Assassination

Killing Pipelines 1st Class Style

Greg Gibeling
Tuesday October 31st, 2006
GDG06 – Assassination

1.0 Introduction

The largest current difficulty impeding progress on the FLEET specification is the current general silence on the setup, use and teardown of pipelined interfaces. The ZOMA move instructions have taken care of most of these difficulties, and in conjunction with well designed SHIPs, including proper token/pipeline interfaces, allow us to pipeline many large designs. The one qualification on this success is that nothing in the current specification, or in any imaginable SHIP specifications can cope with the case where more than a single data item is in flight per-pipeline stage.

Notice that this document refers often to "valves" a concept proposed in AM08.

2.0 Proposal

Many systems exhibit a similar behavior with respect to having an unknown number of data fragments (packets, datagrams, words, etc.) in flight and still requiring synchronous termination as the result of an arbitrary condition.

I use the term arbitrary condition here, to mean that the programmer, not the system architect or SHIP designer, has complete freedom over when a pipeline is torn down. In reality many pipelines will have a fixed count or some simple condition, but I suggest that attempting to build these termination conditions into SHIPs or even valves, may be a serious mistake.

In this design the output valve would keep a count of the number of outstanding tokens or data items at one time. Upon receipt of a "start," which might be encoded in any number of ways but must include the number of data items which can be in flight concurrently, the output valve should produce as many data items as specified by the "start" and increment it's "in flight" counter to that value. Upon receipt of a token, the counter could be decremented. Whenever the count is less than it's initial value another data item may be sent. Notice that so far, this is a fairly simple credit-based flow control scheme, wherein the output valve keeps track of the credits, and there are two standing moves to send data one way, and tokens back.

Tear down could be accomplished by means of the delivery of an "assassination request," note again that this may be encoded in many ways but must include the address of the input valve, to the output valve. Upon receipt of an assassination request (or hit) the output valve would tear down the data half of the pipeline and then wait until it's in-flight counter once again reaches the initial value, meaning all sent data has been acknowledged, and then output a "poison pill" to the input valve's token output, by generating a first class move zero to the input valve's token. This poison would in turn cause a token to be sent from the input valve to an arbitrary destination, specified in the poison instruction and taken from the assassination request, which will indicate that both valves have quiesced, and that all of the standing moves have been killed.

3.0 Conclusion

While this proposal requires the introduction of 1st class moves, and is still fairly rough it does have the nice benefits of integrating with valves from AM08, and providing a very general and powerful mechanism for shutting down pipelined programs upon arbitrary conditions.

Furthermore, one could imagine actually subsuming standing moves into the valves themselves through the use of first class instructions, allowing each valve to keep a register of the address to deliver its output to. This further simplifies standing move instructions from valves, as the switch fabric no longer needs to implement them, and additional means that the assassination request no longer needs to include the address of the input valve. This in turn implies that the global shutdown of pipelines, e.g. for reset purposes, could be simplified to sending assassination requests to all output valves, assuming output valves are designed to ignore a request while they are not operating.

4.0 A Word of Warning

I have voiced this opinion carefully and quietly of late, but with the addition of valves, and other such features I believe that we need to become more mindful of our own reasoning about the switch fabric. Much of our work on pipelined SHIPs and ZOMA moves is predicated on the theory that switch fabric traversals will be costly and thus should be avoided as much as possible.

This theory is in turn predicated on the existence of a very large number of SHIPs per FLEET, which is in turn predicated on our need to use up silicon area. As always alternate designs, such as smaller FLEETs building up to Flottillas will demand different tradeoffs. What is more worrisome is that different assumptions may lead to different tradeoffs, and that our assumptions are in fact self fulfilling.

My primary concern, and a good example of this, is the assumption that switch fabric traversal will be expensive. In order to mitigate this *perceived* cost, we have no working design to provide hard data, we are adding machinery such as ZOMA moves and valves, which will increase the cost of switch fabric traversals. Similarly, by assuming the existence of some complex SHIPs, e.g. memory with a stride, we are ensuring the need for other complex SHIPs to produce a balanced design.

This style of design is forcing us into a CISC mentality, which ultimately leads to the design of e.g. a polynomial multiply SHIP. I suggest that we need to be mindful of this as a serious design problem, and take advantage of the tenets of RISC design, a notably Berkeley product.