AK01: Unified Copies

Amir Kamil
kamil@cs.berkeley.edu

October 4, 2006

1 Introduction

In *AM05*, Adam describes a new, unified `MOVE` instruction with the following variations:

- An *unmove*, that produces only a token at the destination.
- A *simple move*, that moves a single piece of data from the source to the destination.
- A *counting move*, that moves a small number of data items from the source to the destination.
- A *standing move*, that continually moves data items from the source to the destination, until it is canceled.

In this memo, I explore the issues involved in extending these variations to copying moves, an alternative using a special SHIP, and problems with using unified moves with undrainable sources.

2 Unified Copies

2.1 Uncopies

An *uncopy* presumably would copy zero items from the source to destination, perhaps producing a token at the destination. Since this is exactly what an unmove does, I no reason to allow uncopies.

2.2 Simple Copies

Simple copies copy the data item at the source and forward it to the destination. These are already allowed in *IES30*.

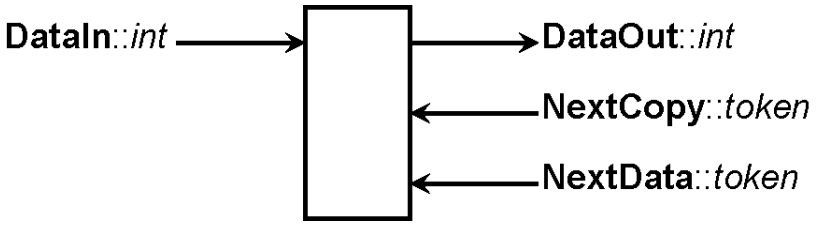

Figure 1: A copying SHIP.

2.3 Counting Copies

A *counting copy* would send n copies of a data item at a source to the destination, where n is some small integer. On the surface, there appears to be no reason to disallow them. However, such a copy could potentially flood the switch fabric. In a counting move, data items cannot move from the source to the switch fabric faster than the source SHIP can produce them. In a counting copy, on the other hand, there is no mechanism to control the rate at which copies are sent to the switch fabric, once the original item is produced at the source. Depending on the maximum size of n, this may or may not be a problem.

2.4 Standing Copies

A *standing copy* would send infinitely many copies of a data item at a source to the destination, until it is canceled by a subsequent `MOVE` or `COPY` instruction with the same source. As with counting copies, there is no restriction on how fast the copies can be made. Unlike a counting copy, there is no upper limit on the number of copies to be made, so performing a standing copy would immediately flood the switch fabric with infinitely many copies of the original data item. This is obviously bad, so standing copies seem doomed.

2.5 Conclusion

Unfortunately, it seems that extending the move variations to `COPY` is a bad idea, since the variations besides the original, simple copy are either worthless or create more problems than they solve. As such, I recommend sticking with only the simple copy already described in *IES30*.

3 A Copying SHIP

There may be cases in which counting or standing copies would actually be very useful. For example, suppose you wanted to add a number x to each element in a vector. It would be useful to set up a standing copy from the source SHIP of x to one of the inputs of an adder SHIP.

Such a result can be obtained using only standing moves if a new copying SHIP is introduced, as in Figure 1. The copying SHIP stores the data item it receives at **DataIn**, and each time it receives a token at **NextCopy**, it produces a copy of that item at **DataOut**. The latter input and output compose a pipeline interface. A token at **NextData** notifies the SHIP to discard its current item and start using the next item at **DataIn**.

In order to use the copying SHIP, a programmer would set up standing moves between the pipeline output of the SHIP and the pipeline input of a destination SHIP. In the adder example, he would move a data item to the **DataIn** input of the copying SHIP and a token to **NextData**. He would then set up a standing move from **DataOut** to an input of the adder (as described in *IES31*), say **DataAin**, and another standing move from the adder's **AckDataA** to the copier's **NextCopy**. Unlike a standing copy, this setup allows copies to be sent only when the adder can receive them, averting a flood in the switch fabric.

4 Unified Moves and Undrainable Sources

Using counting or standing moves with undrainable sources, such as the output of the TokenSource SHIP, can result in the same issues as in §2.3 and §2.4. The rate at which moves occur is only limited by how fast the corresponding SHIPs can replenish the data at their undrainable sources, which is likely to be much faster than the destination SHIPS can process them, and perhaps even faster than the switch fabric can handle them. Either way, gridlock would result in the switch fabric. This indicates that standing moves, and maybe copying also, should be illegal when used with an undrainable source.

Unfortunately, I have not been able to come up with an alternative like the copying SHIP for this case. Perhaps someone else can think of one.