Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 10, October 2013, pg.115 – 125

SURVEY ARTICLE

Analysis of Parsing Techniques & Survey on Compiler Applications

Ch. Raju[1], Thirupathi Marupaka[2], Arvind Tudigani[3]

[1,2,3] Assistant Professor, Osmania University, Hyderabad, India

Abstract - Designed to provide good system programming in semantic and syntactic is the process of compiler design. Any program written in a high level programming language must be translated to object code before going to be executed. Compiler is the need to design and connectivity between the hardware and software process, learning English grammar provides a precise way to specify the syntax and meaning of a language to speak and write proper English. A grammar in compiler is a set rule that specify how sentences can be structured with the terminals, non-terminals and the set of productions. Code generation for embedded processors is the design of efficient compilers for target machines, we describes the application specific features in a compiler and backend design that accommodates these features by means of compiler register allocation and supports the embedded systems and media applications. This analysis presents the techniques of compiler design and also design of network processor and embeds system, compiler not only translates the information can be used for the processor design.

Keywords – Compiler; Parsing; Processor; Networks; Grammars

I INTRODUCTION

Since digital computers are programmed using a complex system of binary codes and memory addresses, user interface is the mechanism through which the user of a device communicates with the device. Assembly language of instructions that the computers CPU are capable of executing. There are often instructions which do some kinds of operation like add the two numbers stored in memory, move numbers from one location in memory to another, move information between the CPU and memory. Language which permit complex operations or expressions such as high-level language or programming languages. A compiler accepts as input a program written in any high-level language and produces as output an equivalent program in machine understandable language is target language. The process

contains two language one is source program and other is object program have many benefits of high-level language over machine or assembly language.
Most innovation that is important to the next major boost of microprocessor performance is instruction level parallel processing reflected by the fact that high performance microprocessors are increasingly designed to exploit instruction level parallelism. Mainstream microprocessors in 1990 executed one instruction per clock cycle those in 1995 execute up to four instructions per cycle.
Traditionally optimizing compilers improve program execution speed by eliminating unnecessary instruction processing by keeping memory data and intermediate computation results in high speed processor registers optimising compilers reduce the program execution cycles spent waiting for the memory system and performing redundant computation. The quality of compiler parallelization technique can potentially make an order of magnitude difference in program execution performance for processors that exploit. For a processor that executes up to eight instructions per cycle parallelized program can run at several times the speed of a poorly version of the same program.

In general interpreter needs to understand only the source language of translating the source text an interpreter immediately executes the instructions in the source text. Many languages are usually interpreted either directly or after a compilation to some virtual machine code. Lisp Smalltalk Prolog, SQL are among those. Advantage of interpreter are that is easy to port a language to a new machine all one has to do is to implement the virtual machine on the new hardware. Compiler have wider applications than just translating programming languages conceivably any large application might define its own command language which can be translated to a virtual machine associated with the application. Using compiler generating tools defining and implementing such a language need not be difficult SQL can be regarded as such a language associated with a database management system.

II Related Work

2. Related Work: In one of research performance characteristics of data mining applications using the Nu-Mine Bench written in C/C++ with the exception of AFI GETI are parallelized with open directives. Choose the example of Intel as the order architecture processor for our study for two main reasons, first Itanium arguably relies on sophisticated compiler optimizations to achieve its performance making the proper platform for evaluating the effectiveness of compiler directed prefetching. Itanium sets a hardware performance counters that enable us to study performance characteristics without cumbersome simulations or intrusive instrumentations.

2.1. Phases of Compiler: In order to construct process the compiler is implemented in phases lexical analysis syntax analysis code generation.

Lexical analysis in grouping the input string into words known as a lexeme or lexical token is string of input characters which is taken as a unit and passed on to the next phase of compilation.

Key words - While, Void, if. For
Identifier - Declared by the programmer
Operator - - +,-,*,/,=,.

The output of the lexical phase is a stream of tokens corresponding to the words, builds tables which are used by subsequent phases of compiler. Symbol table stores all

© 2013, IJCSMC All Rights Reserved

identifiers used in the source program including relevant information and attributes of the identifiers.

Syntax Analysis Phase is a parser is critical to understand both phase and the study language in general will check for proper syntax issue appropriate error messages and determine the underlying structure of the source program.

Syntax tree each interior node represents an operator or control structure and each leaf node represents an operand. A statement if (expr) stmt1 else stmt 2 could be implemented as a node having three children one for conditional expression one for the true part and one for else statement. While control structure would have two children one for loop condition and one for statement to be repeated, compound statement could have an unlimited number of children one for each statement in the compound statement.

Assembly language aware that the computer is capable of executing only a limited number of primitive operations on operands with numeric memory addresses all encoded as binary values. Code generation translated to machine language instructions or assembly language which translated to machine language instruction in which the assembler invoked to produce the object program. For target machines with several CPU registers the code generator is responsible for register allocation.

Local optimization is optional is needed only to make the object program more efficient which involves examining sequences of instructions to find unnecessary or redundant instructions. Local optimization is often called machine dependent optimization source program result in Load operand into a register, add the other operand to the register, store the result.

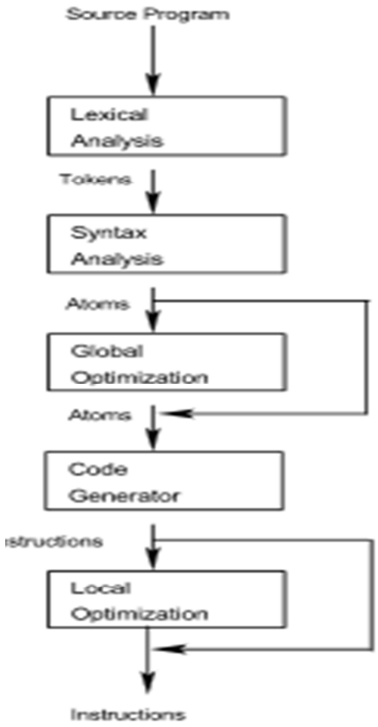

Figure 1 shows the phases of compiler

2.2. Design Methodology: Microprocessor design uses CAD tools provide a starting point in the design process. Existing processor designs provide an architectural reference point from which design modification can be made desirable architectural features. The most accurate method of architectural assessment involves circuit level timing simulation of full processor layout and cycle level simulation of full applications based on optimized compiled code.

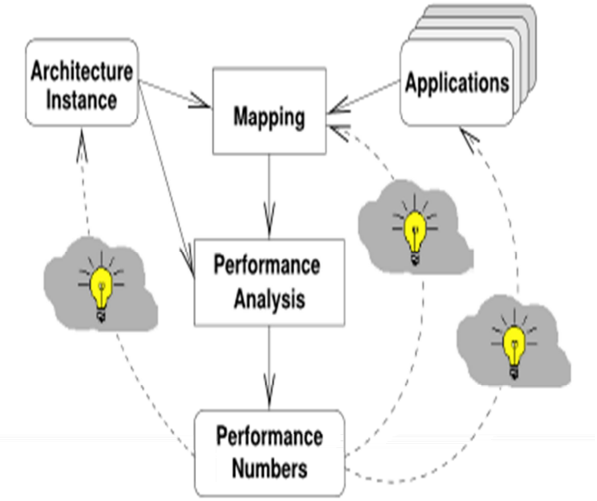

Figure 2 shows the Design Methods of Compiler

This architecture referred to as an architecture instance by evaluating its performance on a suite of applications using a mapping by a compiler to generate the assembly code performing analysis evaluate the resulting code. The implications of the results can be used for iterative improvements to the architecture instance mapping or applications. For our purpose we desire a media processor that I high-level language programmable without requiring special libraries or iterative improvements by the programmer for performance. The figure describes the design process at each level of the design space exploration, abstraction pyramid represent the early stages of the design space exploration.

Evaluation environment for this processor was provided by the IMPACT compiler is ideal because it supplies not only an aggressive ILP compiler but also the simulation environment and performance analysis tools necessary for a design space exploration using the design methodology. The basic compiler method enables the primary optimization of three paths

1. Classical optimizing and procedure inlining
2. Superblock includes all optimizations in classical and adds the superblock optimization and loop unrolling
3. Hyper block includes all optimizations in superscalar and adds the hyper block optimization.

III Parsing Application

3. Parsing Application:

Parsing can be defined as a process of analysing a text which contains a sequence of tokens to determine its grammatical structure in given grammar. One the source code is syntactically

valid the compiler has generated into abstract syntax tree or syntax directed translation of the source code.

Classical parsers conventionally accept a context free language defined by a context free, for each program parser does produce a phrase structure referred to as an abstract syntax tree. Parse including error stabilization and AST constructors can be generated from context-free grammars for parsers. Classical parsing techniques may be applied as long as program conforms to the syntax of a programming language however can be assumed in general as programs to analyze can be incomplete erroneous or conform to a dialect of the language.

Fuzzy parsing was designed to efficiently develop parsers by performing the analysis on selected parts of the source instead of the whole input. It is specified by a set of fuzzy context free sub grammars each with their own axioms does not require strict adherence to a language grammar. It scans for instances of the axioms and then parses according to the grammar makes parsing more robust since it ignores source fragments including missing parts errors and deviations that subsequent analyses abstract from anyway. Fuzzy parsers DelphiXPG utilise a context fuzzy parsing technology for building a cross identifiers, OPARI is source to source translation tool which automatically adds all necessary calls to the pomp runtime measurement which allows to collect runtime performance data of fortran, C, C++ applications.

Using Natural Language processing technique can be applied to formal language dependency structure is one way of representing the syntax of natural language. This technique automatically generates the language specific information extractor using machine learning and training of a generic parsing instead of explicitly specifying the information extractor using grammar and transformation rules.

Android Application Development web service request and response uses three types of parsing xml DOM Parser PULL Parser SAX Parser. Android provides a library that contains classes used to parse xml by constructing a document and matching each node to parse the info to parse with DOM parser as shown below.

```
Void parseByDOM(String response) throws ParserConfigurationException, SAXException,
IOException{
        Person person=new Person();
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance ();
        DocumentBuilder db = dbf.newDocumentBuilder ();
        Document doc = db.parse(new InputSource (new StringReader (response)));
        // normalize the document
        doc.getDocumentElement ().normalize();
        // get the root node
        NodeList nodeList = doc.getElementsByTagName ("person");
        Node node=nodeList. Item(0);
        // the node has three child nodes
        for (int i = 0; i < node.getChildNodes().getLength(); i++) {
        Node temp=node.getChildNodes().item (i);
        if(temp.getNodeName().equalsIgnoreCase("firstname")){
                person.firstName=temp.getTextContent ();
        }
        else if(temp.getNodeName().equalsIgnoreCase("lastname")){
```

© 2013, IJCSMC All Rights Reserved

```
                person.lastName=temp.getTextContent ();
        }
        else if(temp.getNodeName().equalsIgnoreCase("age")){
                person.age=Integer.parseInt (temp.getTextContent ());
        }

        }

        Log.e ("person", person.firstName+ " "+person.lastName+"
"+String.valueOf(person.age));
   }
```

It retrieves the correct information but the user needs to familiar with the xml structure so that we know the order of each xml. Android provides org.xml.sax package that has provide the event driven SAX parser to parse the previous response with SAX parser have to create a class extending Default Handler.

StartDocument() invoked when the xml document is open there we can initialize any member variables.

StartElement () invokes when the parser encounters a xml node here we can initialize specific instances of our object.

endElement () invoked when the parser reaches the closing of a xml tag here the element value would have been completely.

Characters () this method is called when the parser reads characters of a node value.

The method optJSONArray, optString, optInt instead of using getString, getInt because the opt methods return empty strings or zero integers if no elements are found. Parse platform provides a complete background solution for mobile application, storing data on parse is built around the ParseObject contains key value pairs of JSON compatible data.

IV Grammars in Compiler Design

4. Grammars in Compiler design: Lexical splits the input into tokens, syntax analysis is to recombine these tokens list of characters into something that reflects the structure of the text is syntax tree of the text. Designing a grammar to describe the syntax of a programming language usually made able to develop of a translator for the language so that programmers who use the language.

Grammars are used for developing practical algorithms part of compiler is concerned with the ability to find equivalent grammars, two grammars are said to be equivalent if they describe the same language that can generate exactly the same of sentences. In general we may be able to find several equivalent grammars for any language, problem in this regard is a tendency to introduce far too non-terminals or alternatively far too many, should not have escaped attention that the names chosen for non-terminals usually convey some semantic implication to the reader and the way in which production are written often serves to emphasize.

Equivalent Grammars: Two grammars are said to be equivalent if they describe the same language that can generate exactly the same set of sentences. In general several

© 2013, IJCSMC All Rights Reserved

equivalent grammars for any language, distinct problem is a tendency for terminal, choosing few nonterminal means that semantic implications are very awkward to discern at all that runs ambiguity and of hiding the semantic implications. It may be of some interest to give an approximate count of the numbers of non-terminals and productions that have been used in the definition of a few languages.

Language	Non-terminals	Productions
Pascal (Jensen + Wirth report)	110	180
Pascal (ISO standard)	160	300
Edison	45	90
C	75	220
C++	110	270
ADA	180	320
Modula-2 (Wirth)	74	135
Modula-2 (ISO standard)	225	306

Useless productions and Reduced Grammars: For a grammar to be practical value especially in the automatic construction of parsers and compilers should not contain superfluous rules that cannot be used in parsing a sentence. Detection of useless productions may seem a waste of time but it may also point to clerical error in writing the productions.

$G = \{ N , T , S , P \}$
$N = \{ W , X , Y , Z \}$
$T = \{ a \}$
$S = W$
$P =$

$W \rightarrow$	aW	(1)
$W \rightarrow$	Z	(2)
$W \rightarrow$	X	(3)
$Z \rightarrow$	aZ	(4)
$X \rightarrow$	a	(5)
$Y \rightarrow$	aa	(6)

A reduced grammar that does not contain superfluous rules, non-terminal that can never be reached from the start symbol and non-terminal that cannot produce terminal strings. Formally a context free grammar is said to be reduce if each non-terminal B we can write

$$S \Rightarrow^* \alpha B \beta$$

For some strings α and β, and where

$$B \Rightarrow^* \gamma$$

For some $\gamma \in T^*$.

Non-terminal that cannot be reached in any derivation from the start symbol are sometimes added so as to assist in describing the language.

ε-free grammars: Intuitively detecting the presence of "nothing" would be a little and for certain compiling techniques require that a grammar should contain no ε-productions which generate null string referred to as ε-free grammars. ε-productions are usually used in BNF as way of terminating recursion and are often easily removed

```
Integer      = digit RestOfInteger .
RestOfInteger = digit RestOfInteger | ε .
digit        = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
"8" | "9" .
```

can be replaced by the ε-free equivalent

```
Integer      = digit | Integer digit .
digit        = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
"8" | "9" .
```

V Applications of Compiler Design

5. Applications of Compiler Design:

Compiler is a program that reads the many application in networks, operating system, and embedded system with high positive rate.

5.1. Compiler Network Processor: Network processor instruction set allows avoiding costly shift operations special instructions for packet level addressing, compiler bit packet manipulation is made visible to the programmer by means of compiler known functions and maps call to compiler known functions not the regular function calls into instruction sequences. Using compiler known functions the developers have detail knowledge about the underlying processor readability of the code is improved significantly.

Compiler known functions with simple example consider a case where we would like to add the constant to a 7-bit wide packet stored in bits of some register denoted by the C variable a.

For example a = (a&0xFE03) | ((a+ (2<<2))&0x01FC).

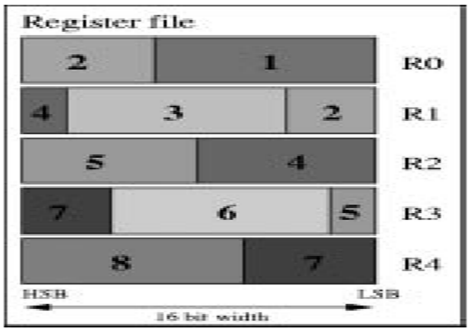

Figure 3 shows the values allotted in Register file

This expression be simplified into standard compiler optimization techniques that are constant folding which translate into a relatively large instruction sequence on a standard processor. Network processor can implement the entire assignment within a single instruction, for this purpose can use packet access

PA (int op, int var1, int off1, int var2, int off2, int width).

Parameter op denotes the operator var1 and var2 are the operands and off1 and off2 are the operand packet offset, denotes the packet bit width and directly reflects the packet level network processor instructions. using packet access the above example can be expressed in C language as shown.

```
int a, b;

b = 2;
PA (PA_ADD, a, 3, b, 0, 7).
```

The variables a and b are mapped to registers in the assembly code by the register allocator. C compiler for the network processor is fully operational and generated code has been measured by means of a cycle true network processor instruction simulator for a set of test programs. Quality of compiler generated code as compared to handwritten assembly code largely depends on the clever use of compiler known functions, when using compiler knowledge function without specific knowledge on application the performance overhead of compiled code may be acceptable for the intended application domain.

5.2. Collaboration of Compiler & Operating System: System designer must define how to initiate a speed change and how to select a speed level to automatically deciding on the proper locations to insert PMPs by the compiler in an application code, how frequently the speed should be changed. The solution over here that determines how far apart any two PMPs should be placed with sequential code and an estimate of instruction latency the code is inserted. In real time the problem occurs due to the presence of branches, loops and procedure calls that eliminate the determinism of the executed path compared to sequential code. To control the overhead of speed scheduling during the execution of an application we use a compiler directed technique, during compile time the compiler inserts instrumentation code that computes information about the worst case remaining cycles of the application. Benefits to use collaborative scheme that includes both PMHs and PMPs that timing information can be inexpensively collected without actually doing a speed change and incurring its high overhead.

To implement the collaboration the operating system requires a definition for a new ISR for adjusting the processor speed and two system calls to communicate with the application. ISR reads the current WCR and computes the speed according to the selected dynamic speed setting scheme then ISR issues a speed change request if needed, the system calls transfer the timing information from the application to the operating system. Two system calls are needed that are called only once at the start of an application one is a system call that gives the address of the buffer that holds timing information collected by hints.

© 2013, IJCSMC All Rights Reserved

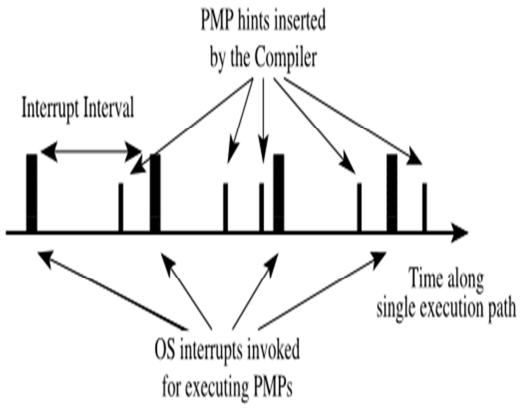

Figure 4 Specific path for Invocations

PMP interrupt service routine is periodically invoked to adjust processor speed based on WCR. Executes the power management hints at some point before a PMP to update the WCR based on the path of execution. Compiler role is to support and inserts PMHs in the application code. Dynamically the compiler instruments the PMHs in a way to collect the application dynamic behaviour information. The compiler allocates and manages memory space for a table that stores p_wcr_j updated by the PMHs the entry in the table deleted at the termination procedure instance that corresponds to this entry.

VI Conclusion

In this analysis describes the implementation of compiler for real-time application in order addressing the accessible at the high level language which uses the compiler knowledge function and register allocation technique. Outlines the techniques of compiler parsing application and types the grammars which can be used in real-time application in design process, aim are to model the compiler not only used for translation. Future work extends with the design problem solution how the compiler are being developed in order to further improve of quality gap between compiled code and handwritten assembly code.

Reference:

[1] J.H Edmondson P. Rubinfeld R. Preston and Rajagopalan, Superscalar instruction execution in the 21164 Alpha microprocessor "IEEE micro pp. 33-43 April 1995.
[2] Jens Nilsson, Proceedings of the 11th International Conference on Parsing Technologies (IWPT), Paris, October 2009 c Association for Computational Linguistics
[3] elphixpg.com/docs/contextparsing.html
[4] Gepard DSP core, Austria Mikro Systeme International 2000 Online on http://asic.amsint.com/databooks/digital/gepard.html.
[5] A.V. Aho M.Ganapathi and S.W.K. Tjiang "code generation using tree matching and dynamic programming". ACM Trans Program Lang. Syste. Vol 11, no. 4, pp, 4911-516 2009.
[6] About Ghazaleh N, Childers B, Mosse D, Melhem R, Craven M. Collaborative Compiler OS power management for applications. TR-02-103, 2002 http://www.cs.pitt.edu compiler design.
[7] Azevedo A Issenin I, Cornea R, Gupta R, Dutt N, Veidenbaum A, Nicolau A dynamic voltage scheduling using program Design automation and test in Europe 2005.

Authors Profile:

Ch. Raju, M.Tech Computer Science and Engineering from JNTUH MS.Is from Osmania University BCA from Osmania University. He is having 8+ years of experience in teaching currently working as Assistant Prof for Osmania University Hyderabad has guided many UG & PG students. His research areas include Distributed Systems, Network Security, and Design Patterns.

Thirupathi Marupaka pursuing M.Tech Computer Science Engineering from IETE B.E from Osmania University He is having 5 years of experience in teaching currently working as Asst Prof for Osmania University Hyderabad has guided many UG & PG students. His research areas include Distributed Systems, Network Security, and Design Patterns.

Arvind Tudigani M.Tech (IT) from AAIDU, Allahabad, B. Tech(IT), JNTU,Hyderabad He is having 8+ years of experience in teaching currently working as Asst Prof for Osmania University Hyderabad has guided many UG & PG students. His research areas include Distributed Systems, Network Security, and Design Patterns.