The study of execution mechanism of Function Language for A Multithread Processor Architecture

Jin Yu (210027)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 13, 2004

Keywords: parallel,SML,thread,multithread,token,top-down parsing,symbol table.

Abstract

In this paper, it aims at efficient execution of a functional language with the combination of multi-thread processor architecture and the functional language. Although paradigm of functional language, logical language and object-oriented language are efficiently simulated by the sequential architecture, the parallel performance possibility which is in those paradigms is ignored. Moreover, although the multithread type processor architectures that were suitable for execution of a function type program were proposed, the compiler applied to the multi-thread processor was not proposed. In this research, the compiler which performs parallelism detection, generation of multi threads and the synchronization of threads is proposed. First of all, let's pay attention to Standard ML.

1 INTRODUCTION

In this paper, it aims at efficient execution of a functional language with the combination of multi-thread processor architecture and the functional language. Formerly,architectural support for programming language has traditionally been popular as an in indirect way of application specific support. But "High-Level Language Computer" has been forgotten. Today,the Microprocessor focuses our attention on RISC. Pepople tried that problems of software was solved by hardware on "High-Level Language Computer". Nevertheless,for users, value of computer systems is not the archievement

of "High-Level Language Computer" but the ratio of cost and efficiency. There are two major subjects in software development. One approach is to catch the method of software development in engineering function. The other one improves from the base of the programming paradigm and aims at conversion from the imperative paradigm which bases on von Neumann type architecture to other paradigms. So,parallel processing or the parallel computer architecture make an interest in the functional and logical programming. However, this is not in agreement with overall cost effectiveness.From the viewpoint of cost effectiveness, if a machine is created with simpler structure and less resources is priority even if its performance is a little inferior. Therefore, a "High-Level Language Computer" is no worth to exist.

Generally programming language processing is divided into three phases. First is language translation phase. Second is operation sequence control phase. Third is execution phase. In translation phase, we can see that it is cost effectiveness to assign the software compiler translation jobs. On the other hand, in order to speed up execution phase, using hardware leads execution pahse themselves to highest effectiveness. Speed-up by hardware is effective in parallel processing. Remaining opinion that we have already discussed in "High-Level Language Computer" is operation control phase. So far, this is discussed in the problems of instruction issue or instruction scheduling for sequntial architecture having instruction pipeline. Although the paradigms including functional, logical, and object oriented langurages can be simulated effectively in cost on sequential architecture, those paradigms involving parallel execution potentialities are ignored. Moreover, although there are some researches on parallel execution of each paradigms parallel language, few of them concerned about parallel execution of language, i.e. the compiler which can detect parallelism.

2 Functional Language and the manner of process

In this paper, there are some discussions about Standard ML and research its interior parallel execution possibility. There are two main reasons that interfere with the efficient execution of a functional language. First, it is

execution of a command of the stack operation by the function call which occurs frequently in a program. This operation produces a hazard on the usual pipeline. A multi-thread processor can avoid the hazard. Next, a program's interior parallel execution possibility is ignored. In a strict meaning, The functional language does not have side effect. Program is defined as *f* like a certain function. It is defined as evaluation of *f* **application** *f(x)* when input *x*. In the definition of function *f*, *f* itself and other functions can be used. For example, the evaluation of *f(x)=g(h(x),k(x))*can start from the outside of it-*g*, or can start from the inside *h(x)* or *k(x)*, or can start from *h(x)* and *k(x)* simultaneously. In strict function language, the same evaluation result is guaranteed in different order evaluation. This character shows the possibility of parallel processing. Thus, it is believed that a compiler which can perform parallelism detection and the synchronization of threads is necessary.

3 Detection of Parallelism and Synchronous Processing

In this paper, a data dependency relation and synchronous processing which are necessary for the implementation of the complier are proposed as follows.

3.1 The detection method of a data dependency

In SML, evaluation of a expression is under a certain environment. Environment is a set of the variable bounded by the value. Therefore, a expression is evaluated by the environment. SML is a language with a static scope rule, and the value of a variable will not change after the definition of the value. This rule is also applied to a function definition. There are free variables without bounded variables in the body of a function. A free variable are a values bound to the variable at the time of a function definition, and is not related to the definition of the same name variable when a function is performing.

Therefore, in this research,a data dependency is inspected while observing environment. Making environment in a symbol table at the each stage

of the front-end, and investigating the symbol table in the next processing stage at the compile time check the data dependency. It is necessary to add new attribute of environment into the symbol management table.

3.2 Synchronous processing along Expressions

The execution times of the expressions are various, and along the expressions it is necessary to synchronize after parallel operation. If the value of a expression returns a value before the complete evaluation of its partial expressions , it will cause a incorrect result. In order to avoid incorrect result, in this research,synchronous processing is performed as follows.

When evaluating the whole expression, evaluation of the expression element (partial expression) returns the mark which means the finish status of the formula and its the value together. If an element is all completed, returns 1, if not 0 will be returned. For example,

```
val Exp = exp1+exp2+exp3;
```

if exp_1,exp_2 and exp_3 are parallel executions possible, synchronous processing is judged by checking Marks A, B, and C. Figure1 is the diagram of synchronous processing of the expression which bases on the proposed method.

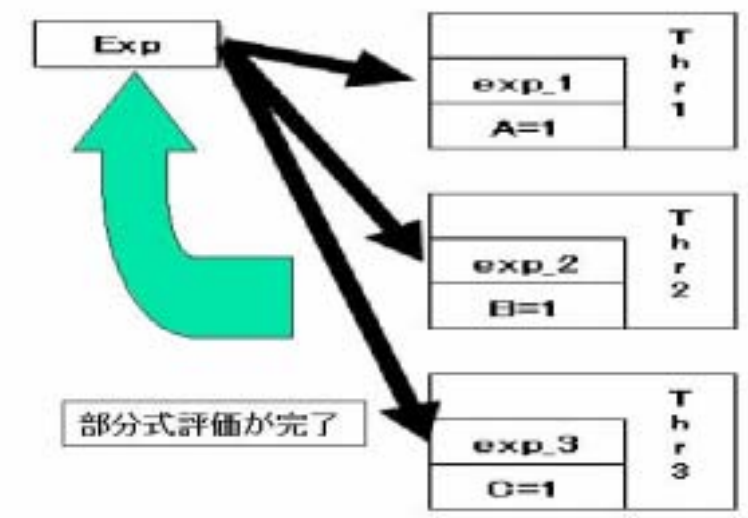

Figure 1: synchronous processing

4 Implementation of a Compiler

4.1 Lexical Analysis

Lexical analysis is the first phase of a compiler. The main task is to outputs the sequence of tokens by the demand of syntax analysis, while reading characters in order. When lexical analysis receives a demand of "get the following token" from syntax analysis, an input character will be read and the following token will be decided. For the implementation of lexical analysis, make the definitions as follows. The data type is integer and it is forbidden to use a keyword as a identifier. Token is specified in Table 1.

token	::=	keywords	identifier	numeric	special_sign	
keywords	::=	andalso	orelse	if		
	|	then	else	fun		
	|	val	let	in		
	|	end	ture	false		
identifier	::=	letter	identifier	alphanumeric		
alphanumeric	::=	letter	digit			

Table 1: specification of tokens

Mainly, lexical analysis consists of three portions; blank skip, token judgment, reading of token. Always, lexical analysis pre-read next character and judges it should be the end of the present word or the head of the following word.

4.2 Syntax Analysis

Syntax analysis match the combination of the token delivered by lexical analysis with grammar, and shows the processing which changes to a tree structure as a result. Grammar is described in Table2. Generally there are two kinds of syntax analysis, the top-down analysis analyzed from the route of a tree structure, and the bottom-up analysis analyzed from a leaf. In this research, top-down parsing analysis is used.

This grammar is "If one character's is predicted, it is possible to analyze syntactically without ambiguous". Therefore, a "prediction token" is

needed in order to perform syntax analysis. A pair of a prediction token and a syntax tree is delivered in syntax analysis processing.

The information about a token is registered into the symbol table during the syntax analysis. That means, the variable name and the function name will be registered and researched. Since operation of a syntax error is not the main goal of this research, it is simplified. After finding a syntax error, analysis will stop, and will not execute an error recovery.

program	::=	phrase_list					
phrase_list	::=	phrase					
	|	pharase	phrase_list				
phrase	::=	declaration					
	|	exp					
exp	::=	exp_4					
	|	LET	declaration	IN	exp	THEN	
	|	IF	exp	THEN	exp	ELSE	exp
exp_4	::=	exp_3					
	|	exp_3	,	exp_4			
exp_3	::=	exp_2					
	|	exp_2	=	exp_3			
	|	exp_2	<>	exp_3			
exp_2	::=	exp_1					
	|	exp_1	+	exp_2			
	|	exp_1	-	exp_2			
exp_1	::=	exp_0					
	|	exp_0	*	exp_1			
	|	exp_0	/	exp_1			
exp_0	::=	exp_{simple}					
	|	exp_{simple}	exp_0				
exp_{simple}	::=	integer					
	|	identifier					
	|	boolean					
	|	(	exp	)			
	|	()					
declaration	::=	VAL	exp				
	|	FUN	exp				

Table 2: specification of grammar

4.3 Management of a Symbol table and Code Generation

In a source program, identifier means a variable or a function. Each identifier has various information, such as the data type and the scope. A compiler extracts these information from a program and holds it to the symbol table. In source program, a variable name or a function name expresses a certain object. These are declared with the information of data type in the declaration part of the program. Moreover, these are referred to in the execution part of a program.

In common for the symbol table is treatedas a whole, no matter what kind the identifier is. However, in this research, the variable name and the function name are handled in different table. Only a variable name is put into the symbol table and a function name is put into another table. Declaration of a symbol table is implemented as follows.

```
(*****variable-name*****)
type binding = {count:int, level:int,
value:{tag:int,boundvalue:int} list}
type bucket = (string*binding) list
type table = bucket Array.array
val t : table = Array.array(SIZE,nil)
```

In this, *count* is used for the management of the identifier and the value of *count* will increase 1 if the same name declaration appears.

When *level* meets a local declaration, it increases a value, and refer to the no.*level* value list. Tag records the information of whether the identifier is defined or not.

The function table is implemented as a simple array. The element of the array is implemented by record which has 4 fields;*name, parameter, funcbody* and *idtable*. *funcbody* means the defined function and *idtable* is the symbol table of the identifier which appears in the definition place of a function. If there is a global variable in a function definition, a top-level table is referred to. If there is a local one, it will be registered into the idtable of the function.

A instruction code is generated with using the syntax tree and the symbol table which were obtained from syntax analysis, while carrying out a hand compiler.

5 Conclusion

In this paper, we discussed the union between multithread process and functional language, furthermore, we think that it is very possible to implement functional language high-efficiently. Here, we only deal with it in theory, in our future research, we will further to perfect this proposal in theory then to implement it.