Micro-Analysis of the Titan's Operation Pipe

John Sanguinetti
Ardent Computer Corporation

Abstract

Much of the performance analysis done in designing a computer is based on fundamental operation rates, like cycle time and number of pipe stages in an operation pipeline. This kind of analysis yields peak computation rates which, in fact, may never be realized. Resource contention between different units, each of which has a fundamental operation rate adequate to support a given overall peak, may cause the actual obtainable rate to be much less. In order to determine the effects of interactions between different requestors and common resources, micro-analysis, or detailed modelling of the part of the system in question, is necessary. In this paper, we report on the results of such an analysis on the operation pipe of the Titan graphics supercomputer. The Titan is a new class of machine with a supercomputer-style architecture implemented in a technology appropriate for a single-user machine. The micro-analysis reported here resulted in enhancing the actual obtainable computation rate from 10.8 Mflops to 14.6 Mflops for a particular real application, while the fundamental operation rate is 16 Mflops.

Description of the Titan

The Titan was conceived to be a "Visualization Tool" — a machine which would allow an engineer or scientist to model a physical entity and then visualize the results of the model. Consequently, the machine would have to first be able to do the computation required for physical modelling and second be able to render the resulting image, all in a reasonable amount of time. The goal of the Titan was to provide a significant fraction (about 25%) of the performance of a super-computer and provide superior graphics performance for a price less than $100,000.

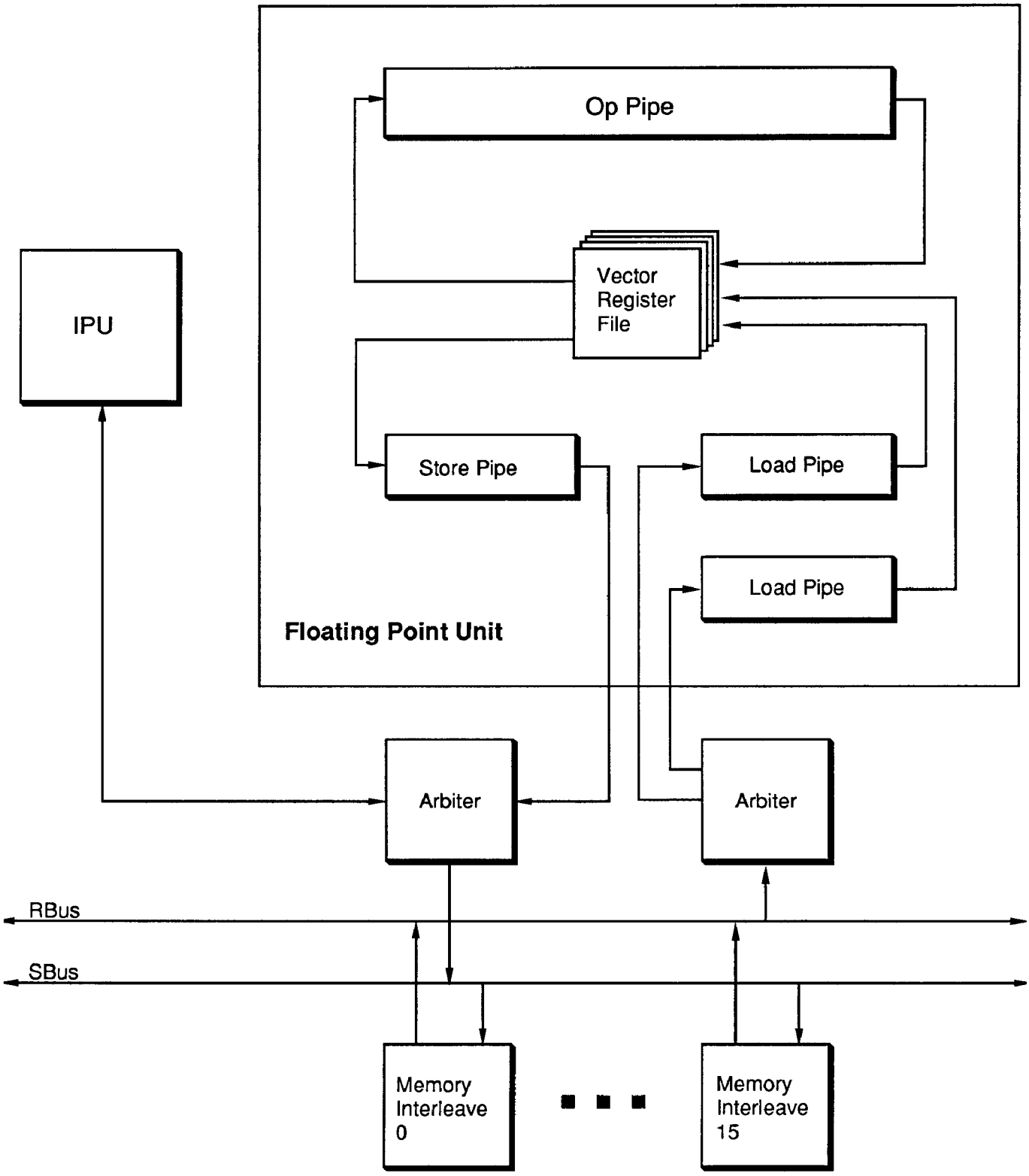
Particular performance goals included executing the Linpack 100x100 benchmark at a rate of 6 Mflops (compiled) on a single processor. This was considered an important performance goal since many real problems have characteristics similar to the Linpack benchmark, and typical supercomputer rates on this benchmark range from 20 to 40 Mflops — a Cray 2S (1 processor) is rated at 23 Mflops [Dongarra].

Aside from computational power for physical modelling problems, floating point power is also needed for doing 3-D graphics rendering. Drawing a picture on a 1024x1280 screen which occupies 1/2 of the screen area using a ray-tracing algorithm with shading, 3 light sources, and environmental reflection would typically take about 3.5 billion floating point operations. Titan was intended to have enough computational power to draw such pictures in "reasonable" time. A rate of 2 Mflops for scalar computation would allow such rendering to be done in about 8 minutes on a 2 processor Titan.

In order to satisfy its performance goals, the Titan needs a fast processor and a high bandwidth processor-to-memory interconnection. In order to do double-precision vector triad operations, the processor-to-memory connection must deliver 16 bytes to the processor and 8 bytes to memory for each operation, making a requirement of 24MB per Mflop (this requirement can be reduced by chaining in the floating point unit for some operations).

The Titan is a symmetric multiprocessor, with up to four identical processors. The organization of the processor is fairly typical of a vector computer. There is an integer processor (IPU) and a floating point unit (FPU). The FPU contains two load pipes to memory, one store pipe, a single operation pipe which can accommodate a chained multiply and add, and a reasonably large set of vector registers (8192 cells) which can be configured in a variety of vector sizes. Figure 1 is a block diagram of the processor and memory of the Titan.

The fundamental rates of the various units are matched so that a peak of one result every 2 clock cycles can be



TITAN Block Diagram
Figure 1.

achieved. Each load pipe can deliver one operand from memory to the vector register file every 2 cycles, the vector register file can deliver 2 operands every 2 cycles to the operation pipe, the operation pipe can deliver one result every 2 cycles to the vector register file, the vector register file can deliver one result every cycle to the store pipe, and the store pipe can deliver one result to memory every cycle. Finally, the interleaved memory can receive and deliver results and operands at a rate of 1 operand and 1 result every cycle. Since the clock frequency is 16 MHz, floating point results can be delivered to memory at a frequency of 8 Mhz. For add and multiply operations that is a processing rate of 8 Mflops, and for chained multiply and add operations, that is a rate of 16 Mflops.

The peak processing rate has been defined as the rate which the manufacturer guarantees the machine will not exceed. Since the operation pipe can execute at 16 Mflops, the peak rate is 16 Mflops. A load-load-add-multiply-store chained operation will have a sustainable rate of 16 Mflops, as long as no contention occurs anywhere in either memory or the vector register file. In real operation, contention does sometimes occur, and the question of interest is how close to the peak rate can sustained operation run. The detailed design of the various units can have a dramatic effect on the answer to that question.

In the case of the Titan, micro-analysis of the operation pipe (that is, a cycle by cycle analysis) revealed that a non-intuitive change in the design could increase the actual sustainable rate for a given benchmark program from 10.8 Mflops to 14.6 Mflops. The code being executed here was Daxpy (figure 2), which is the dominant part of the Linpack benchmark. The modelling was done using a mixed-level model of the Titan, with the relevant components of the vector unit modelled at the gate level. This is a slow, but absolutely accurate, model of the vector unit (in a sense, the model *is* the design).

```
      do 1 i = 1, n
1       y(i) = y(i) + a*x(i)
```

Figure 2.

The Operation Pipe

The components of the floating point unit involved with a typical triadic vector operation are the load pipes, the operation pipe, the store pipe, and the vector register file (VRF), which serves as source and destination for the operands and results. The central component is the operation pipe, or op pipe. This pipeline has a number of operational characteristics which affect the overall FPU operation.

The fundamental characteristic of the op pipe is that it operates at half the frequency of the rest of the FPU. That is, each stage of the pipeline requires two cycles. This comes from the use of an off-the-shelf arithmetic chip (the Weitek 2264/2265), which must be cycled at a rate slower than 16 MHz. The pipeline is divided into sections, as shown in figure 3. The first section fetches operands from the VRF and the second section presents the fetched values to the appropriate arithmetic chip. The third section consists of stages 3, 4, and 5, each of which takes two cycles. This is where the arithmetic is actually performed. Stages 6, 7, and 8 are an optional section, which does a second operation in the case of a chained multiply and add. The last section is stage 9, where results are stored in the VRF. All of the stages within a given section must be in the same phase (A or B), and there can be only one entry in a single stage at a time.

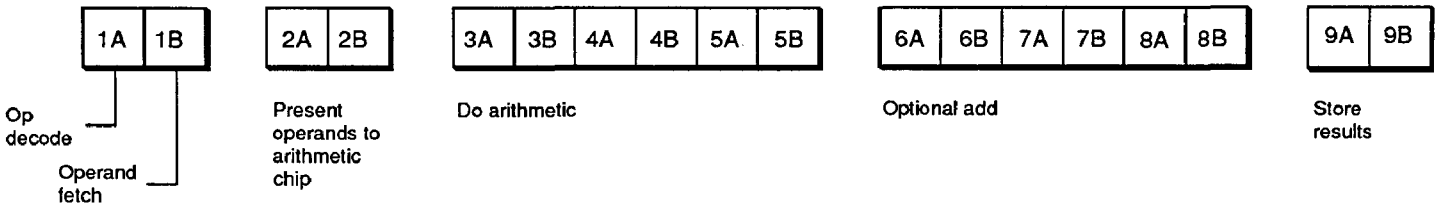
In this pipeline, stalls can occur when there is either an operand-result dependency or contention in the VRF inhibits loading operands or storing results. These occur in stages 1B, 5A (for chained operations), and 9A. Another source of stalls is the requirement that stage 2 and stages 3-5 must stay in phase with each other. That is, an entry in stage 2A cannot move to stage 2B if an entry in stages 3-5 is moving from the B phase to the next stage A phase. Typically, stalls due to VRF bank contention last for one cycle, and the same is true for out-of-phase stalls.

Standard pipeline design (see [Kogge]) follows a greedy strategy. An entry in the pipe moves to the next stage unless stalled by either a stall condition for the destination stage, e.g. operand fetch stalled, or the stage ahead is stalled. Events which occur in earlier stages in the pipeline (later in the stream of operations) do not affect stalling behavior in later stages.

The Vector Register File

The vector register file is a shared resource, serving as the source and destination for all the FPU units. As a result, it is subject to contention when two units need to access vector registers on the same cycle. The bandwidth of the VRF was increased by dividing it into four banks, each of which can be accessed by one of the units on each cycle. Thus, if each unit is accessing a different bank, no contention occurs.

For example, figure 4 shows the op pipe accessing two operands from different banks on the same cycle, as both operands are obtained in stage 2A. At the same time, the store pipe can access a different bank, obtaining a result to store in memory. Note that results are stored from the op pipe stage 9 on the opposite phase as operand fetch, so



Operation Pipeline

Figure 3.

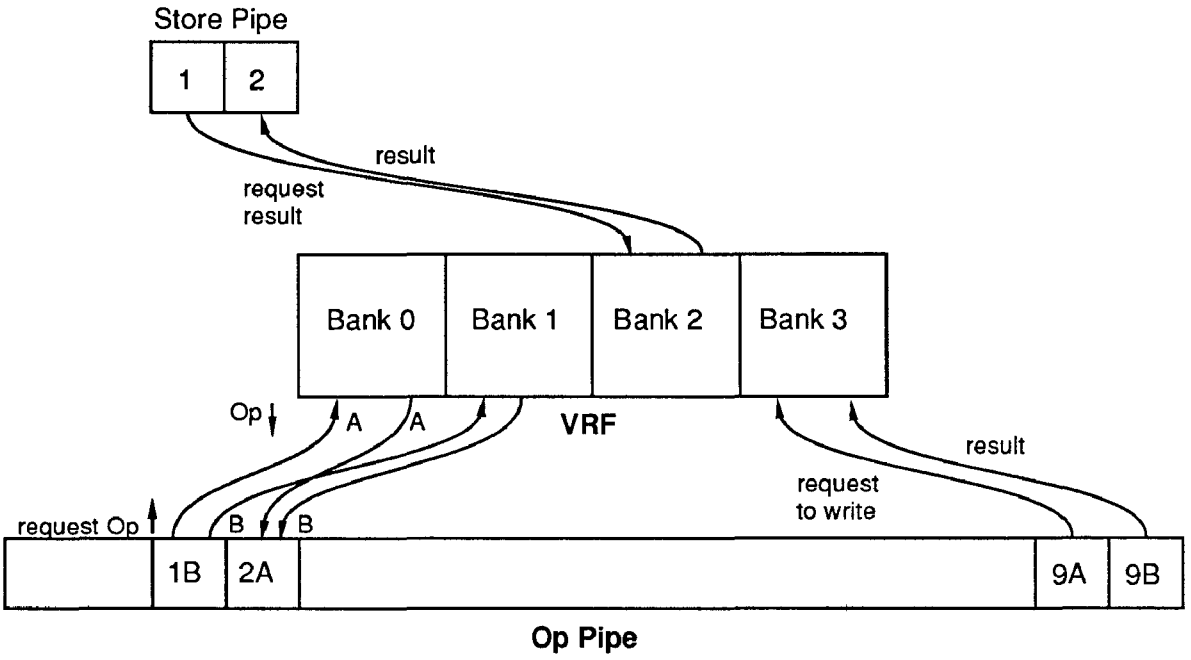


Figure 4.

contention is avoided even in the same bank between operands and results.

If two units do access the same bank on the same cycle, the conflict is resolved by priority, with writes into the VRF having higher priority than reads from it. This typically happens with the load pipes contending for the same bank as operand fetch. If a load pipe write and an operand read occur to the same bank on a given cycle, the operand fetch is stalled.

A consequence of this conflict resolution is that if load pipes were allowed to run at a rate of one element per cycle, which the bus and memory system can support, the op pipe would be locked out of that bank. For this reason, load pipes are limited to produce one element every two cycles, though for a triadic operation, they would typically alternate cycles anyway.

Performance Effects

The fundamental rates of the various units are matched so that the op pipe should be the limiting resource, emitting results every two cycles. Indeed, if the triad being performed is strictly done within the vector register file, that is always the case. However, in the common case where operands come from memory, are chained to the operation, and the results are chained to the store pipe to go back to memory, VRF contention does occur.

When the initial design was simulated, the operation rate of the Daxpy loop, which is essentially a single load-load-multiply-add-store chained operation, was 10.8 Mf, not the peak rate of 16 Mf which would be obtained if the operation was contained within the VRF. In order to determine the cause of this lower than expected operation rate, micro-analysis, or detailed modelling, of the FPU was used.

Micro-analysis

Micro-analysis of a performance phenomenon is the detailed modelling of a small part of the system, from which the system's performance is determined [Beizer]. This is most appropriate to the analysis of that part of the system which is the primary performance factor. Such analysis usually takes the form of an analytic model on a restricted enough part of the system to make solution feasible. In the problem at hand, micro-analysis is applicable, but a tractable analytic model is not apparent.

The modelling technique used was a mixed-level simulation model of the Titan. The model incorporated different levels of abstraction for different components of the machine. The highest levels of abstraction were used for those components which were not involved in the operation in question, for example the I/O and graphics boards. Intermediate levels of abstraction, which generally retained the important behavior of a component but did not include the unused functionality, were used for models like the memory boards, the integer processor, and the arithmetic chips. This level is sometimes called the functional, or behavioral level.

The register transfer level is a lower level of abstraction, in which the structure of the component is modelled, but the mechanism for transforming data is modelled abstractly (for example, x = y + z would model the collection of gates which make up the adder). This level was used for the cross-bar switch which routes data between the FPU units and the VRF. The lowest level of abstraction is the gate level, in which the component is modelled by its actual netlist. Gate level models were used for the op pipe and the load and store pipes, as well as the VRF controller and scoreboard. In a very real sense, the model *is* the component being modelled.

Mixed-level modelling was proposed in the context of operating system design by [Zurcher]. The idea was that successive refinement would lead in a top-down fashion to progressively more complete simulation models, until ultimately the final model was the operating system itself. This methodology has not gained wide acceptence for doing operating system design. However, it is proving successful in hardware design. A gate level simulation model can automatically be transformed into a netlist from which components can be fabricated.

For the purposes here, a major advantage of mixed-level simulation was that it facilitated micro-analysis of the FPU. Those components which were involved with the operation in question could be elaborated at the gate level, while those which were involved to lesser degrees could be represented by more abstract models. Figure 5 shows the various levels of abstraction in the Titan model used to analyze the op pipe phenomenon here.

Among the tangible benefits of this method were reasonable execution time of the simulation model. The model was simulated on a Sun 3/260 workstation at a rate of about 3 simulated cycles/second, which was fast enough to do a simulation run in a matter of minutes (or a small number of hours). The simulation model was written in Verilog-XL ([Gateway]).

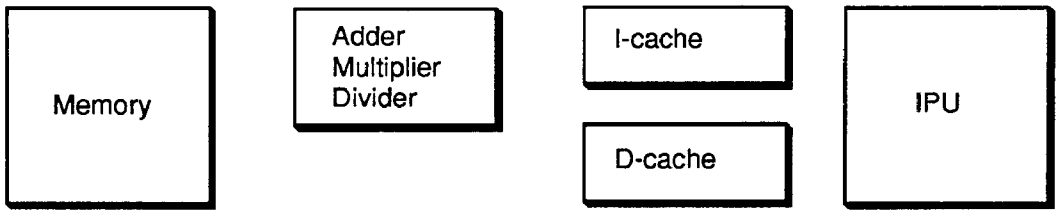
Modelling Results

When the Daxpy operation was simulated in the system model, a sustained rate of 10.8 Mf was obtained. The cause of this degradation can be found in the interaction between operand fetch stalls and phase stalls in the op pipe.

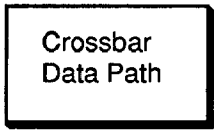
Because of more or less random variations in the memory system, occasionally the load pipes will deviate from the normal alternating pattern of returning operands. These random variations are due to memory traffic generated by other system components, such as the integer processor, other processors, or I/O. Figure 6 shows an example of typical bus activity for the two system buses at the beginning of this vector operation. Note that at cycle 16 there are two consecutive references to the same bank (pipe b). After that, the two pipes begin alternating again.

When one of these deviations occurs, the op pipe reaches a state where repeating stalls reduce the pipeline throughput by a factor of two. The two stalls in question are the VRF bank contention stall in 1B and the phase stall in 2A. Figure 7 shows how this occurs. Each line in the figure corresponds to a cycle. VRF accesses by the load pipes are indicated by the pipe identifier (a or b), and the operation element number is given for each op pipe stage. An entry in 1B will stall if it occurs on the same cycle as a B pipe VRF access (because it is accessing the same bank). The first entry marked by a * shows a VRF bank conflict stall, which was a result of the deviation in returned data.

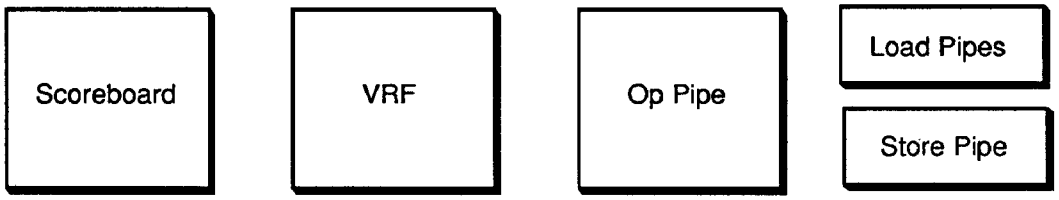
The stall in 1B causes the two pipe sections 2 and 3-5 to get out of phase with each other, causing a phase stall in the entry marked by a +. Thus the VRF bank contention stall has turned into a 2-cycle stall. Since the load pipes have resumed their alternating behavior, however, a second VRF bank stall will occur on the next operand fetch, and the process has reached a recurring state. This is very similar to the linked-conflict phenomenon in the Cray X-MP memory system described by [Cheung].



Functional Level



Register Transfer Level



Gate Level

Figure 5.

---

cycle	11	12	13	14	15	16	17	18	19	20
Pipe A		a		a			a		a	
Pipe B	b		b		b	b		b		b

Load Pipe VRF Requests

Figure 6.

Cycle	Vrf access	1b	2a	2b	3a	3b	4a	4b	5a	5b	6a	...
15	b		4		3		2		1			
16	b	5*		4		3		2		1		
17	a	5			4		3		2		1	
18	b		5+			4		3		2		1
19	a	6	5				4		3		2	
20	b	6*		5				4		3		2
21	a	6			5				4		3	

Operation Pipe With Recurring Stalls

Figure 7.

Cycle	Vrf access	1b	2a	2b	3a	3b	4a	4b	5a	5b	6a	...
15	b		4		3		2		1			
16	b	5*		4		3		2		1		
17	a	5			4		3		2		1	
18	b		5			4+		3+		2+		1+
19	a	6		5		4		3		2		1
20	b		6		5		4		3		2	

Operation Pipe With Downstream Stalls

Figure 8.

The solution to this problem is relatively straightforward, though counter-intuitive. If, instead of stalling the element in stage 2A to regain correct phase ordering, the downstream operations are stalled (stages 3-5), then the next operand fetch will happen one cycle earlier, and will not suffer a VRF bank conflict. This is illustrated in figure 8.

When this solution was implemented in the simulation model, the operation rate for Daxpy increased to 14.8 Mf. The difference between the new rate and the maximum of 16 Mf is now solely a result of the occasional memory deviations, which cause 1-cycle stalls in the op pipe. When the Titan was actually built and this operation was measured, the modelled performance was obtained. As a result, the complete Linpack 100x100 benchmark executes at a rate of more than 6 Mflops on a single processor Titan.

Conclusion

By using micro-analysis of the operation pipe and its interactions with the vector register file, we were able to modify the design to improve the sustainable processing rate 35 percent on a significant part of the Linpack benchmark. Because this is a common operation in many programs of interest, the overall effect is significant.

References

1. Beizer, Boris. *Micro-Analysis of Computer System Performance.* Van Nostrand Reinhold Co. New York, 1978.

2. Cheung, Tony and James E. Smith. "A simulation Study of the Cray X-MP Memory System" in *IEEE Transactions on Computers*, Vol. C-35, No. 7, July 1986.

3. Dongarra, Jack J. "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment" Technical Memorandum No. 23. Argonne National Laboratory, Feb. 1988.

4. Gateway Design Automation Corp. *Hardware Description Language and Simulator.* Gateway Design Automation Corp., Westford, Mass., March 1987.

5. Kogge, Peter M. *The Architecture of Pipelined Computers.* McGraw-Hill Co., New York, 1981.

6. Zurcher, Frank and Brian Randell. "Iterative, Multi-Level Modelling—a Methodology for Computer System Design" in Proceedings IFIP Congress 68, 1968.