1 Introduction

Several different memories in an SSS node can contain copies of global memory, specifically:

- SRF - indexed via stream instructions.
- Stream cache - physically addressed.
- Scalar cache - virtually addressed.

This document discusses the issue of whether some level of hardware support is needed for stream and/or scalar cache coherence; the SRF is software managed.

2 Scalar Cache Coherence

The purpose of the scalar cache is to reduce the latency of memory operations as seen by the scalar processor. There are several possible choices which can be made with respect to scalar cache coherence in the SSS (i.e.) keeping scalar caches across the system consistent with each other and also with global memory.

1. No hardware support - the software (compiler) must handle the coherence problem, possibly marking segments as non-cacheable if they contain memory addresses used for communicating with other nodes.

2. If the scalar cache were restricted to only caching local addresses, then it could be made coherent with respect to its local memory; whenever a write to that local memory came from a remote node, it would update the cache as well as the memory if the address was cached, and likewise reads from remote nodes for locally cached addresses would be returned with the correct (cached) value.

3. If the scalar cache were restricted to caching only addresses from nodes on its board which shared a bus, a bus snooping protocol could be used to make the intra-board caches coherent.

4. Allowing the scalar cache to cache any global address, but restricting both the number of local addresses which can be cached remotely in each node and also the number of remote nodes which can concurrently cache a single local address, would enable a directory-based scheme to be used. Each node would contain a table specifying a small number of local addresses being remotely cached, and for each, a list of remote nodes caching that address. The cache could also be coherent with respect to local addresses which are not cached remotely but can still be written by a remote node.

    Deciding which nodes can cache which remote addresses could become an allocation problem for the programmer / compiler. Perhaps the program would need to explicitly allocate a shared variable? These shared variables could be implemented either by making them non-cacheable, or by providing an entry in the node’s table of globally cacheable addresses.

Note that the scalar cache is virtually addressed, while the requests from remote nodes are for physical addresses; to handle this, there would need to be a reverse address translation unit between the scalar cache and the memory system.
3 Stream Cache Coherence

The current assumption is that the stream cache will be non-coherent. The intention is that the stream cache will be used for bandwidth amplification rather than inter-node communication or latency reduction; hopefully, programs will be able to use it in read-only mode, and if the write-back mode of operation is used, they will be able to explicitly flush at synchronization points to communicate the newly computed stream data to other nodes via memory.

4 Scalar Processor Touching Stream Data

It is possible that the scalar processor may read or write data which is used in a stream operation. This creates some potential problems if the scalar cache is used.

- A stream store may write data to a memory location which is stored in the scalar cache. If this happens, then the older value in the scalar cache will overwrite the newly-computed value in memory if flushed. To solve this in the case of a locally coherent scalar cache, the written stream data would need to update the scalar cache as well as the memory, just as a write from a remote node would need to update the scalar cache.

- A stream store may write data to a memory location which is stored in the both the scalar cache and the stream cache. This creates an additional problem because the stream cache can sink updates at a much higher rate than either the scalar cache or the memory. To keep the scalar cache locally coherent in this situation, there would need to be a very large FIFO of pending updates which can buffer the stream outputs until the scalar cache has caught up, and if the stream of updates was long enough, some stalls would need to be introduced.

- The scalar cache may contain data which is read from the memory in a stream load; the stream operation will get stale data, unless the memory system knows to get the data from the scalar cache instead.

- If the scalar cache is caching stream data which is also in the stream cache, then the same bandwidth mismatch occurs in the case of a load; the stream cache can supply the data at a much higher rate than the scalar cache, so if an attempt is made to make the stream data come from the scalar cache if it has a more recent copy than the stream cache or memory the bandwidth gain of the stream cache will be lost.

Due to the large bandwidth mismatch between the stream cache and the scalar cache, it would not be feasible to try to keep the scalar and stream caches consistent with each other. The program would be responsible for ensuring that data isn’t cached by both the stream and scalar caches simultaneously, or at least making sure that if it is cached by both then it is used in a read-only fashion.

It may be possible, however, to have the stream data read out of the scalar cache if the stream cache were not involved; the bandwidths match up in this case. Otherwise, the scalar program would simply need to be careful any time it touched stream data, either making it non-cacheable or flushing before it was needed.

5 Scatter-Add

The SSS memory system supports a scatter-add feature. From the stream side, the SRF will contain a stream of partial sums which are to be added to the memory values pointed at by a stream of addresses. There are 2 issues which can arise if the scalar cache contains an address which is the subject of a scatter-add operation.

- If the stream unit issues an add-and-store request for an address in the scalar cache, the value in the cache would need to be read, added to, and stored back in the cache.

- If the scalar processor issues an add-and-store request for an address which is in the cache, the same behaviour must happen.
The choices here are either to support add-and-store in the scalar cache or to force the software (compiler) to take the appropriate precautions to ensure that it always achieves correct behaviour on scatter-adds.

6 Memory Locking

The memory system supports a memory-locking feature to support atomic scalar operations, in which a small table contains a list of addresses which are locked. This feature is used to support scalar synchronisation and atomic scalar fetch-and-op type instructions.

A lock can be either a write-lock, in which only the scalar processor local to the locked address may write it, or an access-lock, in which only the scalar processor local to the locked address may read or write it. If a remote node issues a request which is prevented by the lock, the request will fail.

Note that the stream unit local to the locked address is also prevented from accessing the address while it is locked, just as the stream units on remote nodes are locked out.