### Sample SVM and Merrimac Code

```c
streamInitRAM (&cs_str, SRF, 65536, ...);  
sum.Init ( &sum_1, ... , 128, as_str, bs_str, &cs_str, ... );  
kernelRun ( "Kernel" , &sum_1);  
blockInit ( &cs_blk, global_mem, ... );  
StridedScatterDMA_Init (&StridedScatterDMA_1, &sum_1);  
kernelRun (&StridedScatterDMA_1);  
```

### Mapping Legacy Applications

**Objective and Motivation**
- Stream Programming Paradigm encourages programmers to think slightly differently for better performance.
- Can be mapped reasonably well to General Purpose Processors (like Intel’s Pentium 4 hyper-threaded processor).
- Creates an evolutionary path to using stream processors.
- Enables more efficient compilation.
- Provides instant gratification of the benefits of stream programming.

**Mapping Issues**
- SRF is simulated by large L2 cache (as shown in figure).
- Contiguous segment of global memory mapped to L2 cache.
- Intermediate streams in SRF not written back to memory.
- Streams blocked efficiently based on SRF size.
- Streaming stages (gather, operate, and scatter) are mapped to hardware hyper-threads.
- Overlap memory and computation.
- Map control, computation, and memory tasks on to hyper-threads.

**Compiler Implications**
- Parallelism is explicit in the program enabling more efficient compilation.
- Several compiler analyses (like software prefetching) are simpler to perform.
- Software pipelining is performed at a larger granularity (stage-level rather than instruction-level).
- Cache utilization is improved.

**Mapping Legacy Applications**
- SVM code is generated by HLC and transformed to Merrimac code using LLC.
- SVM code has three components:
  1. Scalar code
  2. Stream code
  3. Kernel code
- LLC Compilation tool: cTool, a parsing library for source-to-source transformation.
- Stream code is used for analyzing dependences due to streams and for subsequent resource allocation and stream instructions generation.
- Kernel code is first transformed to kernelC (language natively developed at Stanford to program SIMD clusters) and then compiled to micro-code using Merrimac Kernel Scheduler.
- Merrimac code generated by LLC is run on the Merrimac Simulator.

**High-level Compiler**
- Key technologies + Robust infrastructure + Modular interfaces

**Architectures**
- Streaming on Conventional Processors
- Streaming on Conventional Architectures

**Status and Future Work**
- Implemented first version of LLC
- Completed and implemented simple SVM code and test cases.
- Features currently being added to R-Stream 1.9.
- Released for use by others.
- SIMD scheduling and code generation.
- SPMD support for multiple nodes.
- Developed C-subset front-end.
- Implemented standard affine analyses.
- Next step: To target affine mapping to Merrimac.

**Status:** October 2004