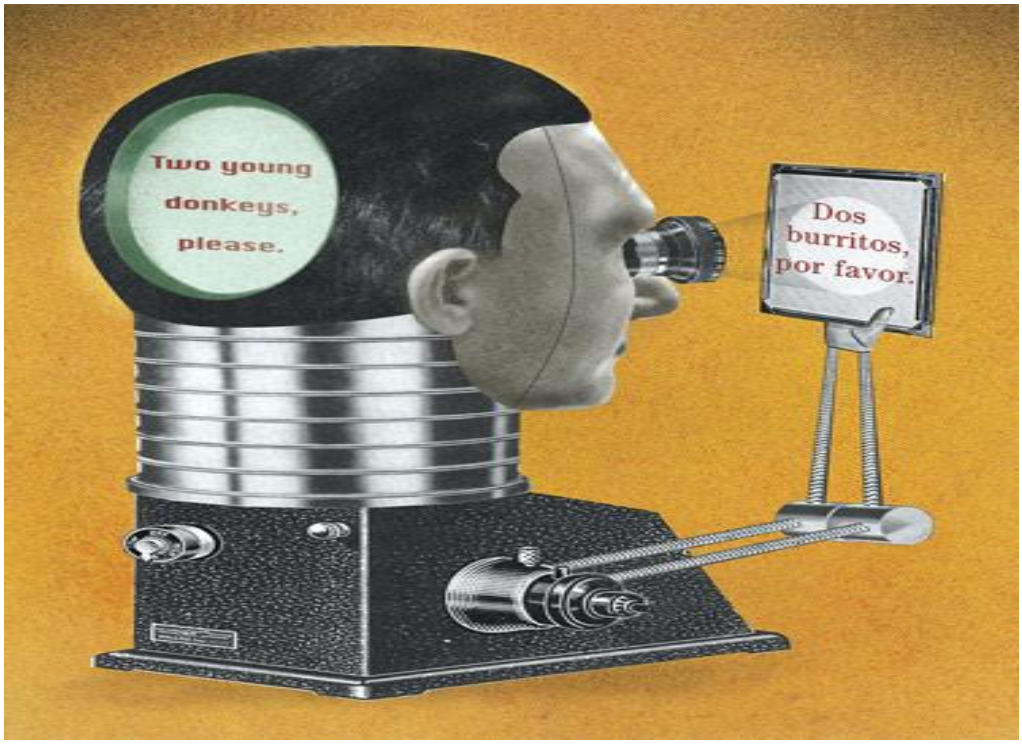
Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Occam & C++ Translator



Student Name: Shaoguang Miao

Student ID: C00131017

Supervisor: Joseph Kehoe

Table of Contents

Introduction

My project is Occam to C++ translator. It means the final software should translate the Occam programming language to C++.

Where the idea comes from?

Occam is a parallel programming language that makes it easy to write a program consisting of small parts (threads or processes) which could run concurrency. Additionally, implementing communication between those small parts is also not a difficult job. However, managing those small parts in C++ is not as easy as Occam. Because C++ is a more low-level programming language, the programmer who wants to write a parallel program using C++ needs to consider more things than an Occam programmer. Communication between those small parts is another difficult problem. Thus, we consider, if we can build a software which could translate Occam to C++, the programmer just need write the parallel part using Occam, then translate the Occam program to C++, the software development should be easier than writing a pure C++ program. That's the reason for the title-- Where the idea comes from.

Introduction to Parallel Programming

Parallel Programming is a kind of technology which could run multi-tasks on the multi-core computer at same time. It improves the program execution speed and makes calculation more efficiency than traditional program. Basically, programmers divide the program into sets of different function block, assign a thread (or process) for each function block and allow multi-threads (process) running on the different CPU at same time. There are serval ways to manage and schedule those function blocks. The detail will be given in Part 3 of this document.

Introduction to Occam

As I mentioned before, Occam is a parallel programming language which could separate the program into different part. Programmer could assign one thread for each part and manage those threads using special way in Occam. There is away to let threads communicate with each other – channel which is a special structure in Occam. Occam is a procedure programming language like

C. There are lots of special features which are not like some popular programming, such as C/C++, Java, etc. The detail will be given in Part 1.

Introduction to Compiler Tools

There sets of popular compiler tools (languages) existing. What they want to do is translate one kind of syntax to another grammar. The programmer could use compiler tools (languages) to build their own programming language. The first compiler language is Lex & Yacc which is developed for Linux. However, now, there are sets of tool we can use, such as ANTLR, Gold Parser, Grammattica, Spirit, and, Flex & Bison. The detail will be given in Part 2.

Introduction to Target Language (C++)

The Target Language is C++; it means the project product should translate Occam to C++. There are some threads algorithms in C++ standard library, but it's not AN efficient or convenient way to use. Fortunately, there are some the other kind of libraries from some the other developer we can use, like raw threads (POSIX threads, Windows threads), MPI, Open MP, etc. The detail will be given in Part 3.

Part 1 Occam

1.1Introduction

There are four different versions of the Occam: Occam 1, Occam 2, Occam 2.1 and Occam PI. Occam is a parallel programming language developed by David May at INMOS Limited (INMOS), Bristol, England. "*It's based on Tony Hoare's Communicating Sequential Process (CSP). The name is derived from William of Ockham of Occam's Razor frame. Occam's razor or the ancient philosophical principle of "keep things simple," is attributed to William.* "**[01]** The first version, Occam 1, has been obsolete. The most widely know version of those three is Occam 2. The Occam-pi is the newest version, now there are still lots programmers using Occam-pi to implement mobile programs. The most important different feature of the Occam is making concurrent programming simple. The programmer could implement the multi-processes working at same time easily.

1.2 Occam and Transputer

After the Occam Model, INMOS wanted to develop hardware to make their model more efficiency. This hardware has to be in the form of a very large scale integration (VLSI) integrated chip (IC). The Transputer was made following this idea.

In 1977, a collaborative project between Oxford University and INMOS Limited began work to design a microprocessor with input and output channels, making them easier to connect together to produce networks of processors of arbitrary size. In 1982, the transputer (TRANSistor comPUTER) emerged. For the T414 model the architecture consists of:

- *Reduced Instruction Set Computer (RISC) 32-bit processor*
- *Fast on-chip RAM (2 Kbytes)*
- *Hardware time-slicing features*
- *Four high-speed serial links (INMOS links)*

The T414 operates at 10 MIPs. Each of the INMOS links provides two Occam channels - one in each direction. Communication may occur concurrently with communication on

all other links. The synchronisation of processes at each end of the link is automatic and needs no explicit programming.

The transputer reflects the Occam model and can be considered to be an Occam machine and it provides a direct implementation of the occam process in its hardware. Although occam2 is not an assembly language, it provides full control of the transputer with regards system configuration, input/output and real-time interrupts and as such makes programming in native assembler unnecessary.

Until recently, the only complete Occam compilers were for transputers. This has restricted the use of the Occam language as transputers are not readily accessible. Compilers to allow Occam to be run to UNIX work-stations would facilitate portability between transputer and non-transputer systems. **[02]**

1.3 Different Versions OCCAM

There are four different versions of the Occam: Occam 1, Occam 2, Occam 2.1 and Occam PI-π. They are no big differences between them on the basic things. The high version also could run the low version's code. The high version just adds some new features in.

1.3.1 Occam 1

Occam 1 was the first version of this language. It's developed by David May at INMOS Limited (INMOS), Bristol, England. It come to the word is because the programmer wants to make their parallel programming easier. However, it's too much simple to implement all the work the people want to do.

This version just supported only the VAR data type, which an integral type was corresponding to the native word length of target architecture, and arrays of only one dimension. **[03]**

1.3.2 Occam 2

The Occam 2 was developed by INMOS Ltd in 1987. This version was added some new features which satisfy with the programmer to develop concurrent programs and which make the Occam become to a high level programming language. It not only added floating point, some data types,

like different sizes of integers (INT 16, INT 32), multi-dimensional arrays into this version but also functions. Those features give important information to the whole word that a new kind of programming language could be used. Occam was also known by the programmer since this time.

1.3.3 Occam 2.1

The next version of Occam language is Occam 2.1. It's also developed by INMOS, in 1994. Like Occam 2, it did change too much basic things, but added some new features. New features include: **[04]**

- Named data types
- Named Records
- Packed Records
- Relaxation of some of the type conversion rules
- Crate New operators (e.g. BYTESIN)
- Implemented channel retyping and channel arrays
- Function could return fixed-length array.

1.3.4 Occam Occam-π

Occam-π is final version for Occam. However it's not from INMOS either. Now, Kent Retarget developed Occam-π (or Occam-pi) as the variant of the Occam programming at the University of Kent. The introduction of elements of the π-calculus into Occam was reflected by its name, particularly concepts involving mobile processes and data. The language contains a significant number of extensions to Occam 2.1, including: **[05]**

- Build nested protocols
- Could create Run-time process

- Build channels, data, and processes for mobile
- Recursion are available for users
- Protocol inheritance
- Array constructors are implemented
- Extended rendezvous

1.4 The keywords of Occam

1.4.1 Primitive processes

Occam programs are built from processes. The simplest process in an Occam program is an action. An action is an assignment, an input or an output.

Assignment :=

The syntax is:

variable := expression

a,b,c := x, y+1, z+2

1.4.2 Communication

Communication is an essential part of Occam programming. Values are passed between concurrent processes by communication on channels. Each channel provides unbuffered, unidirectional point-to-point communication between two concurrent processes. The format and type of communication on a channel is specified by a channel protocol referenced in the declaration of the channel.

i. Input ?

An input receives a value from the channel on the left of the input symbol (?), and assigns that value to the variable on the right of the symbol. The value input must conform to the channel protocol and be of the same as the variable to which it is assigned, otherwise the input is not legal:

The syntax is:

channel ? variable

ii. Output

An output transmits the value of the expression on the right of the output symbol (!) to the channel named on the left of the symbol. The value output must conform to the channel protocol; otherwise the output is not valid.

The syntax is:

channel ! expression

iii. SKIP and STOP

The primitive process SKIP starts performs no action and terminates. The primitive process STOP starts, performs no action and never terminates.

1.4.3 Constructed Processes

i. Sequence

A sequence combines processes into a construction in which one process follows another.

The syntax is:

SEQ

{ Process}

A sequence can be replicated to produce a number of similar processes which are performed in sequence.

The syntax for a replicated sequence extends the syntax for sequences:

SEQ base expression for count expression

{Process}

ii. Conditional

A conditional combines a number of processes each of which is guarded by a Boolean expression. The conditional evaluates the Boolean expressions in sequence; if a Boolean expression is found to be true the associated process is performed, and the conditional terminates. If none of the Boolean expressions is true the conditional behaves like the primitive process.

Eg:

```
IF
  x > y
    order := gt
  x < y
    order := lt
  TRUE
    order := eq
```

iii. Selection

A selection combines a number of options, one of which is selected by matching the value of a selector with the value of a constant expression (called a case expression) associated with the option.

Eg:

```
CASE direction
  up
    x := x + 1
  down
    x := x – 1
```

iv. WHILE Loop

A WHILE loop repeats a process while an associated Boolean expression is true.

Eg:

```
WHILE buffer <> eof
  SEQ
    in ? buffer
    out ! buffer
```

This loop repeatedly copies a value from the channel in to out the channel. The copying continues while the Boolean expression

buffer <> eof is true. The sequence is not performed if the Boolean expression is initially false.

v. Parallel

A parallel combines a number of processes which are performed concurrently.

The syntax of a parallel is similar to that of a sequence:

```
PAR
    {Process}
```

vi. Alternation

An alternation combines a number of processes, only one of which is executed. Each of the combined processes is guarded by a guard which may or may not be ready to proceed. Examples of such guards are inputs on channels and delayed inputs on timer channels. The alternation performs the process associated with a guard which is ready.

Eg:

```
ALT
    left ? packet
        stream ! packet
    right ? packet
        stream ! packet
```

1.4.4 Data Type & Variable

Values are classified by their data type. A data type determines the set of values that may be taken by objects of that type and the set of operators which may be applied to objects of that type. These are the primitive data types built into Occam

BOOL	Boolean values true and false. A boolean type.
BYTE	Integer values from 0 to 255. A byte type.
INT	Signed integer values represented in twos complement form using the word size most efficiently provided by the implementation. An integer type.
INT16	Signed integer values in the range -32768 to 32767, represented in twos complement form using 16 bits. An integer type.
INT32	Signed integer values in the range -2^{31} to $(2^{31} - 1)$, represented in twos complement form using 32 bits. An integer type.
INT64	Signed integer values in the range -2^{63} to $(2^{63} - 1)$, represented in twos complement form using 64 bits. An integer type.
REAL32	Floating point numbers stored using a sign bit, 8 bit exponent and 23 bit fraction in ANSI/IEEE Standard 754-1985 representation. The value is positive if the sign bit is 0, negative if the sign bit is 1. A real type. The magnitude of the value is: $\begin{cases} (2^{(exponent-127)}) * 1.fraction & \text{if } 0 < exponent \text{ and } exponent < 255 \\ (2^{-126}) * 0.fraction & \text{if } exponent = 0 \text{ and } fraction \neq 0 \\ 0 & \text{if } exponent = 0 \text{ and } fraction = 0 \end{cases}$
REAL64	Floating point numbers stored using a sign bit, 11 bit exponent and 52 bit fraction in ANSI/IEEE Standard 754-1985 representation. The value is positive if the sign bit is 0, negative if the sign bit is 1. A real type. The magnitude of the value is: $\begin{cases} (2^{(exponent-1023)}) * 1.fraction & \text{if } 0 < exponent \text{ and } exponent < 2047 \\ (2^{-1022}) * 0.fraction & \text{if } exponent = 0 \text{ and } fraction \neq 0 \\ 0 & \text{if } exponent = 0 \text{ and } fraction = 0 \end{cases}$

Table 1. Basic Types in Occam [06]

Declaring a variable

The declaration of a variable associates its name with its data type.

The syntax for the declaration and use of variables of any data type is;

Data.type Variable.name

The declaration of an array with multiple dimensions is similar to other declarations:

[x][x] Data.type array.name

1.4.5 Channel and Channel Protocol

Occam programs act upon Variables , and communicate using channels and times. A variable has a value, and may receive a new value in an assignment or input. Channels communicate values. Timers produce a value which represents time.

A Communication channel provides unbuffered, unidirectional point-to-point communication of values between two concurrent processes, which are components of a parallel process or of constructions which are themselves components of a parallel program. The format and type of values passed on a channel is specified by the channel protocol. The name and protocol of a channel are specified in a channel declaration.

The keyword CHAN is always followed by the keyword OF which is followed by a protocol according to syntax elaborated below (page 47). Channel types cannot be named, but protocols can.

The syntax is:

CHAN OF protocol

Arrays of channels can be declared in the same way as arrays of variables:

[x]CHAN OF protocol (necessary?)

There is a subtle semantic distinction to be made between an array of data type and an array of channels. An array of variables is itself a variable, as it may receive a new value by assignment or input. However, an array of channels is not itself a channel, as only single components of the array may be used in input/output, but a means of referencing a number of distinct channels identified by consecutive subscripts. This distinction is not made in the description of the syntax of channels.

1.4.6 Channel Protocol

A channel communicates values between two concurrent processes. The format and data type of these values is specified by the channel protocol. The channel protocol is specified when the channel is declared. Each input and output must be compatible with the protocol of the channel used. Channel protocols enable the compiler to check the usage of channels, and to ensure the same effect whether the sending or the receiving process is ready to communicate first.

1.4.7 Timer

Timers produce a value which represents the time, and allow processes to be delayed until the time has reached or passed a particular value. The use of timers is essential in most real time control systems. A timer provides a clock which can be accessed by any number of concurrent processes. The relationship between the time returned by an Occam timer and real time is not defined by the language, i.e. implementations may differ in the granularity of timers and consequently in their cycle period.

A timer is declared in a manner similar to channels and variables.

Eg:

TIMER variable.name

[X] TIMER array.name

Another special timer input (called a delayed input) specifies a time, after which the input terminates.

Eg:

clock ? AFTER t

Explanation: This input waits until the value of the timer clock is later than the value of t. In other words, if c is the value of the timer clock, then the input will wait until (c after t) is true. The value of t is unchanged.

1.4.8 Procedure

The keyword PROC, the name of the procedure, and a formal parameter list enclosed in parentheses is followed by a process indented two spaces, which is the body of the procedure. Any data type may be used as a specifier following VAL, any type as a specifier otherwise. The procedure definition is terminated by a colon which appears on a new line at the same indentation level as the start of the definition.

The syntax for a procedure instance is:

```
PROC   name (parameters)
        Process
:
```

Eg:

```
PROC  increment (INT x)
        x := x + 1
        INT y:
        SEQ
         Increment(y)
```

Is equal to:

```
        INT y:
        SEQ
```

```
X IS y:

x:=x+1
```

1.4.9 Functions

A function defines a name for a special kind of process, called a value process. A value process produces a result of any data type, and may appear in expressions. Value processes may also produce more than one result, which may be assigned in a multiple assignment. Occam functions are free from all side effects, as they are forbidden to communicate or to assign to free variables. This helps to ensure that programs are clear and easy to maintain.

Eg:

```
INT FUNCTION sum (VAL []INT values)

INT accumulator :

VALOF

SEQ

accumulator := 0

SEQ i = 0 FOR SIZE values

accumulator := accumulator + values[i]

RESULT accumulator

:
```

This function definition defines the name for the associated value process. The type of the result of this function is INT. The result type or types, which may be any explicit or named fixed size data types appear in a comma separated list before the keyword function. For More information, please look at ***Occam 2.1 Manual.***

Part 2 Compiler Tools

2.1 Introduction

Compiling is one of the most important parts of the project. How to translate the Occam to C++ Programming Language is the point of the project. There are sevral different ways to solution the problem I mentioned. For example: Lex & Yacc, Antlr, Gold Parser, Grammattica, Spirit, and Flex &Bison. First two of them will be introduced briefly: Lex & Yacc, and Antlr. All of them derive from the Lex & Yacc which was produced by Stephen C.Johnson at AT&T for the Unix operating system.

2.2 Lex & Yacc

2.2.1 What the Lex & Yacc can do for us?

Lex & Yacc can help you to develop your programming language easily. Lex & Yacc can help you to write a compiler for your language. Lex & Yacc can read streams from the keyboard and translate to anything what you want.

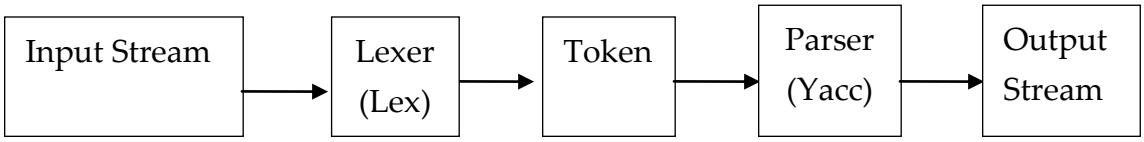
2.2.2 What are Lex & Yacc?

Briefly, Lex & Yacc is a kind of special programming language (compiler tool) which gets a stream of characters from the input and translates it to another format easily.

The program which has structured input can be divided into several different functional units by Lex & Yacc. Furthermore, the Lex and Yacc can print the context which we want on the screen according to the structured input. For C program, the units are variable names, constants, and so on. Lex will divide the program into sevRal different units. The units (which are usually called tokens) are produced by lexical analysis, or lexing for short. Lex also helps us taking the tokens and identifying them from the input stream. Lex source is a kind of table of the units (described using regular expression) and the corresponding fragments are output to Yacc. However,

the Lex can run BY ITSELF. We can get the whole token from the Lex, but we also need the relationship between. Yacc will help us to organise the tokens into new expression. Yacc will use a rule for those tokens and make them to expression according to the new grammar like parser.

Here is the diagram for flow of how they are working:



[Dig.1]

2.2.3 Example

There are two pieces of code IN Lex & Yacc shown below:

Lex:

```
%{
#include <stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+              return NUMBER;
heat                return TOKHEAT;
on|off              return STATE;
target              return TOKTARGET;
temperature         return TOKTEMPERATURE;
\n                  /* ignore end of line */;
[ \t]+              /* ignore whitespace */;
%%
```

Yacc:

```
commands: /* empty */
 | commands command
 ;
command:
 heat_switch
 |
 target_set
 ;
heat_switch:
 TOKHEAT STATE
 {
  printf("\tHeat turned on or off\n");
 }
 ;
target_set:
 TOKTARGET TOKTEMPERATURE NUMBER
 {
  printf("\tTemperature set\n");
 }
 ;
```

[07]

Brief explanation:

The Lex file defined serval tokens ([0-9], heat, on|off, target, temperature) to the corresponding fragments (NUMBER, TOKHEATM, STATE, TOKTARGET, TOKTEMPERATURE). The bottom two tokens don't have corresponding fragments, it means nothing will be happen when the Lex getS those. The Yacc file defined serval functions for the tokens passed by Lex.

command:
heat_switch
|
target_set

This piece OF code defined the command; it will call heat_switch or target_set.

```
heat_switch:
    TOKHEAT STATE
    {
        printf("\tHeat turned on or off\n");
    }    ;
```

If the “heat” “on (or off)” passed in Lex, Lex will translate them to “TOKHEAT” “STATE”, then pass them in the Yacc, Yacc will print “Heat turned on or off” on the screen.

2.3 ANTLR

2.3.1 What is ANTLR and what is ANTLR useful?

ANTLR stands for ANother Tool for Language Recognition. It’s also a compile tools which can use in translators, compilers, recognisers and static/dynamic program analysers. It’s like Lex and Yacc, but there only one separated Lexer and Parser by terminal and non-terminal symbols. The lexer rules will start with an uppercase letter and the parser rules with lower case letter. Like:

1. *declare : ’int’ ID ’=’ INT ’;*
2. *ID : (’a’..’z’ | ’A’..’Z’)+;*
3. *INT : ’0’..’9’+;*

First one is lexer rule and the parser rules are two of the rest. ANTLR mix lexer and parser together, it makes the rules simpler than Lex & Yacc, but it easy to confuse the programmer with those rules. Fortunately, it also can run if we separate lex rules and parser rules in two files.

2.3.2 What are the features of ANTLR?

Actually, the Lex & Yacc and ANTLR is kind of same tools designed for translate one language to the other. From the system supported level, the ANTLR could run on the every system easily, the Lex & Yacc can run base on the UNIX or Linux. From now on, there is lots of target language supported by ANTLR, like C Programming Language, Python, Object – C, Java, C#, and so on. For Lex & Yacc doesn't have too much libraries. In grammar, there are lots of same points between them, like basic operators:

- *() - Parentheses. Used to group several elements, so they are treated as one single token*
- *? - Any token followed by ? occurs 0 or 1 times*
- **- Any token followed by * can occur 0 or more times*
- *+ - Any token followed by + can occur 1 or more times*
- *. - Any character/token can occur one time*
- *~ - Any character/token following the ~ may not occur at the current place*

[08]

There is a piece of ANTLR code shown below:

```
grammar Expr;

@header {
package test;
import java.util.HashMap;
}
@lexer::header {package test;}

@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}

prog:   stat+ ;

stat:   expr NEWLINE {System.out.println($expr.value);}
```

```
    |  ID '=' expr NEWLINE
       {memory.put($ID.text, new Integer($expr.value));}
    |  NEWLINE
    ;

expr returns [int value]
    :  e=multExpr {$value = $e.value;}
       (  '+' e=multExpr {$value += $e.value;}
       |  '-' e=multExpr {$value -= $e.value;}
       )*
    ;
multExpr returns [int value]
    :  e=atom {$value = $e.value;} ('*' e=atom {$value *= $e.value;})*
    ;

atom returns [int value]
    :  INT {$value = Integer.parseInt($INT.text);}
    |  ID
       {
       Integer v = (Integer)memory.get($ID.text);
       if ( v!=null ) $value = v.intValue();
       else System.err.println("undefined variable "+$ID.text);
       }
    |  '(' e=expr ')' {$value = $e.value;}
    ;

ID  :  ('a'..'z'|'A'..'Z')+ ;
INT :  '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :  (' '|'\t')+ {skip();} ;          [09]
```

It almost seem as Lex and Yacc, but there is something the Lex and Yacc doesn't have.

```
 @header {
package test;
import java.util.HashMap;
}
```

```
@lexer::header {package test;}

@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}
```

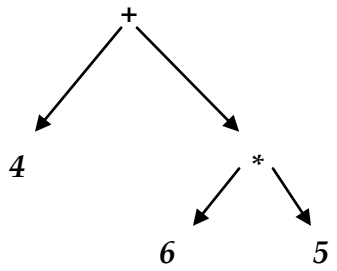
This is special part of the ANTLR named actions. It can control every piece of cod which is following the conditions in the whole project. There is the other kind of structure in ANTLR name Abstract Syntax Tree (AST). It could be set in the options command. The example of the AST will show below:

```
expression
: ^(unary_op expression)
| ^(CALL expression expressionList)
| ^(INDEX expression expression)
| primary
;
unary_op
:
UNARY_MINUS|UNARY_PLUS|UNARY_NOT|UNARY_BNO
T
;
primary
: ID
| INT
| FLOAT
| 'null'
;     [10]
```

Explanation:

^(unary_op expression)

It means build a AST whish use unary_op as root and expression as children. If the input is 4+6*5, the tree will be build like:



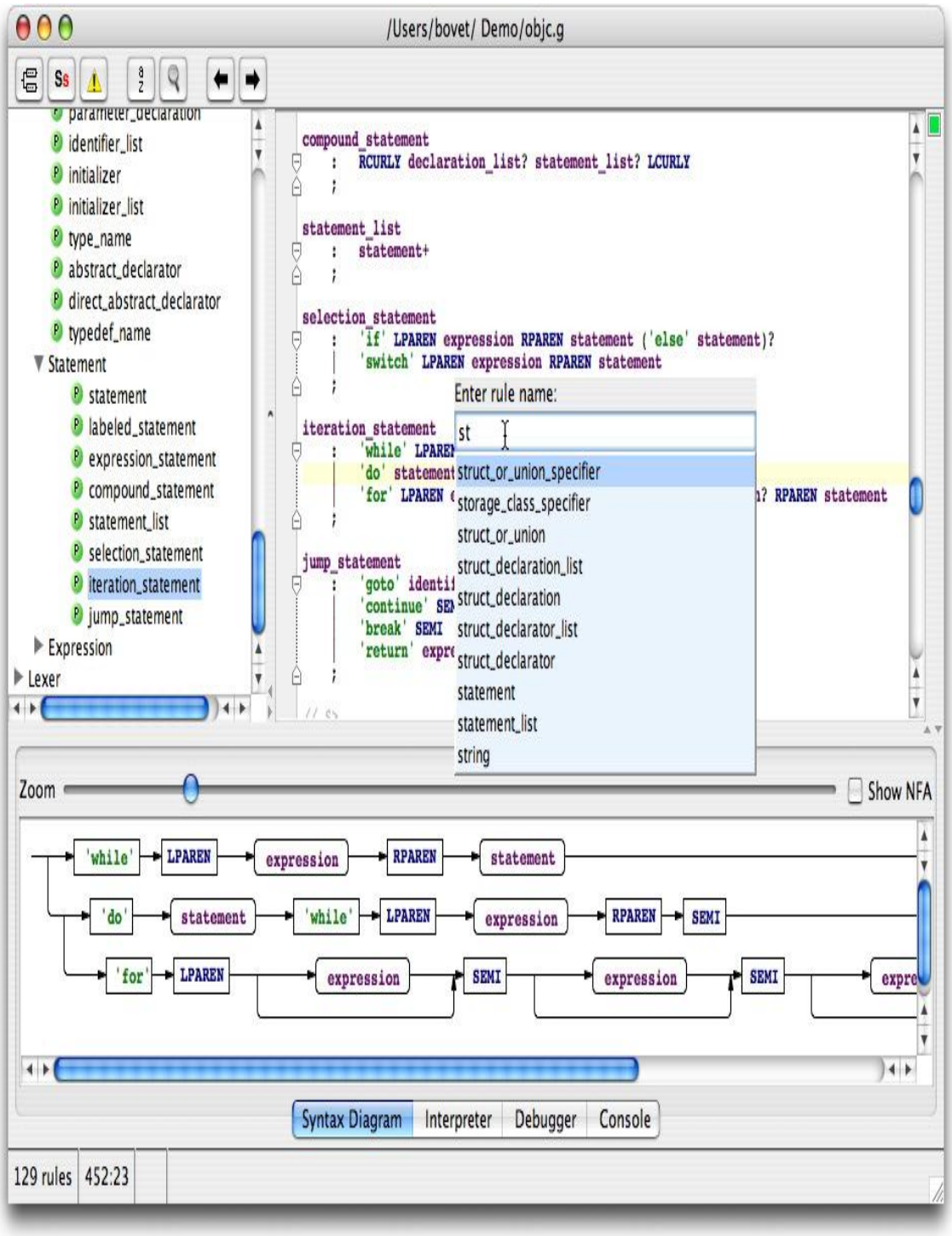
2.3.3 The environment for ANTLR

There are server ways to run ANTLR program. There is one plug in program for eclipse, but we just could write Java code for during the ANTLR. There is also a plug in program for Visual Studio. But the compiler I want to mention is ANTLRWorks. It's an individual compiler for ANTLR. The features will be shown in **[11]**:

- Only one window could use at one time
- highlight special syntax
- Build navigation tree for each rule
- User can jump to any rule or token definition
- Go To Rule
- Find usages of tokens or rules
- Find and Replace using regular expression
- Sensitive keyword, rule, and token auto-completion
- Fold rules and actions
- Tips and ideas
- Auto-indentation are available
- Refracturing - remove left recursion, extract or inline rule.
- Using special colour to show generated lexer or parser code
- The lexer and parser rule will be displayed by Syntax diagram
- Unreachable alternative(s) also will be shown in syntax diagram
- Display nondeterminism warning as ambiguous paths through the syntax diagram
- Decision DFA will be highlighted

- Rule dependency graph
- Export syntax diagram, NFA and parse tree to bitmap image or EPS file
- Perforce integration
- Maces key was blinded (Mac OS X only)
- Contextual menus
- Print

There is a diagram of the ANTLRWorks [Dig 2]:


/Users/bovet/ Demo/objc.g
identifier_list
initializer
initializer_list
type_name
abstract_declarator
direct_abstract_declarator
typedef_name
Statement
statement
labeled_statement
expression_statement
compound_statement
statement_list
selection_statement
iteration_statement
jump_statement
Expression
Lexer
compound_statement
: RCURLY declaration_list? statement_list? LCURLY
;
statement_list
: statement+
;
selection_statement
: 'if' LPAREN expression RPAREN statement ('else' statement)?
| 'switch' LPAREN expression RPAREN statement
;
iteration_statement
: 'while' LPAREN
| 'do' statement
| 'for' LPAREN
jump_statement
: 'goto' identi
| 'continue' SE
| 'break' SEMI
| 'return' expr
;
Enter rule name:
st
struct_or_union_specifier
storage_class_specifier
struct_or_union
struct_declaration_list
struct_declaration
struct_declarator_list
struct_declarator
statement
statement_list
string
Zoom
Show NFA
'while'
LPAREN
expression
RPAREN
statement
'do'
statement
'while'
LPAREN
expression
RPAREN
SEMI
'for'
LPAREN
expression
SEMI
expression
SEMI
Syntax Diagram
Interpreter
Debugger
Console
129 rules
452:23

[Dig.2][12]

2.4 The others

Gold Parser, Grammattica, Spirit, Flex & Bison are also famous compilers tool. But all of them are same like Lex & Yacc or ANTLR, the different is that they add more functions in their compiler or tools. There are list for comparison for all of the compiler tools shown in [Dig.3].Even Antlr didn't include any features which gold parser has, it also has lots of own features which will be given in next part.

Comparison Chart					
Feature	GOLD	ANTLR	Grammatica	Spirit	YACC / Bison
License	Free	Free	Free	Free	Free
Parsing Algorithm	LALR	LL	$LL_{(1)}$	LALR	LALR
Grammar Notation	BNF	EBNF	EBNF	EBNF	BNF
Grammar / Code	Independent	Mixed	Mixed	Mixed	Mixed
Parser Source Code	✓	✓	✓	✓	✓
Create Skeleton Programs	✓	✓	✓	✓	✓
Integrated Design Environment	GOLD	$ANTLR_{(2)}$	Grammatica	Spirit	YACC / Bison
Integrated Testing	✓	✓			
State Browsing	✓	✓			
Generate Webpages	✓	?			
Export to XML	✓	?			
Export to Formatted Text	✓	?			
Supported Languages	GOLD	ANTLR	Grammatica	Spirit	YACC / Bison
ANSI C	✓				✓
Assembly - Intel x86	✓				
C++	✓	✓	✓	✓	✓
C#	✓	✓	✓		
Delphi 5 & 6	✓				
DigitalMars D	✓				
Java	✓	✓			
Pascal	✓				
Python	✓	✓			
Visual Basic 6	✓				
Visual Basic .NET	✓				
All Other .NET Languages (3)	✓				
All Other ActiveX Languages (4)	✓				

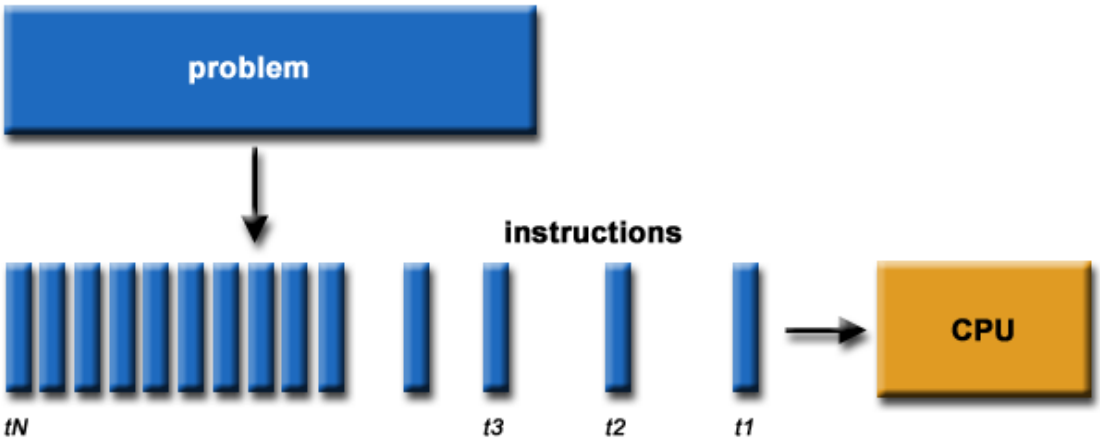
[Dig 3][13]

Part 3 Parallel Programming

3.1 What's parallel programming?

Multiple processors computer could handle many calculations at the same time. A large problem could be handled by dividing into some smaller parts, and many those part could run on the computer concurrently, that's we called parallel programming.

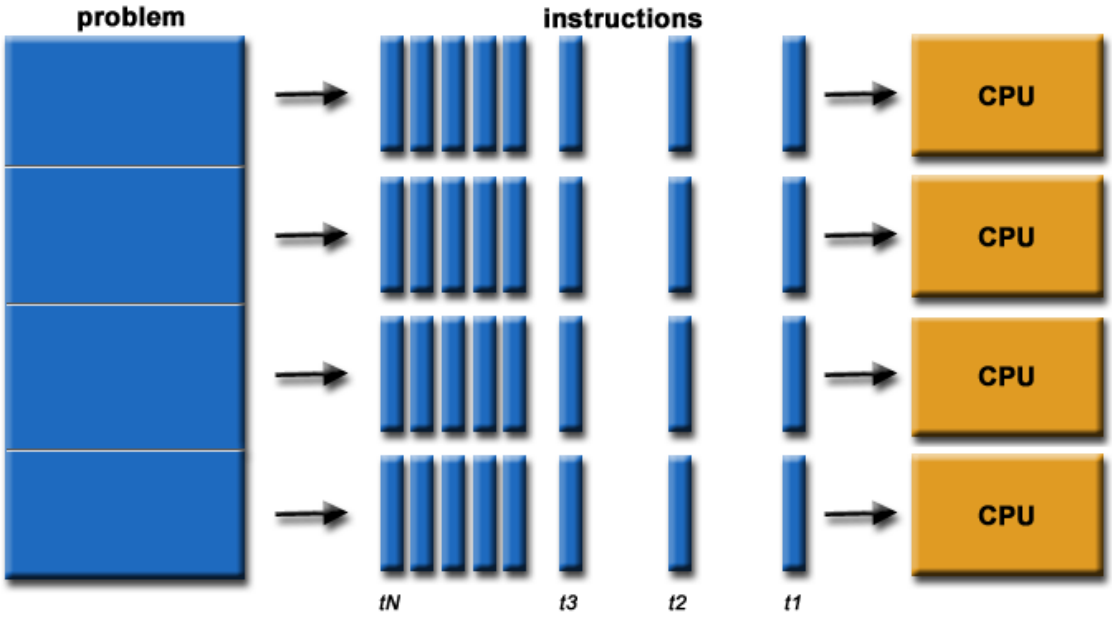
Traditionally, programmers write serial programs to be executed. The programs are written by a discrete series of instructions which will be executed one by one. The program could run on the single computer which just has one Central Processing Unit (CPU). Only one instruction which is in the program may just be able to be executed by CPU at any moment. The Dig.4 will show this type below:



[Dig.4][14]

Parallel program have lots of small parts which could get resource and calculation at any particular time. However, there is also some limitation of the parallel program. It only could run on the multiple CPUs computer, if it wants to run multiple parts at same time. Obviously, it also could run on the single CPU process normally. For multiple CPUs computer, lots of instructions which are not

in the same parts of executing program could be executed simultaneously on different CPUS. The Dig.5 will show this type below:



[Dig.5][15]

There are 4 different ways to perform parallel computing: bit-level, instruction level, data, and task parallelism. I will not talk about this part on this document. There are also lots models of parallel programming models in common use. These models include **[16]**:

- **Data Parallelism**
  This is where the same instructions are applied to different sets of data concurrently
- **Shared memory**
  Two or more processes can share one piece of memory when they are running.

- **Remote memory operation**
  Every process can access the memory from another process without its participations.

- **Message Passing**
  The processes can communicate with each other by sending and receiving messages.

- **Thread**
  One single process could have multi-thread which could execute indecently.

- **Combined models**

  It composed of two or more models of the above

I will give some details about threads and Message Passing in this document.

3.2 Parallel Programming Models

3.2.1 Message Passing Model

i. What is the Message Passing Model

Message Passing Model is base on multiple processes which has multiple objects run concurrency (multiple tasks). Each object has its own memory space for computation. They communicate with each other by messages. One object can request another object to do something by sending messages. All those messages convey some form of information and always pass argument back and forth. There are two types of message definition shown in **[17]**:

- **There are Meilir three types of messages are defined by Page-Jones:**
  1. Informative - send update information to one object.
  2. Interrogative - ask some reveal information from one object
  3. Imperative - take some action on one object, or another object

- **There are four types of messages are defined by Grady Brooch :**

1. Synchronous - receiving object is ready; it will start only when it receives a message from a sender.
2. Balking - if the receiving object is not ready to accept the massages sent from sending object, sending object will give it up.
3. Timeout - sending object will not wait for more than a certain time period for the receiving object to be ready to accept the message.
4. Asynchronous - whether the receiver is ready to receive messages, the sender can send a message to receiver without any more considers.

ii.Implementations of Message Passing:

Lots of operations that are allowed by an implementation of message passing model from the component of a message passing library, such as PICL, PVM, PARMACS, P4, MPI, etc. They don't target a special platform for using. The detail will not be given in this document except MPI. However, there is a list [18] which measure latency and bandwidth for the MPI, MPL and PVMe/PVM libraries on the SP2. The table shown that the MPI will run faster than any the other kind of way. The MPL is close to it.

	latency (microseconds)	*bandwidth (Mbytes/s)*	*portable*
MPI-F	*43*	*34*	*yes*
MPL	*45*	*34*	*no*
MPICH	*58*	*33*	*yes*
*PVMe**	*83*	*31*	*yes*
PVMe	*220*	*27*	*yes*
*PVM***			
RouteDirect	*642*	*12*	*yes*
DontRoute	*1450*	*3*	*yes*

** interrupts off*
*** in place packing*
MPI-F = prototype IBM MPI
MPICH = ANL/MSU MPI **[18]**

iii. MPI (Message Passing Interface)

MPI is the most typical form for a Message Passing Model. It is not a compiler, Language specification or a specific product or implementation. It is a kind of specification for programmers not a library itself but rather the specification of what such a library should be. The goal of the MPI is to provide a more widely and normally used standard for the programmer who want to write message passing programs. The MPI interface specification has been defined for C/C++, Fortran Programs and attempts to be a practical, portable, efficient and flexible interface.

The features of MPI are: **[19]**

- General
  -- Guarantee to Communicators combine context and group for message security
  -- Guarantee to Thread safety

- Point-to-point communication
  -- Buffers and derived data types was structured.
  -- There are four modes for point-to-point communication: normal (blocking and non-blocking), synchronous, ready (to allow access to fast protocols), buffered.

- Collective
  -- Both built-in and user-defined are collective operations
  -- There are large number of data movement routines
  -- Subgroups will be defined directly or by the other kind topology

iv. Advantages of MPI

The advantages of MPI as stated in **[20]** are shown below:

- Standardization - MPI is a standard message passing library which is supported on virtually all HPC platforms and has replaced all previous message passing libraries.
- Portability –The source code which using the MPI standard could run on a different platform without any modification.

- Performance Opportunities - Vendor implementations should be able to exploit native hardware features to optimise performance.
- Functionality – MPI-1 defined Sets of routines alone.
- Availability - both vendor and public domain implementations are available for the users.

3.2.2 Threads Model

Compare MP Model, I rather than Thread Model. The detail of thread model will be shown after.

i. What is a thread?

We have to known what is a thread before us doing the parallel programming. "*A thread is a single sequence stream within in a process.*" **[21]** A thread is kind of structure of the operating system like process, but smaller. Lots of people call it "light process". There are two types of thread user-level and kernel-level threads. The detail will not be given in this document (*Operating systems internals and design principles* ***(See reference [23])*** gives all of this information citation). So what's the different between process and threads?

ii. Differences between process and threads.

A process is created by operating system which will handle program actions. Process contains the information about program resource and program execution state. It has:

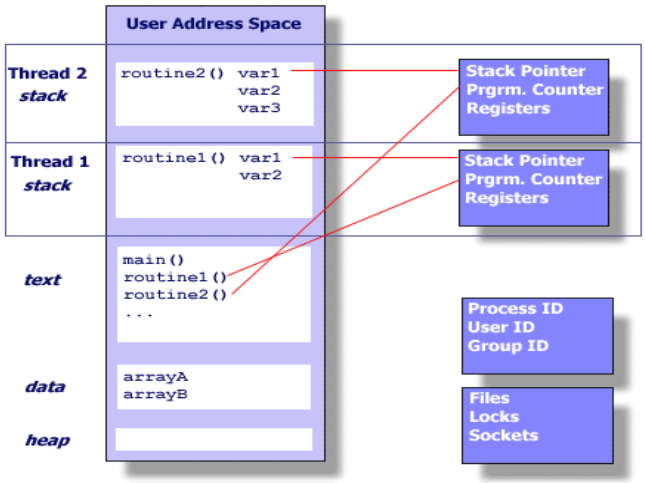
- Process ID
- Environment
- Working Directory
- Program instructions
- Registers
- Stack
- Heap
- File Descriptors
- Signal actions
- Shared Libraries

- Inter-process communications tools

Threads use and exist within process, but those could run independently in the operating system. Like process, the thread also is an entity which includes:

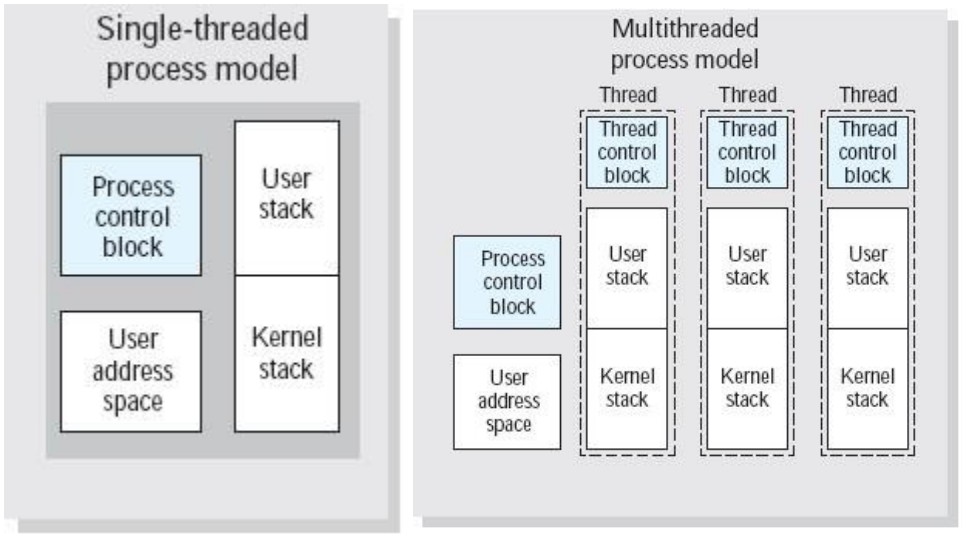
- Thread ID
- Stack pointer
- Registers
- Scheduling properties (long term, short term, etc)
- Set of pending and blocked signals
- Thread specific data.

Unlike process, threads did have their own memory. All of threads which are in same process share same memory space. And the all those threads are also share the same resource. Like the Dig.3 shown below [Dig6]:



[Dig.6][22]

In this model, multithreads could run as the small part of program in multiprocessors computer. Here, we just call more than two threads which belong to same process multithreads. Multithreaded process model and single-thread process model shown below **[Dig.7]:**



[Dig.7][23]

iii. Implementations:

From a programming perspective, threads implementations include:

- A Library which contains parallel source code
- A set of compiler embedded parallel source code

There are lots of compilers embedded parallel source codes, like Visual Studio, Eclipse, QT, Netbeans, Open MP, etc. There are just a few libraries which are professional for parallel programming, like POSIX Threads (Pthreads), Threading Building Blocks (TBB), Boost Threads, etc. The detail of TBB will be given following.

TBB

Threading Building Block (TBB) is a C++ library which was developed by Inter Corporation for writing program that has multiple threads.

"The library consists of data structures and algorithms that allow a programmer to avoid some complications arising from the use of native threading packages

such as POSIX threads, Windows threads, or the portable Boost Threads in which individual threads of execution are created, synchronized, and terminated manually." **[24]**

TBB doesn't require any special languages or compilers; it could run on the compiler which supports ISO C++. Programs writing by TBB could run on the single processor system, as well as multiple processors. Additionally, it fully supports nested parallelism, so you can build a large parallel program by putting sets of smaller component together although it's not a good way to do. As a high level library, the goal of the TBB is move programmer's focus from threads to tasks; it means that the programmer will think what to do instead of how to do it during development.

Advantage's of TBB **[25]**

1. *Threading Building Block enables you to specify tasks instead of threads.*

   Because threads are low-level, heavy constructs that more close to the hardware; it's very hard to control. The programmer who uses level normal threads has to map logical tasks on to thread. However, the Threading Building Block provides a kind of automatically schedules tasks on to threads. It makes source code easier to be written and understand. More important, it changes the design opinion of programmer from how to do to what to do.

2. *Threading Building Block targets threading for performance.*

   Most general-purpose threading packages support many different kinds of threading; the result of this opinion is making those packages to be solution rather than a foundation. Nevertheless, the TBB focuses on particular goal of parallelising computationally intensive work, simpler solutions.
3. *Threading Building Block is compatible with other threading packages.*

4. *Threading Building Block emphasizes scalable, data-parallel programming.*

In multithreads programming, we divide our program into many different functions block and assign a separate thread to each function block. Because of the number of function is fixed, so the thread cannot be scaled easily. In contrast, TBB emphasizes data-parallel programming, it allow using multiple threads to work most efficiency together. Besides, there are sets of algorithm for avoiding some classic problem of multithreads, such bottlenecks.

5. *Threading Building Block relies on generic programming.*

iv. Comparison with other parallelism

Whatever using raw thread interface, such as POSIX thread (pthreads), Windows threads which are using shared memory parallelism, Boost Threads which are very portable raw threads interface or MPI which suit for amount of threads run concurrence, the programmer have to manage every detail of each thread. Raw threads and MPI represent the assembly languages of parallelism which offer maximum flexibility, but at a high cost in terms of programmer effort, debugging time, and maintenance costs. However, Threading Building Block provides sets of way to map tasks onto processor cores explicitly instead of controlling each thread. "*With TBB, programmers could express more concurrency and finer-grained concurrency than would be possible with threads, leading to increased scalability.*" **[26]** Additionally, TBB is a library which could run on different platform easily, such Windows, Mac, Linux/Unix.

3.3 Suitable Development Languages

There are sets of development languages we can use for developing. I just want to compare three of them C, C++, and Java in this document.

3.3.1 Introduction to Java Programming Language

Java was developed by James Gosling at Sun Microsystems and released in 1995. As an Object-Oriented Programming Language, it become more and

more popular since it appeared. It's the most close to human language in those three programming languages. It means the developer could use Java to develop software will be easier than the others.

The goal of Java is making development easier. Actually, it got its goal in those years. Java has its own mechanism for collect rubbish during the program is running. When a programmer creates a variable, Java will allocate a piece of memory for it. Java will free the space that is no longer to be referenced automatically. The Garbage Collection system will implement to free the memory that hasn't been referenced in the program. The programmer also needn't to worry about the problem like which element the pointer is point to. They need not to free the space that he just allocates, because the pointer disappeared in the Java.

The most important feature of Java should be Platform-Independent. Java could run across the different operating system and hardware platforms which support JVM (Java Virtual Machine) without writing and compiling twice. This is the biggest different between Java and the other programming language. The reason that Java could run across multiple platforms is the origin Java code is not actually running in the CPU but JVM. JVM is a translator that could translate the Java Byte code to the machine code that the system and hardware could recognize. It means that Java code doesn't be translated into machine code by compiler but Java byte code which could run on the JVM. It's exactly like anything has two sides- wonderful and terrible, Java code doesn't be translate into machine code immediately but Java Byte Code, it become slower than the other languages.

3.3.2 Introduction to C and C++ Programming Language

Compare to Java and C++, C programming language is close to hardware. It means it could run on the CPU extremely efficiency and could save more time for users. C is a procedure programming language which the programmers have to write every piece of detail in program. Like Java C++ is also an Object-Oriented Programming Language. Although, they are even different type of programming language, they still have lots of commons. That's the reason I want to put them together to introduce.

Because of C/C++ are closer to hardware, they are control hardware easier than Java. C/C++ Programming Language are just like a bridge from the programmer to the hardware of the computer. They just need the compiler to translate what the programmer means, and then the computer will do the thing the programmer wants to. Besides, they do not rely on the compiler or something else, because they just care about the hardware of the computer. Once the code run in one machine, it could easy to adapt some different machines. It will be no problems if the basic thing is the same.

The most important features in C/C++ Program Language would be Pointers. Pointer is a very useful and complex data structure. The value's address is stored in the pointer. The pointer could store not only the basic data type's (int, float, double, etc.) address, but also the location of the function and array.

*"The run-time representation of a pointer value is typically a raw memory address (perhaps augmented by an offset-within-word field), but since a pointer's type includes the type of the thing pointed to, expressions including pointers can be type-checked at compile tim*e." **[27]**

The pointer almost could be used wherever and any types the programmer wants. It would be more flexible than the other language.

3.3.3 Comparison C, C++ and Java on Thread Control

All of them have their own thread library. All of those libraries provide thread creating, deleting and controlling. They are easily to be used for programmers. However, all of them are not very well to implement how to communication between threads. They didn't define a very well strategy to handler passing information and control each other in their own library. For Occam Translating, the channel (which have introduced in part 1) is an important part. How to handle this part is one point of this project. Fortunately, I got some extra libraries from Internet, which has been introduced in part 2 of this document. For C programming Language, TBB doesn't support it now but Java and C++ could use it easily. Between Java and

C++, I prefer C++ because it's faster and flexible. For the fist vision project, I won't let it to run on the system beside Windows. On the other hand, I have learned C programming for a couple of yeas. It's easier to use C++ than Java. Thus C++ would be my first choose. As I mentioned before (Part 2), C++ would not support C++ but C. Thus, actually, C and C++ this would be mixed using together in the project.

Conclusion

The project (Occam, C++ Parser) will translate the Occam which is a parallel programming language to C++. We want to decrease the development time for C++ programmers who want write a parallel program. We want to help programmer work more efficiency on parallel programming.

This document introduced what's the parallel programming (in part 3), what's Occam (in part 1), what's and how to use compiler tools, which way to implement parallel programming language is better and easier in C++. The detail of Occam, like how the Occam works and the most key words of Occam (in part1) were introduced in this document. For compiler tools, ANTLR was the most popular and important in this time, a lot of information of ANTLR was given in part 2, like how ANTLR works, why I want to use ANTLR to be parser language in my project. Additionally, Lex & Yacc is also another important parser language for introducing, because it's the original model of ANTLR. Besides, the comparison of different parser language and tools was also given. Finally, the target language libraries introduction has been given in part 3. Two different ways, message passing and threads are the point of target libraries. Especially, TBB which was developed by Intel is the most important part of target libraries. Hopefully, I can use TBB to implement parallel programming in C++.

Reference

[01] Dr.Daniel C.Hyde. 1995. *Introduction to the Programming Language Occam*.

www.eg.bucknell.edu/~cs366/occam.pdf

[02] Martin Stuart Mamo, 1995

http://freespace.virgin.net/martin.mamo/dissert.html#intro

[03] WIKIPEDIA, 2009

http://en.wikipedia.org/wiki/Occam_%28programming_language%29

[04] WIKIPEDIA , 2009

http://en.wikipedia.org/wiki/Occam_%28programming_language%29

[05] WIKIPEDIA, 2009

http://en.wikipedia.org/wiki/Occam_%28programming_language%29

[06] SGS-THOMSON Microelectronics Limited,1995.

http://wotug.org/occam/documentation/oc21refman.pdf

[07] Bert Hubert ,2001

http://ds9a.nl/lex-yacc/cvs/lex-yacc-howto.html

[09] Johannes Luber ,2008

http://www.antlr.org/wiki/display/ANTLR3/Quick+Starter+on+Parser+Grammars+-+No+Past+Experience+Required

[09] ATNLR V3

http://www.antlr.org/works/help/tutorial/calculator.html

[10] Terence Parr, 2007

The Definitive ANTLR Reference Building Domain-specific Language

[11] Jean Bovet

http://www.antlr.org/works/index.html

[12] Terrence Parr. Antlr

http://www.antlr.org/works/screenshots/editor.jpg

[13] Gold Parser

http://www.devincook.com/goldparser/about/comparison-parsers.htm

[14] Blaise Barney, 2009 *Introduction to Parallel Computing*

https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[15] Blaise Barney, *2009. Introduction to Parallel Computing*

https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[16] Maui High Performance Computing Center,1996

http://www.hku.hk/cc/sp2/workshop/html/message_passing/message_passing.html#message1

[17] Devdaily.com

http://www.devdaily.com/java/java_oo/node14.shtml

[18] Maui High Performance Computing Center ,1996
http://www.hku.hk/cc/sp2/workshop/html/message_passing/message_passing.html#message2

[19] William Gropp *The Message-Passing Interface*

http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/node16.html#Node16

[20] Blaise Barney ,*2009*

https://computing.llnl.gov/tutorials/mpi/

[21] kent

http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/threads.htm

[22] Blaise Barney, 2009, Introduction to Pthreads

https://computing.llnl.gov/tutorials/pthreads/

[23] William Stallings.2005. *Operating systems internals and design principles (Fifth Edition)*. Prentice Hall, 2005

[24] WIKIPEDIA, 2009

http://en.wikipedia.org/wiki/Intel_Threading_Building_Blocks

[25] Inter Corporation

http://www.threadingbuildingblocks.org/uploads/81/91/Latest%20Open%20Source%20Documentation/Tutorial.pdf

[26] James Reinders, 2007

Inter Threading Building Block—Out fitting C++ for Multi-core Processor Parallelism

[27] WIKIPEDIA, 2009

http://en.wikipedia.org/wiki/C_(programming_language)#