Optimality of Tomasulo's Algorithm

Tian Sang and Lin Liao
Department of Computer Science and Engineering
University of Washington

Abstract

Little work has been done about the optimality of Tomasulo's algorithm, the most widely used dynamic scheduling strategy. In this paper, we try to answer the question whether Tomasulo's algorithm is optimal. We first present a reference model in order to define the optimality. We have defined three kinds of optimality by adding different constraints to the reference model. Then we discuss the optimality of Tomasulo's algorithm with unlimited resources and limited resources respectively.

1 Introduction

To increase the instruction-level parallelism, dynamic scheduling algorithms are widely used to support out-of-order execution. Tomasulo's algorithm is the most classical one among them. It was first introduced in IBM 360 Model 91 in 1967 [13]. The basic idea is using register renaming to eliminate name dependencies and using Common Data Bus (CDB) to broadcast results globally. Today, its variants are still popular in modern high-performance microprocessors, such as Intel's Pentiums, AMD's K7, IBM's PowerPCs and so on [12].

The correctness of Tomasulo's algorithm has been verified by various methods [1][9][10][12]. But little work has been done to evaluate the optimality of Tomasulo's algorithm, although most people agree on Tomasulo's algorithm's excellence for dynamic scheduling. There are two possible reasons. First, the concept of optimality is too vague to be evaluated. Second, to determine the optimality of a scheduling algorithm is often very hard, even for some very simple cases.

In this paper, we try to tackle the optimality of Tomasulo's algorithm. Overall, the contributions of this paper are:

- We present a reference model in order to give a precise definition of optimality. By giving different constraints to such a reference model, we get the definitions of three kinds of optimality.
- We discuss the optimality of Tomasulo's algorithm and the complexity to achieve the optimality under different assumptions and different definitions of optimality. Especially we show that in most cases Tomasulo's algorithm is not optimal or it is NP-hard for it to achieve optimal value.

The remainder of this paper continues with a brief introduction of our model for Tomasulo's processor[1] in section 2. Then in section 3, we discuss the optimality with the assumption of infinite resources. We then suppose only limited resources are available and discuss the optimality in section 4. We extend our model to include load/store buffers in section 5, and it is followed by the related work in section 6 and the conclusions in section 7.

2 Models

In this section we introduce the Tomasulo's processor model and present our basic assumptions. Then we introduce the reference model, which will be used to define optimality.

2.1 Tomasulo's Processor Model

Figure 1 shows the model of Tomasulo's processor. Instructions are issued from the instruction buffer to reservation stations. A register alias table (RAT) maintains the identity of the latest pending instruction writing a particular register. A scheduler controls the movements of instructions through the pipeline. Such a pipeline has three stages: decode & issue, execute and write back. The scheduler uses three signals as issue, dispatch and write_back to control the movement. CDB is used to transfer the results from function units to reservation stations, RAT and register file. Throughout the paper, we suppose:

a) No branches, we only consider the scheduling in a basic block;
b) Infinite instruction buffer and there are always enough instructions in the buffer;
c) Multiple and out-of-order issue, but in one cycle the window size and the issue number have fixed upper bounds;

[1] Tomasulo's processor means the processor that schedules instructions using Tomasulo's algorithm.

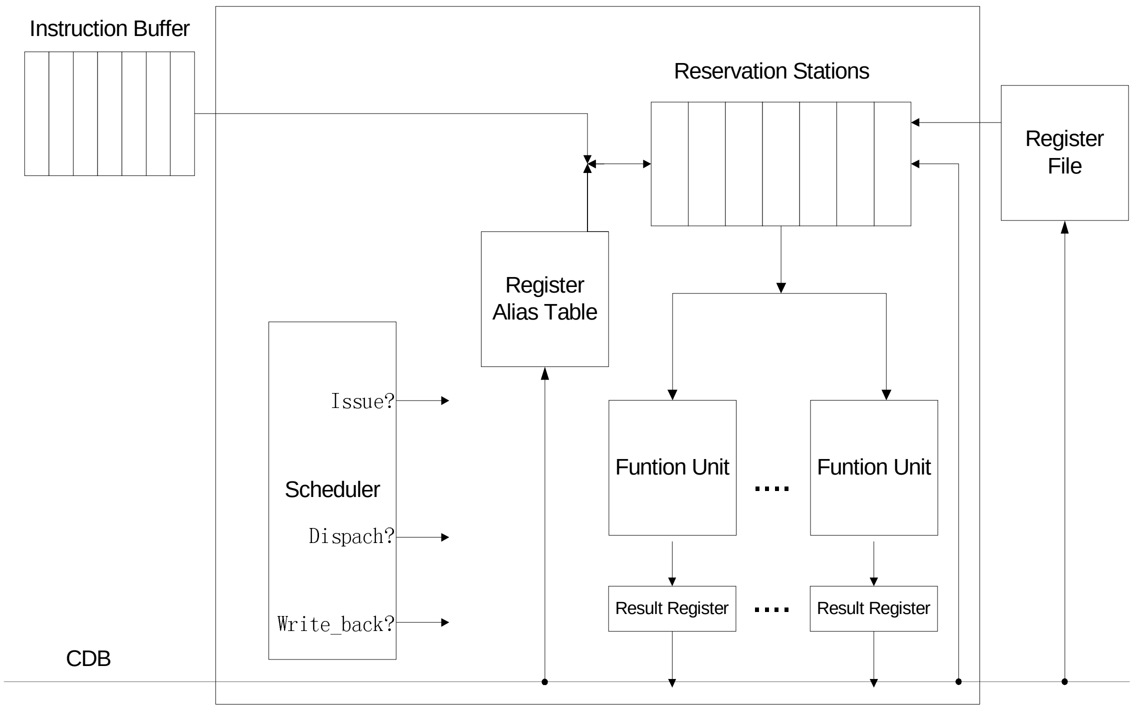

Figure1: Tomasulo's Processor Model

d) Instructions in the buffer can be decoded and issued into reservation stations in one cycle. Although this is often not realistic, we can use pipelines to overlap decode and issue so that it will not impact our discussion;

e) A central reservation station buffer with infinite stations and all the instructions can be dispatched into function units in one cycle. This means all the function units share the same buffer and no stalls are caused by lack of stations. The alternative is each kind of function units has its own stations. Since we suppose reservation station's number is infinite, this alternative will not affect our discussion either. The reason we assume infinite reservation stations is that our reference model does not have to use reservation stations and it is difficult to compare these two models.

f) We will suppose data transmission on CDB only needs one cycle except that we explicitly indicate.

g) We won't discuss load/store instructions until section 5.

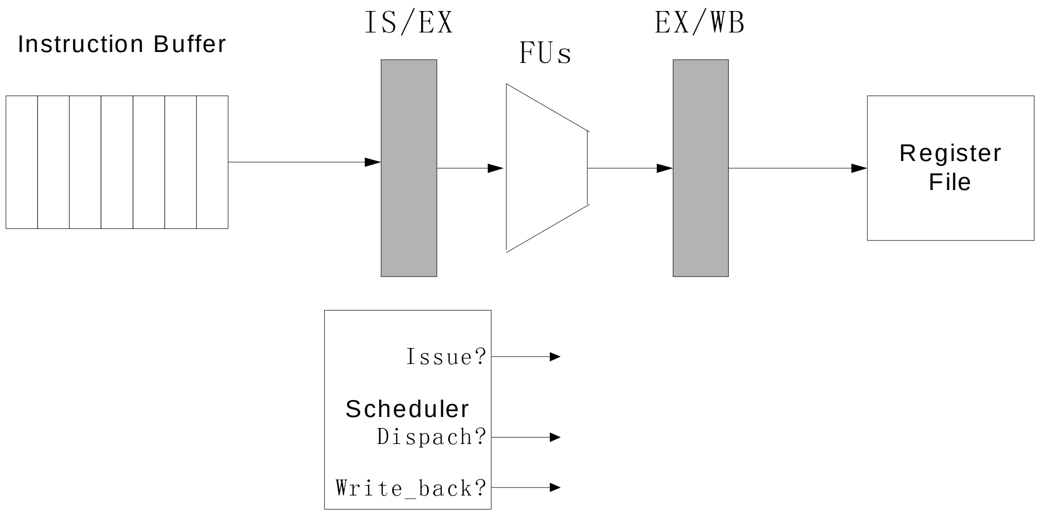

Figure 2: Reference Model

2.2 Reference Model

The reference model is shown in figure 2. The different kinds of optimality are based on different constraints on this model. To be fair, we suppose the reference processor is also pipelined into 3 stages and each instruction stays at least one cycle in the intermediate registers. We also suppose it has the same types and numbers of function units as Tomasulo's processor. And it has the same issue width (the upper bound) with Tomasulo's processor. Since we want to use such a model to define optimality, we suppose that the reference model can use any structures to store intermediate results and can always make optimal schedules under the constraints.

3 Infinite Resources

In this section, we suppose there are infinite resources, e.g. infinite function units and CDBs for the Tomasulo's processor and infinite function units in the reference processor (since the reference model does not have to use CDBs).

3.1 Global Optimality

We first give the definition of ideal reference processor and global optimal value.

Ideal reference processor: if we don't add any additional constraints to our reference model, we get ideal reference processor. Especially, the window size of ideal reference processor is unlimited.

Global optimal value: given a set of instructions, global optimal value is the minimum cycles needed for the ideal reference processor to complete all the instructions and guarantee the data dependencies[2].

We use an example to illustrate our definitions. Suppose there are four instructions in the instruction buffer as follows[3]:

```
ADD R1, R2, R3
ADD R4, R1, R5
ADD R6, R1, R7
MUL R8, R9, R10
```

We can use a DAG (Directed Acyclic Graph) to express the dependencies among instructions, as shown in figure 3. In the figure, the arrows express the dependencies among instructions and the number in each node is the execution time plus one cycle for writing back (we suppose addition needs 2 cycles and multiplication needs 10 cycles through the paper).

[2] In our discussion of the optimal value, we ignore the first cycle to issue the first instruction. Such a simplification will not affect our discussion at all.

[3] The instruction format is: Opcode Dest, Src1, Src2

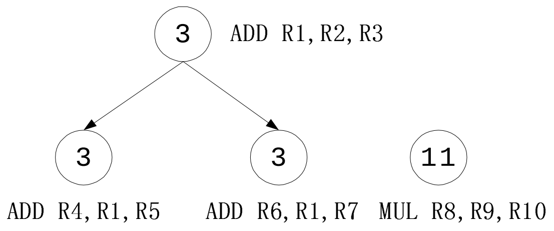

Figure 3: DAG for global optimal value

In this example, the global optimal value is 11 even if the issue width is only one, because the ideal reference processor will issue the fourth instruction first.

Proposition1: Tomasulo's processor cannot guarantee global optimality even though it has infinite function units and CDBs.

The reason is that although we suppose Tomasulo's processor can issue multiple instructions out of order, the window size has an upper bound k1 and the issue number has an upper bound k2. Here k1 and k2 are both constants and k2 ≤ k1. We prove Tomasulo's processor is not global optimal by giving a counterexample.

As shown in the figure 4, there are 2*k1+1 addition instructions. The first k1 is on the left and the last k1+1 is on the right. The arrows express the data dependencies. Since the window size of our Tomasulo's processor is only k1, in the first cycle it cannot issue instructions on the right side. So the cycles needed will be greater than 3*(k1+1). But the global optimal value is exactly 3*(k1+1), because the ideal reference processor will issue the instructions on the critical path first. Thus Tomasulo's processor cannot achieve global optimality in this case.

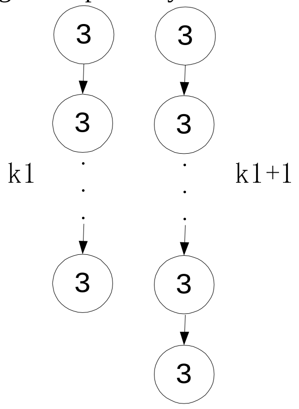

Figure 4: DAG for counterexample

3.2 Constrained Optimality

The reason that Tomasulo's processor cannot achieve global optimality is that the ideal reference model has an infinite window size. In fact, if we suppose the window size of Tomasulo's processor is also unlimited, it can guarantee global optimality (we don't prove here). To be fair and to understand the capability of Tomasulo's processor better, we should add some constraints to our reference model.

Constrained reference processor: If we restrict our reference processor to issue the same instructions with Tomasulo's processor, we get the constrained reference processor model.

Constrained optimal value: given a set of instructions, constrained optimal value is the minimum cycles needed for the constrained reference processor to complete all the instructions and guarantee the data dependencies.

Proposition2: If the Tomasulo's processor has infinite functional units and infinite CDBs, it can achieve the constrained optimality.

Assume we have an instruction set S which contains n instructions. There is a DAG representing the dependencies in S. We'll construct a sequence $S'_0 \subseteq S'_1 \subseteq \ldots$, and we'll show such a sequence will converge to S when all the instructions are completed. We prove that for each instruction in S_k', it can finish on both processors at the same time. Then we conclude all the instructions in S can be completed on both processors at the same time.

According to our definition of constrained reference processor, the constrained reference processor and Tomasulo's processor will always issue the same instructions every cycle, so every instruction is issued at the same time on both processors. We construct the sequence as follows:

$S'_0 = \Phi$

Repeat

$\Delta S_k = \{x \mid x \in S \wedge x \notin S_k' \wedge (\text{predecssor}(x) \subseteq S_k')\}$

$S'_{k+1} = S_k' \cup \Delta S_k$

until $\Delta S_k = \Phi$

Here, predecessor(x) means the set of all predecessors of x in the DAG.

First, any S_k' can be completed at the same time on both processors. We prove this by mathematical induction:

1) S'_1 = {instructions in S that don't have predecessors}, all these instructions are issued at the same time, thus be dispatched and completed at the same time on both processors.
2) Suppose all instructions of S_k' finish at the same time on both processors, then for S'_{k+1}, since all the predecessors of instructions in ΔS_k are in S_k' and they will finish at the same time, all instructions in ΔS_k will be dispatched and finish at the same time on both processors. Therefore, all the instructions of S_{k+1}' will finish at the same time on both processors. Done.

Second, since in the DAG, there are no loops, so ΔS_k cannot be empty until all the instructions are completed. And because S is limited, we have S' = S by at most |S| steps. Thus both processors can finish S at the same time.

4 Limited Resources

In this section, we suppose the resources in processors, e.g. function units or CDBs, may be limited. Then there are potential structural hazards. We first suppose function units are limited but the CDBs are still infinite. Then we discuss the optimality when CDBs are also limited.

4.1 Limited Function Units and Infinite CDBs

In this section, we suppose there are only limited functions units for Tomasulo's processor and reference processor. So there are potential structural conflicts when dispatching. Since the number of CDBs is unlimited, no CDB conflicts will happen (in fact it's enough for each function unit has its own CDB).

We first show that constrained optimality is not always true if we only have limited function units.

Proposition3: The Tomasulo's processor cannot always achieve the constrained optimality if there are only limited function units.

We prove this by showing a counterexample. Suppose there are three instructions already issued to the reservation stations, as shown in figure 5. At some cycle, the first instruction completes and the next two instructions are ready to dispatch. But there are only one adder and one multiplier. So a structural hazard happens and the scheduler has to decide which instruction should be dispatched first.

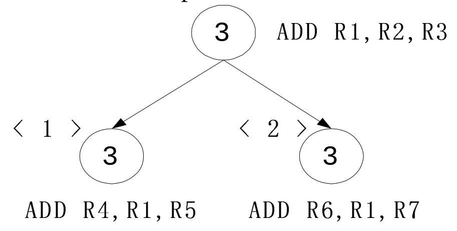

Figure 5: Three Instructions with Function Unit Conflict

Since Tomasulo's scheduler has no idea about the instructions in the instruction buffer, we will show that no matter which instruction the scheduler dispatches first, it may not be optimal.

Without loss of generality, we can suppose instruction <1> is dispatched first. Then we can construct such a case: the issue buffer has only one instruction MUL R8, R6, R9. Thus the DAG is like figure 6.

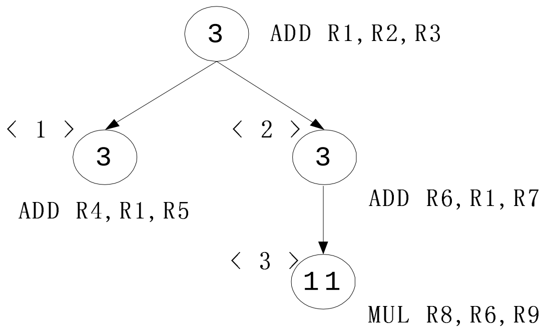

Figure 6: DAG for the Four Instructions

The instruction <3> cannot start until both <1> and <2> complete. That means the cycles for Tomasulo's processor to finish the instructions is 20. But the constrained reference processor has the magic to know what the not-issued instructions are. Thus in this case it will dispatch instruction <2> first and complete all the instructions in 17 cycles.

Again, this is a little unfair, because the practical processor cannot schedule based on the instruction not issued yet. We need to restrict the reference model. Then we get the local reference processor.

Local reference processor: If we don't allow the constrained reference processor to schedule based on the instructions not issued, we get the local reference processor.

Local optimal value: minimum cycles needed for the local reference processor to complete all the instructions already issued and guarantee data dependencies.

We will show that for a Tomasulo's processor, the complexity to achieve local optimality is NP-hard. We present four problems and each is closer to the reality than the previous one, then we show all the problems are NP-hard.

The general problem is "Given a set of issued instructions and their dependencies, find a scheduling strategy that achieves local optimal value" and the following problems have different assumptions.

(P1): Given k identical, empty function units and each instruction only needs 1 cycle to execute, where k is an integer variable greater than zero.

(P2): Same with (P1), except each instruction needs t cycles and all the function units are pipelined into t stages, where t is an integer variable greater than zero.

(P3): Same with (P2), except there are 2 kinds of function units, adder and multiplier. Addition needs t1 cycles and multiplication needs t2 cycles, where t1 and t2 are integer variable greater than zero.

(P4): Same with (P3), except some instructions may be already in function units when scheduling.

Then we prove all the four problems are NP-hard.

Theorem1: (P1) is NP-hard

Ullman showed in [14] that an equivalent scheduling problem is NP-hard using a polynomial time reduction from 3SAT.[4]

Theorem2: (P2) is NP-hard

If we let t=1, (P2) is converted to (P1), we say that (P2) is a generalization of (P1), which means (P1) is a special instance of (P2). Since (P1) is NP-hard, we know that (P2) is NP-hard.

Theorem3: (P3) is NP-hard

If the instruction set only contains addition, scheduling those instructions in (P3) is the same with scheduling them in (P2). So (P3) is also a generalization of (P2) and we know (P3) is NP-hard.

Theorem4: (P4) is NP-hard

Obviously, (P4) is a generalization of (P3) because we don't rule out the possibility that all the function units are empty. Therefore, (P4) is NP-hard.

We've shown that the local optimal problem of scheduling under limited function units in Tomasulo's processor is NP-hard in general. So the space complexity of such a general scheduler is at least exponential of n, where n is roughly the number of reservation stations. Because usually n is in the order of hundred, it's impractical to achieve local optimality in Tomasulo's processor.

4.2 Limited CDBs

How about only limited CDBs are available? We'll show that it cannot achieve local optimality, no matter function units is limited or not.

Proposition4: The Tomasulo's processor cannot guarantee local optimality if the number of CDBs is limited.

We show this by giving a counterexample. Since the number of CDBs is limited, without loss of generality, assume there is a CDB conflict executing the following 3 instructions.

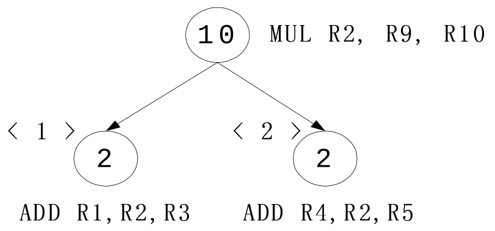

Figure 7: Three Instructions with CDB Conflict

[4] It's a little tricky here. Ullman showed that (P1) is NP-complete for a variable k and didn't rule out the possibility that for each fixed k there exists a polynomial algorithm. In fact polynomial algorithms have been found for k=1 or 2. And it's still an open question for any fixed k where k>=3.

The two additions depend on the multiplication, so they will be dispatched and completed at the same cycle. Suppose only one CDB is available, then there will be a CDB conflict. No matter which addition writes CDB first, the other will be delayed for 1 cycle and it takes totally 15 cycles (execution: 10+2, writing CDB 1+1+1) . However, the local reference processor will use 14 cycles (execution: 10+2, writing registers 1+1) with Scoreboarding because no conflicts will happen when writing back to registers.

If it needs more than one cycle to transmit data on CDB, Tomasulo's algorithm cannot achieve local optimality, no matter whether there are CDB conflicts or not. This is obvious, because local reference processor can write results back to registers in one cycle.

4.3 "Best" Scheduling under Limited CDBs

From above, we know that the Tomasulo's processor with limited CDBs cannot guarantee the local optimality because there might be a CDB conflict, but can we find a best scheduling algorithm for it? The "best" algorithm should not be worse than any other algorithms for the Tomasulo's processor scheduling given limited CDBs, although it cannot reach the local optimality.

Assume the numbers of function units and CDBs are both limited, so there might be both function unit conflicts and CDB conflicts.

Considering the simplest special case, there are k identical function units using 1 cycle to finish and only 1 CDB. This is a typical UET-UCT (Unit Execution Time and Unit Communication Time) scheduling problem. From the previous papers, we know in most cases UET-UCT problems are NP-hard [16][17]. Finta and Liu thoroughly discussed UET-UCT scheduling in single-bus multiprocessor systems [15], similar to the Tomasulo's processor, and they showed that finding a best schedule for tasks with or without pre-allocation (that is a task can only access a part of processors, instead of all) are both NP-hard. Using their result, we get the following theorems.

Theorem5: To find the best schedule for the Tomasulo's processor with limited identical functional units and limited CDBs is NP-hard.

Tasks without pre-allocation in [15] can be reduced to the special case of theorem5 directly: UET-UCT tasks without pre-allocation $\rightarrow$ all instructions can only be issued to any of the k identical function units, which use 1 cycle to finish. Therefore theorem5 is NP-hard.

Theorem6: To find the best schedule for the Tomasulo's processor with limited different functional units and limited CDBs is NP-hard.

Tasks with pre-allocation in [15] can be reduced to the special case of theorem6: UET-UCT tasks without pre-allocation $\rightarrow$ a kind of instructions can be issued to that kind of functions, which use 1 cycle to finish. . Therefore theorem6 is NP-hard.

4.3.1 Single CDB

Although in general scheduling with limited CDBs is NP-hard, we are still interested in using an efficient heuristic algorithm for the 1 CDB case. In the original IBM 360, CDB conflicts were simply resolved by assigning priorities to functional units. We show that could be very bad by giving an example. Suppose all the following instructions are in the reservation stations.

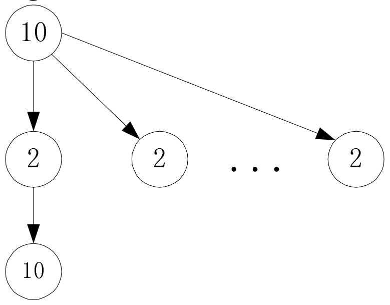

Figure 8: An example of resolving CDB conflicts

There are three level in the DAG, suppose 11 nodes are in the second level, so there will be 11 CDB conflicts at a time. If the priority increases from left to right, resolving CDB conflicts by priority takes 35 cycles (execution: 10+2+10=22, writing CDB: 13).

What about resolving CDB conflicts by critical path? The heuristic critical path approach is widely used in the task graph scheduling [3][8][11]. When there is a CDB conflict, first compute the critical paths of the instructions issued, and then give the priority to the node on a critical path. If more than one or none conflicting nodes are on critical paths, choose one arbitrarily. For this example it takes only 25 cycles (execution: 22, writing CDB: 3) and it is in fact the best.

In a DAG, to find all the critical paths, first add a dummy entry node to all nodes without predecessors and add a dummy exit node to all nodes without successors, then apply a simple labeling algorithm from the exit to the entry, and get all critical paths in the graph. This algorithm can be done in time O(V+E), where V is the number of nodes and E is the number of edges, thus it is very time-efficient.

4.3.2 Resolving n-CDB conflicts

The critical path approach can be extended to resolve n-CDB conflicts easily, just choose n conflicting nodes in critical paths when there is a conflict with n-CDBs (more than n nodes wanting to write n CDBs). If more than n conflicting nodes in critical paths, randomly choose n nodes from those nodes; if less than n conflicting nodes on critical paths, first choose those on critical paths, then randomly choose the left nodes.

Although resolving CDB conflicts by critical path sounds very good, it is not the "best" in all the cases, because the general problem is NP-hard [18].

5 Load/Store Buffers

Now we want to add load/store buffers to our processor model. The same with our assumption about reservation stations, we also suppose the entries in load/store buffers are infinite. So no resource hazards happen due to the lack of entries in load/store buffers, but the buses from the load/store buffers to the memory are limited. Then we need to schedule when bus accesses conflict.

It's obvious that the schedule of load/store instructions is dependent on other kinds of instructions. For example, when two load instructions conflict, the optimal schedule is also dependent on those addition or multiplication instructions that use the data of the two load instructions. So we have to regard the scheduling of load/store instructions and other instructions as a whole.

Since the reservation stations and load/store buffers are all infinite, we can merge these buffers into one. And from the perspective of the scheduler, it is equivalent to that there are some load/store function units and all the load/store instructions are executed in them. Then we modify our Tomasulo's processor model based on this idea, as shown in figure 9.

This scheduling problem actually is an extension of the problem we have discussed before. In those problems, we suppose there are only two kinds of function units, adder and multiplier. In fact all our proofs on complexity do not rely on such an assumption. Thus it is still NP-hard if we want to achieve local optimality with more kinds of function units. So we conclude the general scheduling problem with load/store buffers is also NP-hard.

6 Related Work

In recent years, a lot of work has been done on the verification of the correctness of Tomasulo's algorithm [1][9][10][12]. Although the methods of verifications are totally different from our proof of optimality, they provided some models for Tomasulo's algorithm and our model of Tomasulo's processor comes from the model used in [9].

Tomasulo's algorithm is a scheduling problem per se. A lot of research was done in 1970's on the various scheduling problems. Good surveys on these works can be found in [6][7][2]. Although Tomasulo's algorithm was born in 1960's, it was not discussed in these papers. But these works did become the basis of our analysis. In [14], it was proved that the problem of single executing time scheduling is NP-Complete. This conclusion makes some of our proofs on the complexity of Tomasulo's algorithm straightforward.

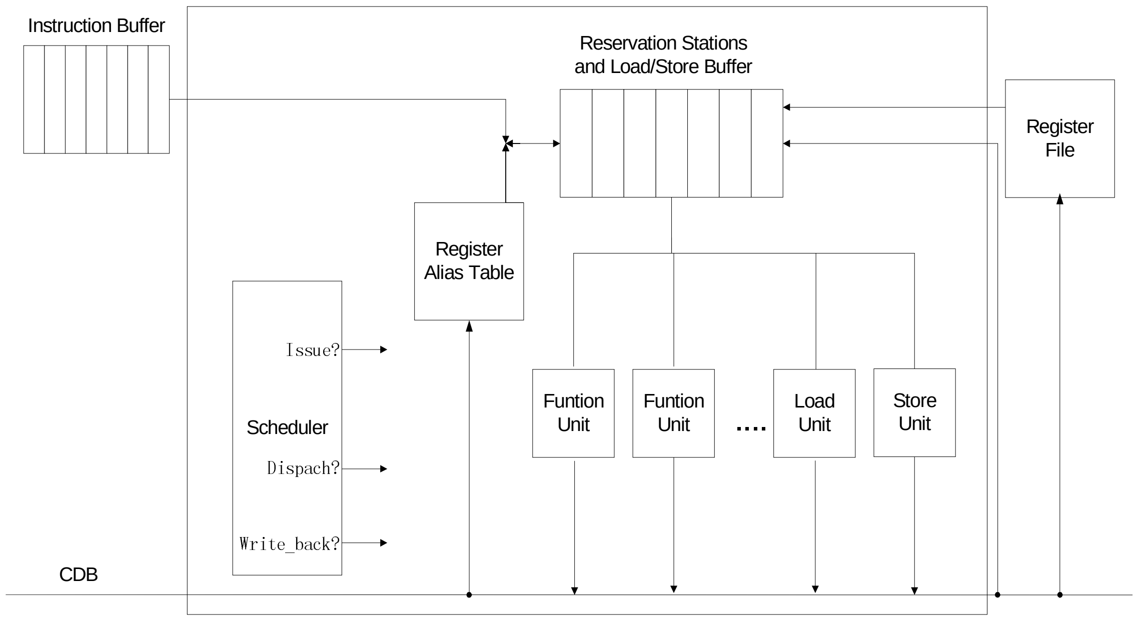

Figure 9: Tomasulo's Processor with Load/Store Buffer

Assumption		Optimality			Best
Function Unit	CDB	Global	Constrained	Local	
∞	∞	No	Yes	Yes	N/A
K	∞	No	No	NP-hard	N/A
K	N	No	No	No	NP-hard

Table 1: Our Conclusions

However, in [14] it didn't take communication delays into account. That's not very realistic. In our discussion of infinite CDBs, the 1 cycle CDB delay can be included in the cycles of functional units. That is reasonable because if there is no CDB conflict, we can always treat a functional unit and a CDB as the same block and bind them together. But when there are limited CDBs, we can't bind them together anymore. If both CDB and function units are limited, we must resolve both CDB and function unit conflicts, which are related to each other. In Tomasulo's processor, the communication delay is a fixed number, and then it becomes a UET-UCT scheduling problem, [16][17][18][19]. In [15], they proved scheduling in single-bus multiprocessor systems is NP-hard, and that can be easily reduced to our models.

Many papers discussed using heuristic algorithms for scheduling problems, where the critical path approach is widely used [3][8][11]. In fact, we can resolve both CDB and function units conflicts by critical path and get decent results.

Since the middle of 1980's, a lot of research on scheduling problems fall into the framework of integer linear programming (ILP) [4][5]. ILP provides a uniform format for scheduling problems. Although we also formulize Tomasulo's scheduling using ILP, we have not found it helpful to investigate this problem.

7 Conclusion

We conclude our results in table 1.

References

[1] T. Arons and A. Pnueli. Verifying Tomasulo's algorithm by refinement. Technical report, Welzmann Institute, 1998.

[2] K.M. Baumgartner and B.W. Wah, First Workshop on Parallel Processing, Taiwan. 1990

[3] Bjorn-Jorgensen, P. and Madsen, J. Critical path driven cosynthesis for heterogeneous target architectures. Proceedings of the 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE '97)

[4] Samit Chaudhuri and Robert A. Walker, IPL-based scheduling with time and resource constraints in high level synthesis. Proceedings of VLSI Design'94, pages 17-25, 1994

[5] Samit Chaudhuri, Robert A. Walker, and John E. Mitchell. Analyzing and exploiting the structure of he constraints in the ILP approach to the scheduling problem. IEEE Transactions on Very Large Scale Integration Systems, Vol. 2, No. 4. December 1994

[6] M. J. Gonzalez, Jr. Deterministic Processor Scheduling. Computing Surveys, Vol. 9, pages 173-204, September 1977.

[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling. Annals of Discrete Mathematics 5, pages 287-326, 1979

[8] C.V. Ramamoorthy, K.M.Chandy, and Marjo J. Gonzalez, Optimal scheduling strategies in a multiprocessor system. IEEE Transactions on computers, VOL. C-21, NO.2, Feb, 1972

[9] Ravi Hosabettu, Ganesh Gopalakrishnan, and Mandayam Srivas. A Proof of Correctness of a Processor Implementing Tomasulo's Algorithm Without a Reorder Buffer. In Advances in Hardware Design and Verification: IFIP WG10.5 International Conference on Correct Hardware Design and Verification Methods (CHARME '99), volumn 1703 of LNCS, pages 8-22, Springer-Verlag, 1999.

[10] Ravi Hosabettu, Mandayam Srivas, and Ganesh Gopalakrishnan. Decomposing the proof of correctness of pipelined microprocessors. In Computer-Aided Verification, CAV '98, pages 440-451, 1998.

[11] Hu, T. C. "Parrallel sequencing and assembly line problems", Operations Research 9,6 (1961), 841-848

[12] Daniel Kroening, Silvia M. Mueller, and Wolfgang J. Paul. A Rigorous Correctness Proof of a Tomasulo Scheduler Supporting Precise Interrupts

[13] R.M. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. In IBM Journal of Research and Development, volumn 11 (1), page 22-33, 1967.

[14] J. D. Ullman. NP-complete scheduling problems. Journal of Computer and System schiences 10, pages 384-393, 1975

[15] L. Finta, Z. Liu, "Scheduling of Parallel Programs in Single-Bus Multiprocessor Systems", Rapport de Recherche INRIA, No. 2302, 1994

[16] Rayward-Smith, V.J. UET Scheduling with Unit Interprocessor Communication Delays and Unlimited Number of Processors. Discrete Applied Mathematics. 18, pp. 55-71, 1987.

[17] Picouleau, C. Etude de Problems d' Optimization dansles Systemes Distribues. These, Universite Pierre et Marie Curie, 1992.

[18] Papadimitriou, C., and Yannakakis, M. Toward an Architecture-Independent Analysis of Parallel Algorithms. SIAM J. Comput. 19, pp. 322-328, 1990. Extended Abstract in Proceedings STOC 1988.

[19] Andronikos, T., Koziris, N. Papakonstantinou, G. and Tsanakas, P. Optimal Scheduling for UET-UCT Generalized n-Dimensional Grid Task Graphs, Proceedings of the 11 th IEEE International Parallel Processing Symposium (IPPS97), pp. 146-151, Geneva, Switzerland, 1997