Balancing System Level Pipelines with Stage Voltage Scaling

Hui Guo, Sri Parameswaran
School of Computer Science and Engineering, The University of New South Wales
Sydney, NSW 2052, Australia
{*huig,sridevan*}*@cse.unsw.edu.au*

Abstract

This paper presents an approach to dynamically balance the pipeline by scaling the stage supply voltages. Simulation results show that by such an approach about 50% improvement in throughput and response time, and 11% improvement in power consumption can be achieved with limited memory overhead.

1 Introduction

Pipelining is an essential method for designing high performance systems. It has been applied at different levels of design.

For a system level pipeline, it is possible to have the supply voltage of the hardware modules scaled. The scaled voltages, in turn, can be used in changing speed, reducing power consumption and even in balancing pipeline stages (as will be demonstrated in this paper).

However, unlike lower level pipelines (such as at the RTL or the instruction level), system level pipelines have some specific design issues. One of them is the coarse grain of components (sub-systems) which a design is based on. These subsystems usually exhibit different execution times for different tasks. As such, traditional static pipelining with a fixed stage execution time is not practical.

Figure 1(a) shows an example of three components where each component performs a specific function or behaviour. There are four tasks processed in this system, with the execution time in each component shown in the brackets. If we pipeline this system into three stages (each component as a stage) and take the worst case into account, the stage execution time is 20 ms. The three stages perform in parallel on different tasks, and tasks flow in the pipeline at the same pace. As shown in Figure 1(b), the four tasks take 120 ms to complete. Obviously, this pipeline is very inefficient as the worst case rarely occurs. The pipeline idles for most of the time.

If we allow each stage to work at its own pace by providing enough buffering devices to store tasks transferred between stages, then the four tasks only take about 50 ms to finish, as illustrated in Figure 1(c). However, stages in this system are likely to be un-balanced. Fast stages will demand more memory to buffer the completed tasks. More waiting time may occur for some tasks. If we can change the stage supply voltages to dynamically adjust the task execution time (which is inversely related to the voltage), a more balanced pipeline can be achieved, as shown in Figure 1(d). We first increase the voltage of stage 2 (hence reducing its execution time) and reduce the voltage of stage 3 (hence increasing the execution time). We then increase the voltage of stage 3 when task 3 comes, to obtain a more balanced pipeline and the four tasks take less time to complete ($< 40ms$).

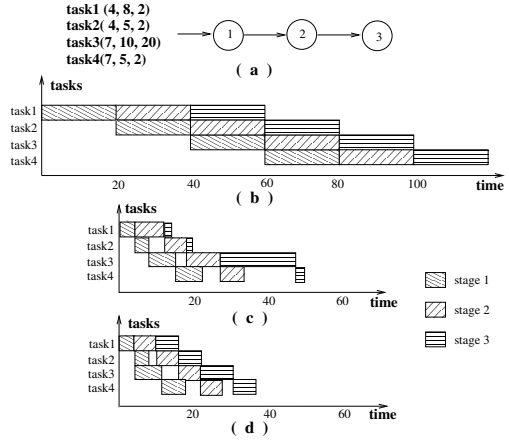

Figure 1. Example system

System level pipelining has been addressed in the past [1] [2] [3] [4] [5] [7] [6] [8] [9]. The issues that have been addressed mainly focused on partitioning and scheduling. All the existing works done so far are based on the assumption that the execution time of each sub-system is known and fixed. We, for the first time, tackle the problem when the execution time of each sub-system can be different.

Our approach is explained in the next section with the simulation results shown in section 3. The conclusion is given in section 4.

2 Dynamic pipeline with stage voltage scaling

For a system with varied execution time and different stage data size, traditional pipelining will result in an inefficient and sometimes impossible design. Here we provide a new type of pipeline, called *dynamic pipeline*. As compared to the traditional model, there are two features in the new model: first, the buffering devices between stages are changed from fixed-size registers to FIFOs (memory with First In First Out data

Proceedings of the IEEE Computer Society Annual Symposium on VLSI
New Frontiers in VLSI Design
0-7695-2365-X/05 $20.00 © 2005 **IEEE**

operation scheme) to accommodate data from a varied number of tasks; second, the flow of tasks within the pipeline is asynchronous - a stage, after it completes a task, can output to the next stage and start a new one without having to wait for other stages. Therefore, the stage resource can be fully utilised and the pipeline execution can be sped up.

With the dynamic pipeline model, tasks may queue in a FIFO if the next stage is jammed with a big task and this queue can become longer if such a situation persists. Therefore, the time a task spends in a pipeline is not determined by the number of stages and the stage execution time; instead, it is decided by the execution times of this task in each stage and the waiting times in each FIFO.

To reduce the waiting time and minimise average response time and increase the throughput, it is desirable that each stage have equal execution time at any time. To maximise the possibility of meeting equal stage execution time and increase the feasibility for stage voltage scaling, we take the average execution time of each implementation as our initial design inputs. The system is first partitioned and pipelined into the minimum number of stages by using the approach proposed in [9]. All stages have equal or near equal execution times and are balanced based on the average execution times. FIFOs are used in the pipeline to allow stages to work asynchronously.

This balance is then maintained and reinforced by a stage voltage controller to dynamically adapt the execution time changes between stages when the pipeline is in operation.

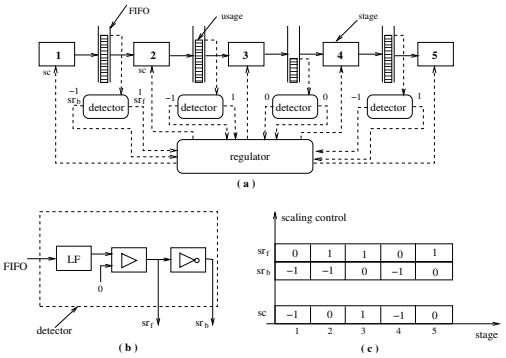

Figure 2. Pipeline with stage voltage scaling

Figure 2(a) illustrates a five-stage dynamic pipeline with the stage voltage controller. The controller includes several detectors and a regulator. Four FIFOs are used in this pipeline. Each FIFO has two related stages and an associated detector. The usage of each FIFO is used by the detector to detect the imbalance of the two stages on either side of the FIFO. The outputs, sr_b and sr_f, of the detector represent the requirements for scaling stage voltage, sr_b is for the previous stage and sr_f is for the next stage. When the two stages are out of balance, $sr_b = -1$ and $sr_f = 1$, meaning that either slowing down of the previous stage or speeding up of the next stage is required. Value -1 denotes the requirement for the voltage scaled down and value 1 for the voltage scaled up. When the two stages are balanced, both sr_b and sr_f are zero, there is no need to change the voltage.

These requirements from all detectors are then processed by the regulator to determine the actual stages that need to be voltage scaled. The output sc to each stage controls the supply voltage of the stage.

Figure 2(b) shows the structure of one detector. It contains a low pass filter, a comparator and an inverter. The low pass filter is used to obtain the trend of FIFO usage, denoted by $\bar{\delta}$. When $\bar{\delta} > 0$, $sr_f = 1$; otherwise, $sr_f = 0$. The inverter provides opposite value of sr_f.

The rules of the regulator are as follows.

- For a stage, there are two scaling requirements from the detectors on either side of the stage. If one requirement is to scale down the voltage and another is to scale up the voltage, then no scale change is undertaken.
- sr_b has higher priority than sr_f. In a case where two stages i and i+1 are out of balance, either scaling down stage i or scaling up stage $i+1$ is allowed. Scaling down stage i should be the first choice to achieve a possible power reduction.
- If more than two contiguous stages are unbalanced, among them, only the first stage has the voltage scaled down and the last stage voltage is scaled up. The voltages of the middle stage remain unchanged.

Figure 2(c) shows the outputs of the detectors and the related results from the regulator for each stage. The first stage has only one signal, sr_b, from the detector, sr_f is, therefore, assigned to be always 0. The similar is in the last stage, where sr_f is constantly 0. The three pairs of stages are unbalanced. The imbalance of stage 1 and 2 requires stage 1 to be slowed down or stage 2 to be sped up; while the imbalance between stage 2 and stage 3 needs stage 2 to be slowed down or stage 3 to be sped up. As well, the detector between stage 4 and stage 5 suggests that stage 4 should slow down or stage 5 should speed up. The final decision made by the regulator is to scale down stage 1 and 4, and scale up stage 3.

3 Simulation results

Five systems are simulated with three types of designs: the conventional pipeline, the dynamic pipeline without stage voltage scaling and dynamic pipeline with stage voltage scaling.

Figure 3 shows the average response time, throughput time, FIFO usage and power consumption with different numbers of tasks. As can be seen, with the stage voltage scaling, the average task response time and FIFO usage are well under the control and are independent from the number of tasks processed . Without voltage scaling, the response time and FIFO usage grows as more tasks are processed. Also it can be seen the average throughput time and power consumption are quite stable in both working conditions as the number of tasks increases. But the higher throughput results in higher power consumption.

Table 1 summarises the metrics of the five simulated systems pipelined with dynamic model under the two working conditions: with and without stage voltage scaling. Column 1 gives the average values in response time ($\bar{T}_r$), throughput time ($\bar{T}$), FIFO usage ($\bar{R}$), and power consumption ($\bar{P}$) of

Proceedings of the IEEE Computer Society Annual Symposium on VLSI
New Frontiers in VLSI Design
0-7695-2365-X/05 $20.00 © 2005 **IEEE**

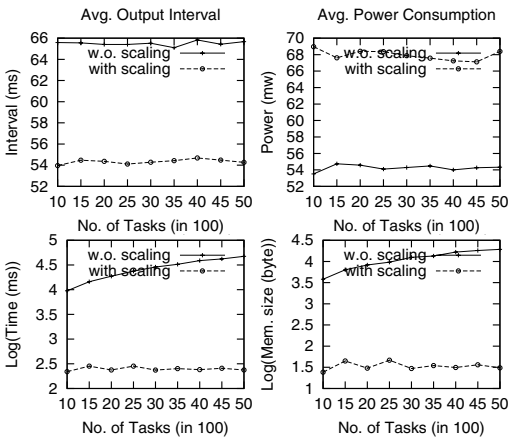

Figure 3. Pipelines with/without voltage scaling

the the dynamic pipelines without stage voltage scaling. The corresponding values with stage voltage scaling are given in Column 2. All the values are in percentage compared to the metrics of the systems pipelined with the conventional model in the worst case. The last row gives the average values of the five systems. Based on these values, we can see that the *dynamic pipeline* always provides high throughput, but in the case without stage voltage scaling, this improvement is at the cost of more FIFOs and a much higher response time due to the imbalance of the pipeline stages, and if the imbalance persists, the cost may grow. This is reflected in System II and V, where the average response times and FIFO usages are much higher than those in the conventional pipelined systems. On the other hand, with voltage scaling, the pipelines are dynamically balanced, which provide both high throughput and quick response time. Also, the memory for FIFOs is limited, and the power consumption is more likely reduced because of the low power oriented scheme used by the controller in the pipeline. Take system IV as an example. The response time is only 39.3% of the normal pipeline, and the throughput is increased by nearly 30%, while there is only 35.7% of FIFO usage and 67.6% power consumption. The increased power consumption can sometimes happen when the stage voltages are scaled up more than they are scaled down, as is the case in System V, where 3% more power is consumed. However, this higher power consumption brings about a greater improvement in throughout with the throughput time reduced to 48.6% of the conventional pipeline, as compared to 58.6% in the case without voltage scaling. The measurements of the five systems show that the average improvement in throughput, response time and power consumption are 42%, 55% and 11%, respectively, with only a little additional cost in memory for FIFOs.

4 Conclusion

The coarse grain of sub-systems in the design makes it difficult to design a pipeline with well balanced stages. To tackle this problem, for the first time, we proposed a new pipeline model, the *dynamic pipeline*, where stages in a pipeline perform concurrently but output asynchronously. A stage can

	dy. pipe. without scaling (%)				dy. pipe. with scaling (%)			
sys.	$\bar{T}_r$	$\bar{T}$	$\bar{R}$	$\bar{P}$	$\bar{T}_r$	$\bar{T}$	$\bar{R}$	$\bar{P}$
I	79.3	79.8	15.3	100	54.5	89.3	10.2	85.7
II	1180	73.4	183	100	84.3	61.8	121	90.8
III	199	20.3	582	100	41.5	21.5	471	95.8
IV	51.9	63	9.2	100	54.1	70.4	35.7	67.6
V	1706	58.6	4925	100	39.3	48.6	31.1	103.1
avg.	643	59	1143	100	54.7	58.3	134	88.6

Table 1. Simulation results

perform more tasks than others in a certain period of time and the tasks between stages are buffered in FIFOs to maintain the integrity and the order of processing.

The dynamic balance of stages in the pipeline is achieved by using stage voltage scaling. The voltages are changed dynamically, during the pipeline execution, in response to the accumulation of tasks buffered in the FIFOs.

We have simulated a number of systems with three designs: the conventional pipeline, the dynamic pipeline without stage voltage scaling and the dynamic pipeline with stage voltage scaling. Simulation results show that the dynamic pipelines both with and without stage voltage scaling can provide higher throughput than the conventional pipelines. With stage voltage scaling, a dynamically balanced pipeline with high throughput, limited memory, quick response time, and potential lower power consumption, can be obtained. The improvement in throughput and response time is about 50%, and the power consumption improvement is about 10% with little additional cost in memory, compared to the system pipelined with the conventional model. It must be pointed out that the simulations were performed with the tasks of execution times within bounded ranges. Tasks that take unbounded execution times are not considered in this paper.

References

[1] Smita Bakshi and Daniel D. Gajski. Hardware/software partitioning and pipelining. In *Proceedings of Design Automation Conference*, 1997.

[2] Hui Guo and Sri Parameswaran. System Level Pipelining. In *1996 APCHDL Conference Proceedings*, pages 28–33, 1996.

[3] Hui Guo and Sri Parameswaran Unrolling loops with indeterminate loop counts in system level pipelines *The Proceedings of the ASP-DAC'98 Design Automation Conference*, pages:99-104, 1998

[4] J.P. Castellano, D. Sanchez, O. Cazorla and A. Suarez Pipelining-based tradeoffs for hardware/software codesign of multimedia systems *The Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing*, pages: 383-390, 1999

[5] S. Bakshi and D. D. Gajski Partitioning and pipelining for performance-constrained hardware/software systems *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, pages:419-432, volume: 7, issue: 4, Dec. 1999

[6] K. S. Chatha and R. Vemuri A tool for partitioning and pipelined scheduling of hardware-software systems *The Proceedings of 11th International Symposium on System Synthesis*, pages:145-151, 1998

[7] Jinhwan Jeon and Kiyoung Choi Loop pipelining in hardware-software partitioning *The proceedings of the ASP-DAC'98 Design Automation Conference*, pages: 361-366, 1998

[8] S. Bakshi and D.D.Gajski A spcheduling and pipelining algorithm for hardware/software systems *The Proceedings of the Tenth International Symposium on System Synthsis*, pages:113-118, 1997

[9] Hui Guo HiPer-LoPow: A framework for system level pipelining *PhD thesis*, The University of Queensland, Australia, 1998