict diagram is an activity model of bubbleSort. This activity model consists of two nested expansion regions that show how we are sorting an array using bubbleSort algorithm. Outer expansion region go through input array in parallel, and inner expansion region compare every adjacent pair, swap their position if they are not in right order. Here we get an input as a collection of array values, and we return the sorted array.
Activity model by Papyrus - eclipse

In this section we will provide Activity diagram for bubble sort function, drawn by papyrus.

Figure 4.14. UML Activity diagram for bubble sort function by Papyrus

Figure 4.14 depict diagram is an activity model of bubbleSort. This activity model consists of two nested expansion regions that show how we are sorting an array using bubbleSort algorithm. Outer expansion region go through input array in parallel, and inner expansion region compare every adjacent pair, swap their position if they are not in right order. Here we get an input as a collection of array values, and we return the sorted array.
Prime numbers
Here is another piece of code that includes cilk_for keyword. This function count prime numbers between 1 to n.[31]

```c
int isPrime(int n){
    int limit = sqrt(n);
    for(int i = 2; i <= limit; i++){
        if (n%i == 0)
            return 0;
    }
    return 1;
}

int primeNumber(int n){
    n = n;
    int gs = n/400; //grainsize
    int numPrimes = 0;  int i;
    struct timeval start, end;

    cilk_for(i = 0; i <n/gs; i++){
        cilk_for (int j = i*gs+1; j<(i + 1)*gs; j+= 2){
            if (isPrime(j))
                numPrimes++;
        }
    }
    return numPrimes;
}
```

The above piece of code is consisting three nested loops. Two of those loops are cilk_for. The outer expansion region is having another expansion region class. The inside expansion region class will also execute in parallel. A while loop is running inside the nested expansion region that calculates number of prime numbers between 1 to N.

Activity model by Microsoft Visio
Microsoft did not provide any support for expansion region in UML activity diagrams, and this implies that the system cannot be used when problem needs this type of functionality.
**Activity model by Visual paradigm**

In this section we will provide an activity diagram for primeNumber function, drawn by Visual Paradigm.

![Activity Diagram](image)

**Figure 4.15. UML Activity diagram for primeNumber function by Visual Paradigm.**

Figure 4.15 depict diagram is an activity model of primeNumber. This activity model consist of two nested expansion regions that shows how we are counting number of prime numbers between 1 to N.
**Activity model by Papyrus- eclipse**
In this section we will provide Activity diagram for primeNumber function, drawn by papyrus.

![Figure 4.16. UML Activity diagram for primeNumber function by Papyrus.](image)

Figure 4.16 depict diagram is an activity model of primeNumber. This activity model consist of two nested expansion regions that shows how we are counting number of prime numbers between 1 to N.

### 4.3 Survey
In this section survey results will be presented. A survey was conducted to evaluate the appropriateness of UML activity models drawn for Intel Cilk Plus keywords.

#### 4.3.1 Participants
A survey was conducted among 15 students and researchers of computer sciences;
- 3 researchers of software architecture from Linnaeus University.
- 9 students of computer science and software engineering at Master level from different universities.
- 3 Students of final semester of software technology at Bachelor level from Linnaeus University.

You can get full information about the participants statistic in Appendix A.
4.3.2 Questions
This survey consists of four questions, different UML activity models were selected for those questions. These questions were design to collect the feedback of participants, so that we can evaluate our models. These questions were design to keep in mind the target participants, and the target participants were those who had good understanding of UML activity models programming languages (e.g. C/C++/Java). Follow is an example of question used in the survey:

![Image of code and UML diagram]

Figure 4.17. An example question used in the survey.

Figure 4.17 depict the survey question. On the left hand side is an Intel Cilk Plus code, on the right hand side is the UML activity model of Intel Cilk Plus codes. The answer for the question above the figure will be done in term of rating where one gets the rating scale right under the figure and by clicking one of the starts one submits a rating grade from 1 to 10.

The participants were asked to read the code and UML activity diagrams, and try to understand Intel Cilk Plus codes with the help of UML activity diagrams. In the end they should rate how much the provided model is helpful to understand Intel Cilk Plus codes. Full information about the survey can be found in Appendix B.
4.3.3 Survey results
In this survey different participants were asked to analyze the UML activity models with respect to Intel Cilk Plus codes, and rate the models. If a survey question is chosen multiple times with high rating it means it is more reliable, and if it get low rating multiple times, it means it has less reliability. According to survey results, the majority of participants said that question 3 and question 4 provides an excellent understanding of Intel Cilk Plus keywords cilk_sync and cilk_spawn. On the other hand for question 1 and 2 there opinion was quite divided, but according to most of them these models provide good understanding of Intel Cilk Plus key word cilk_for. The detailed results of the survey are in the following

![Survey Results Chart](image.png)

**Figure 4.18. Results of our survey in bar chart**

Figure 4.18 depicts our survey results. Here horizontal line shows how many times each question is selected and vertical line shows which question get what rating, user can provide rating between 1 to 10, 1 stand for minimum and 10 stands for maximum. Here blue line bar represent question 1, red line bar represent question 2, green line bar represent question 3 and purple line bar represent question 4.
5 Discussion
This section describes observations, advantages and disadvantages of each of the used tools. In the table below you will see advantages and disadvantages of each of the used tools.

5.1 Pros and Cons
Based on our UML activity models we made a comparison of different used tools in the form for advantages and disadvantages. In table 5.1 advantages and disadvantages of each of the used tools are shown.

Table 5.1. Advantages & disadvantages of tools

<table>
<thead>
<tr>
<th>Tools</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visio</td>
<td>Free for students</td>
<td>Limited support for UML Activity diagrams</td>
</tr>
<tr>
<td></td>
<td>Easy to install &amp; operate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Easy to find the feature we need.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Good for beginner user.</td>
<td></td>
</tr>
<tr>
<td>Visual Paradigm</td>
<td>Free for Academic use under certain condition.</td>
<td>Difficult to use for inexperienced / beginner users</td>
</tr>
<tr>
<td></td>
<td>Also available on monthly subscription with low price.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Provides full support for UML Activity diagrams.</td>
<td></td>
</tr>
<tr>
<td>Eclipse Papyrus</td>
<td>Is open source easy to install.</td>
<td>Difficult to use for inexperienced / beginner users</td>
</tr>
<tr>
<td></td>
<td>Provides full support to UML Activity.</td>
<td>Some versions did not provide full support of UML Activity diagrams.</td>
</tr>
</tbody>
</table>
5.2 Comparison of features
Based on our UML activity models we made a comparison of different features of the used tools, this comparison of different feature of the used tools is give in the table 5.2.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Microsoft Visio</th>
<th>Visual Paradigm</th>
<th>Eclipse Papyrus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Software License</td>
<td>Commercial</td>
<td>Commercial, Free Community Edition</td>
<td>Eclipse Public License *</td>
</tr>
<tr>
<td>2 Student Edition</td>
<td>Yes</td>
<td>Yes (under some conditions )</td>
<td>Yes</td>
</tr>
<tr>
<td>3 Support for expansion region</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>4 Guards of outgoing edge</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>5 CallBehaviorAction</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>6 Pin</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>7 Activity edge</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>8 Object Flow</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>9 Control Flow</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* | Papyrus 1.0.X / Eclipse Luna:
6 Conclusions
This section discusses the key observation and concludes some facts regarding the subject. Conclusion is made in two different directions separately: appropriateness of UML for graphical modelling of Intel Cilk Plus programs and comparison of different UML tools.

6.1 Intel Cilk Plus with UML activity
As we know today is the age of multi-core, and to take advantage of today’s multi-core hardware we can parallelize our code to distribute our work across this hardware. Intel Cilk Plus is a C/C++ language extension that helps us to write parallel programs. In this thesis activity diagrams are used for graphical modeling of Intel Cilk Plus. Different Intel Cilk Plus programs are used in order to evaluate different activity classes in our activity models. In chapter 4 we conclude with different good UML activity models, and those graphical models help readers to have a better understanding of parallel programs in general and Intel Cilk Plus specifically. Our activity models provide modeling behavior of Intel Cilk Plus programs, and describe the actual work flow of the used programs. As it is clear in this thesis, modeling parallel programming languages like Intel Cilk Plus with UML activities is a good approach in order to have a good understanding of parallel programing.

6.2 UML tools
In this thesis, we used our activity models created from Intel Cilk Plus codes to evaluate different UML drawing tools and we observed that among all three used tools; eclipse papyrus and visual paradigm for UML are best tools. They provide full support for UML activity diagrams. Here, eclipse papyrus has an advantage because it is free for all users and visual paradigm for UML is commercial. So we conclude that, with all pros and cons eclipse papyrus is the best tool for UML activity diagrams modeling.
7 Future work
In this section provides a discussion and suggestions for future work that can be done in this field in two directions in Intel Cilk Plus and in UML tools.

7.1 UML tools
The suggested future work for UML tools is that,

- In future one can select some different UML tools and evaluate them using Intel Cilk Plus
- In future someone can also evaluate UML tools by modeling Intel Cilk Plus with some other UML diagrams for example State diagram, Sequence diagram and Component diagram.

7.2 Intel Cilk Plus
In this thesis we modeled Intel Cilk Plus keywords using UML Activity diagrams. Now, the suggested future work for modeling Intel Cilk Plus is that,

- In future one can model Intel Cilk Plus keywords using some other UML diagrams
- In future one can extend this work by modeling reducers using some UML tools.
Reference


### Appendix

#### Appendix A: Survey participants

<table>
<thead>
<tr>
<th>No</th>
<th>Major Study</th>
<th>Educational Level</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software Architecture</td>
<td>Master of Software engineering from Linnaeus University (Sweden)</td>
<td>PhD student at Linnaeus University</td>
</tr>
<tr>
<td>2</td>
<td>Software Architecture</td>
<td>Master of Software engineering from Linnaeus University (Sweden)</td>
<td>PhD student at Linnaeus University</td>
</tr>
<tr>
<td>3</td>
<td>Software Architecture</td>
<td>Master of Software engineering from Linnaeus University (Sweden)</td>
<td>Was researcher at Linnaeus University</td>
</tr>
<tr>
<td>4</td>
<td>Software Engineering</td>
<td>Masters in Distributed Software Engineering from KTH</td>
<td>Royal Institute of Technology (Sweden)</td>
</tr>
<tr>
<td>5</td>
<td>Systems Engineering</td>
<td>Master of Science in Embedded Systems Engineering from University of Freiburg (Germany)</td>
<td>Working as a software engineer</td>
</tr>
<tr>
<td>6</td>
<td>Software Engineering</td>
<td>Master of Software engineering from Linnaeus University (Sweden)</td>
<td>Working as a software engineer</td>
</tr>
<tr>
<td>7</td>
<td>Software Engineering</td>
<td>Master of Software engineering from Linnaeus University (Sweden)</td>
<td>Working as a Software Engineer</td>
</tr>
<tr>
<td>8</td>
<td>Software Engineering</td>
<td>Bachelor's Degree, Software Engineering from University of Sargodha (Pakistan)</td>
<td>Studying Master of Software engineering from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td>9</td>
<td>Software Engineering</td>
<td>Bachelor's Degree, Software Engineering from KNURE (Ukraine)</td>
<td>Studying Master of Software engineering from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td>10</td>
<td>Software Engineering</td>
<td>Bachelor's Degree, Software Engineering From KhAI (Ukraine)</td>
<td>Studying Master of Software engineering from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td>11</td>
<td>Software Engineering</td>
<td>Bachelor of Software Technology from LNU (Sweden)</td>
<td>Studying Master of Software engineering from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td>12</td>
<td>Software Technology</td>
<td>Bachelor of Software Technology from LNU (Sweden)</td>
<td>Studying Master of Software engineering from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td></td>
<td>Software Technology</td>
<td>Some higher secondary education from hi/her home country</td>
<td>In final year of Bachelor of Software Technology from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td>---------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Software Technology</td>
<td>Some higher secondary education from hi/her home country</td>
<td>In final year of Bachelor of Software Technology from Linnaeus University (Sweden)</td>
</tr>
<tr>
<td>15</td>
<td>Software Technology</td>
<td>Some higher secondary education from hi/her home country</td>
<td>In final year of Bachelor of Software Technology from Linnaeus University (Sweden)</td>
</tr>
</tbody>
</table>
Appendix B: Survey

1. The snippet of code below is the parallel version of a normal C/C++ for loop and its UML activity model. Does this UML activity model provide a good understanding of Intel Cilk Plus keyword `cilk_for`? Please rate your answer.

   **Intel Cilk Plus code**

   ```c
   int sum(int number)
   {
       int sum = 0;
       cilk_for (int i = 0; i <= number; i++)
       {
           sum += i;
       }
       return sum;
   }
   ``

   **UML Activity model of given code.**

   This is your answer choice.

2. The snippet of code below uses `cilk_for` to count all the prime numbers up to N (i.e., N>2000) and its UML activity model. Does this UML activity model provide a good understanding of Intel Cilk Plus keyword `cilk_for`? Please rate your answer.

   **Intel Cilk Plus code**

   ```c
   int primeNumber(int n)
   {
       int isPrime = -1; // initialize
       int numPrimes = 0; // count primes
       cilk_for (int i = 0; i < n; i++)
       {
           if (i == 0 || i == 1)
               continue;
           numPrimes += (i * (i + 1)) == n;
       }
       return numPrimes;
   }
   ``

   **UML Activity model of given code.**

   This is your answer choice.
The following function is a parallel version of normal Fibonacci method with its UML activity model. Does this UML activity model provides a good understanding of Intel Cilk Plus keywords `cilk_spawn` & `cilk_sync`? please rate your answer.

**Init Cilk Plus code**

```cpp
double fib(int n)
{
    if (n < 2)
    {
        return n;
    }
    double x = cilk_spawn fib(n - 1);
    double y = fib(n - 2);
    cilk_sync;
    double z = x + y;
    return z;
}
```

This is your answer choice.

---

The following function is a parallel version of normal Pascal Triangular Row method that prints last row of a Pascal triangle, and its UML activity model. Does this UML activity model provides a good understanding of Intel Cilk Plus keywords `cilk_spawn` & `cilk_sync`? please rate your answer.

**Init Cilk Plus code**

```cpp
void pascalTrianglRow(int n, double r);
if (n < 0)
    return;
else if (n=1)
    double fact1 = cilk_spawn factorial(n);
    double fact2 = cilk_sync factorial(n - 1);
    cilk_sync;
    double result = fact1 / fact2 * fact1;
    printf("%d", result);
    i++; r = i;
    return pascalTrianglRow(n, r);
```

This is your answer choice.

---

* Please Enter your name.

[Finish Survey]
Appendix C: Execution time of programs that use *cilk_sync*, *cilk_spawn* and *cilk_for*.

Following programs have been tested:

- Fibonacci sequence
- Passcal Triangle Row
- Karatsuba Parallel
- Bubble Sort
- Int Sum
- primeNumber

All experiments have been done using following system:

1. Processor: Intel(R) Core(TM) i5-2435M CPU @ 2.4GHz 2.40 GHz
2. Installed memory: 8.00 GM
3. Windows edition: Window 7 home Premium Service Pack 1

<table>
<thead>
<tr>
<th></th>
<th>With Intel Cilk Plus cilk_spawn, cilk_sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>N*10 Fibonacci sequence</td>
<td>Threads 1</td>
</tr>
<tr>
<td>1</td>
<td>0.000151</td>
</tr>
<tr>
<td>2</td>
<td>0.000835</td>
</tr>
<tr>
<td>4</td>
<td>11.834355</td>
</tr>
<tr>
<td>N*10 Passcal Triangle Row</td>
<td>Threads 1</td>
</tr>
<tr>
<td>1</td>
<td>0.144040</td>
</tr>
<tr>
<td>2</td>
<td>0.202894</td>
</tr>
<tr>
<td>4</td>
<td>0.651525</td>
</tr>
<tr>
<td>9e+n*250 Karatsuba Parallel</td>
<td>Threads 1</td>
</tr>
<tr>
<td>1</td>
<td>0.044765</td>
</tr>
<tr>
<td>2</td>
<td>0.107773</td>
</tr>
<tr>
<td>4</td>
<td>0.124433</td>
</tr>
<tr>
<td></td>
<td>With Intel Cilk Plus cilk_for</td>
</tr>
<tr>
<td>( N \times 10000 )</td>
<td>Threads 1</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
</tr>
<tr>
<td><strong>Bubble Sort</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.285429</td>
</tr>
<tr>
<td>2</td>
<td>1.048748</td>
</tr>
<tr>
<td>4</td>
<td>4.873793</td>
</tr>
<tr>
<td><strong>Int Sum</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5.499681</td>
</tr>
<tr>
<td>2</td>
<td>10.500256</td>
</tr>
<tr>
<td>4</td>
<td>20.500327</td>
</tr>
<tr>
<td><strong>Count Prime Numbers</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.353255</td>
</tr>
<tr>
<td>2</td>
<td>0.919402</td>
</tr>
<tr>
<td>4</td>
<td>2.463847</td>
</tr>
</tbody>
</table>