B-24

High Parallelism Java Compiler With Queue Architecture

Li Qiang Wang,Yoshinage Tsutomu, Sowa Masahiro *†

1 Introduction

Increasing computational throughput can be achieved via exploitation of potential parallelism. One way of this realizing is to use a queue machine. A queue machine is a processing element that uses a queue as the underlying mechanism for the manipulation of operands. A compiler for the Java programming language has been developed. A new type of syntax tree optimum for queue execution model compiler-Queue Syntax Tree-is also introduced , not only for queue java compiler. A benchmark system to simulate and measure the performance of java in queue execution model shows that the average of parallelism of queue java architecture is 2 to 3 times greater than that of to stack java(normal java). It means that, in a idealize parallel computing envirment, we can increse java computational throughput 2 to 3 times.

2 Queue Execution Model(QEM))

The QEM uses a first−in, first−out (FIFO) queue data structure as the underlying control mechanism for the manipulation of operands and results. A QEM machineis analogous to a Stack Execution Model(SEM) machine in that it has operations in its instruction set which implicitly reference an operand queue, just as a SEM machine has operations which implicitly reference an operand stack. In a SEM machine, implicitly referenced operands are retrieved (popped) from the top of an operand stack and results are returned (pushed) backonto the top of the operand stack. In a QEM machine, operands are retrieved from the front of the queue of operands and results are returned to the rear of the queue of operands.

It is well known that the SEM machine instruction sequence corresponding to a given expression's parse tree can be obtained from the parse tree by doing a post order traversal of it as shown in Figure.1.(a) . It will be shown that the SEM machine instruction sequence can be obtained by traversing the parse tree in a new way . This traversal method is called a level order traversal. The following will serve to illustrate how a level order traversal proceeds .

A level order traversal of the parse tree in Figure.1.(b) is performed by visiting the nodes of the tree in the arrow order shown in Figure.1.(b). Informally, a level order traversal is done by visiting the nodes of the parse tree from the deepest to the shallowest levels and from left to right within each-level. It was shown that a level order traversal produces a QEM instruction sequence.

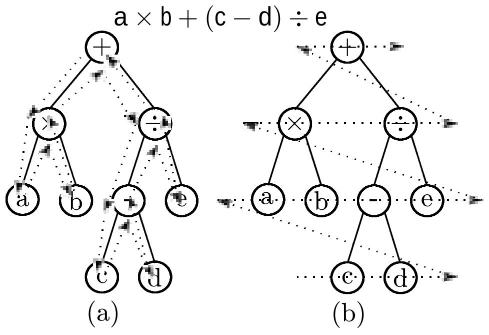

Figure 1: Traversing of post-order and level-order

The QEM mechanism implemented in a pipelined arithmetic/logic unit (ALU) is more efficient than the SEM one−since the results of one operation must be returned to the top of the s tack before they can become the operands of the next operation.

3 The QEM Java

The QEM Java is an extension of the QEM and Java Platform. It is composed of The Language and Virtual Machine(VM) Specification. The syntax and semantics of queue execution mode java take no change with conventional java system released from Sun Microsystem, the difference only occurs in the VM. As this well known, the conventional Java VM uses SEM, the QEM mechanism in a pipelined arithmetic/logic unit (ALU) is more efficient than the SEM mechanism. It can exploit more parallelism in fetch and execution than SEM mechanism, so queue java VM which uses QEM, is expected to have a better performance than conventional Java.

4 Queue Syntax Tree For QEM

Queue Syntax Tree is a new type of syntax tree optimum for QEM Compiler.

For SEM, the conventional syntax trees provide enough information for it's compiler. With the lines between nodes in the parse tree shown in Figure.1.(a) , The SEM compiler can easily jump to

*The Graduate School of Information Systems The University of Electro-Communications
†Advanced Distributed/Parallel Computer Systems Laboratory

the children and back to parent to produce instructions in SEM order. But the problem occurs when we still use conventional syntax trees to deal with the QEM order. The compiler, in order to find the deepest and shallowest nodes, must traverse all the tree at first, remember the position of the nodes, then load the tree again to emit instructions. Because there is not connection information existing between same-level nodes, finding same-level nodes in instructions issue stage is very diffcult and its algorithm becomes very complex and huge, also time-consume.

Here we introduce Queue Syntax Tree. Its characteristics are:

- the node means one or more instructions that should been emitted in here
- the connection is still appeared between two nodes
- the connection is not only between the parent and child nodes and also brother and brother nodes

Thus, the queue syntax of $a \times c + (c - d) \div e$ becomes as shown in Figure.2.(b). We can see that

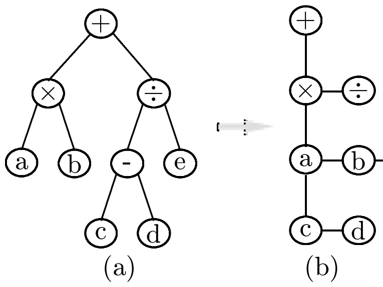

Figure 2: Queue syntax tree for $a \times c + (c - d) \div e$

the sume of nodes in the expression are kept, the connections of the brother-to-brother(same-level) is created , the necessary connections among some parent-to-child nodes are preserved, but some unuseful connection are cut off. Under our experiment, with the number of nodes in a expression increasing, the consumed time of instruction emitting a queue syntax tree can be reduced more comparing with conventional syntax tree. The result of comare is shown in Figure.3.

5 The Benchmark

We also developed a benchmark system to simulate and measure the performance. The benchmark system has functions to calculate the maxinum-parallelism, the average-parallelism and the sume of instructions. It also can give of the transition curve of queue-length and parallelism, it will help us in future adjust source program to fit the optimum status. With the benchmark system, we are satisfied with the result: the average-parallelism of Queue Java is 2 to 3 times greater than that of to conventional Java.[1]

	FFT		Quick Sort	
	javac	qjavac	javac	qjavac
compile time	1.73sec	2.47sec	1.66sec	1.98sec
max-memory consume	2773KB	4399KB	2877KB	4481KB
max-queue length		8		4
ave-queue length		2.46		1.24
max-parallelism	2	6	2	5
ave-parallelism	1.01	2.33	1.07	2.58

Table 1: Benchmark Result of qjavac comparing javac in FFT and QuickSort Algorithm

6 Conclusions and Future Work

The work in this thesis has shown that a queue machine model is as powerful as a conventional stack machine in terms of evaluating arbitrary expressions. Furthermore, it was shown that instruction sequences for the QEM has more parallelism than SEM instruction sequences. An important observation is that a QEM machine is better than a SEM machine at utilizing a pipelined arithmetic/logic unit.

There still remain many unresolved compiler implementation issues. The current compiler dose not completely decompose any loop bodies and conditionals for execution in separate contexts. In many cases, this decomposition is excessive and actually increases computational throughput. Work needs to be done to develop a more intelligent partitioning of the source program.

A queue machine processing element architecture has been proposed and specified in some detail. Work still remains to specify the control logic for the processing element and possibly to implement a prototype in silicon.

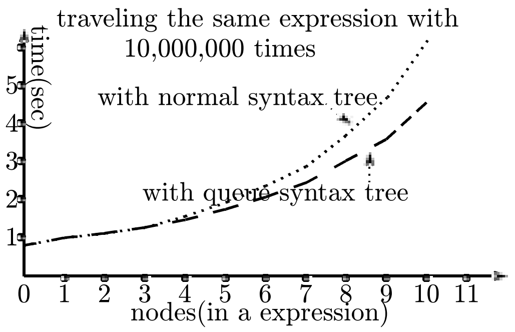

Figure 3: Result of emmiting time with conventional syntax tree and queue sytax tree

[1]hardware and software envirment:PIII 450MHz, 128MB, Windows2000