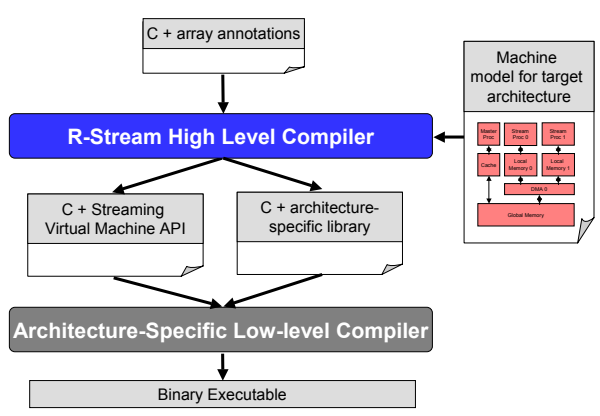
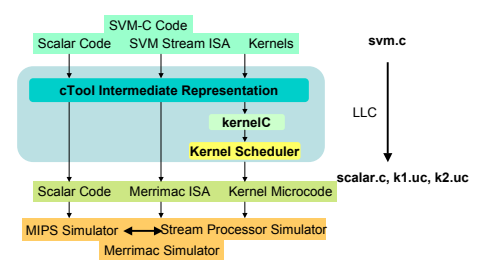


## Overall Compiler Framework



## Low-level Compiler



- SVM code generated by HLC is transformed to Merrimac code using LLC
- SVM code has three component:
  - scalar code
  - stream code
  - kernel code
- LLC Compilation tool: cTool, a parsing library for source-to-source transformation
- Stream code is used for analyzing dependencies 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.

## Mapping Legacy Applications

Research is underway to utilize existing compiler analyses to map legacy programs to Merrimac. Consider example of dense matrix multiplication:

```

    for i = 1..m :
      for j = 1..n :
        C[i,j] = 0;
        for k = 1..t :
          C[i,j] += A[i,k] * B[k,j];
    
```

In conventional machine, matrices are blocked to fit cache.  
In Merrimac, blocking is to fit SRF.  
The same blocking algorithms can be used for both.

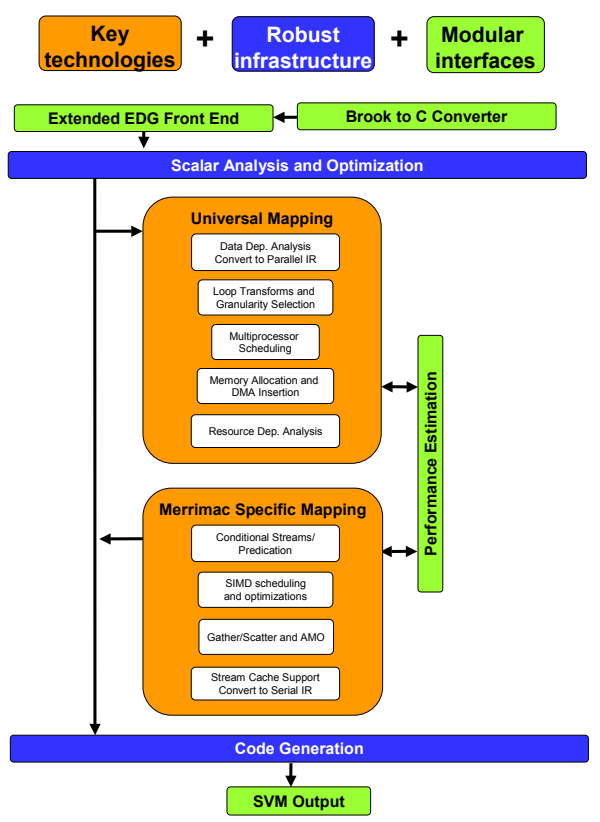
In Merrimac, blocking is also affected by SIMD nature of the clusters and lanes in SRF.

Blocks are distributed across lanes, and the kernel must use the inter-cluster switch to communicate data to the cluster which needs it.

In Merrimac, high SRF bandwidth is obtained by sequential accesses.

The matrix blocks are laid out as 1D streams in the SRF, and the ordering of the data in each 1D stream is matched to the operation order in the kernel which consumes the data.

## High-level Compiler



## Output of Low-level Compiler

```

    scalar.c
    SET_SDR sdr1, length1,
    block1
    x = z;
    SET_MAR mar1, base1, ...
    STRREAD mar1, sdr1, idx
    if (x == y) ...
    LD_UCODE sdr, line
    RUN_KRNL PC, str0, str1
    ...

    Scoreboard
    SET_SDR 00..10 00..01
    SET_MAR 01..00 01..11
    GATHER 00..00 01..00
    LD_UCODE 00..10 00..11
    RUN_KRNL 00..11 10..11
    ...

    k1.uc, k2.uc, ...
    mul r1, r2, r3 r2
    add r2, r1, r4
    jr r5
    ...
    
```

- LLC generates Merrimac code and debug information for every stream instruction to be used by the Merrimac simulator
- Scalar and Stream code are emitted in a C (scalar.c) file
  - asm directives conforming to the scalar-stream interface are embedded in the scalar code to dispatch stream instructions at run-time
  - Dependencies between stream instructions are explicitly encoded in the scalar.c file
  - The simulator stores these dependencies in the form of a scoreboard (stream instruction, start dependency, end dependency)
- LLC also generates separate micro-code files for each kernel

## Sample SVM and Merrimac Code

```

    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, ((char*)cs_addr), 128, ...);
    stridedScatterInit (&StridedScatterDMA_1, SSS_DMA, cs_str, &cs_blk, 128, ...);
    kernelAddDependence (&StridedScatterDMA_1, &sum_1);
    kernelRun (&StridedScatterDMA_1);

    ADD_DEP_S(8, 0UL, 24);
    SETSDR(5, 128, 1024, 25);
    ...
    //ASM ADD_DEP_C 9 0 484 //2 5 6 7 8
    ADD_DEP_C(9, 484, 26);
    RUN_KRNL(0, 0, 0, 3, 4, 5, 0, 0, 0, 0, 0, 0, 27);
    ...
    ADD_DEP_S(10, 0UL, 28);
    SETSDR(6, 128, 1024, 29);
    block_addr = ((int)((char*)(cs_addr)) + 0);
    SETMAR(2, block_addr, 1, 4, 0, 31);
    //ASM ADD_DEP_C 12 0 3616 //5 9 10 11
    ADD_DEP_C(12, 3616, 32);
    STRWRITE(6, 2, 33);
    
```

## Streaming on Conventional Processors

**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 emulated 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

## Status and Future Work

HLC	LLC
<ul style="list-style-type: none"> <li>R-Stream 1.9 released (Sep 2004)</li> <li>Completely rewritten version of R-Stream 1.0</li> <li>Compiles simple programs and produces optimized SVM code</li> </ul>	<ul style="list-style-type: none"> <li>Implemented first version of LLC</li> <li>Natively implemented</li> <li>Compiles simple SVM code and produces Merrimac code</li> </ul>
<ul style="list-style-type: none"> <li>Features currently being added to R-Stream 1.9                             <ul style="list-style-type: none"> <li>Indexed gathers and scatters</li> <li>SIMD scheduling and code generation</li> <li>SPMD support for multiple nodes</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Issues currently being addressed                             <ul style="list-style-type: none"> <li>Conforming to new SVM specification</li> <li>Adapting to updated Merrimac specification</li> </ul> </li> </ul>
<p><b>Mapping Legacy Applications</b></p> <ul style="list-style-type: none"> <li>Developed C-subset front-end</li> <li>Implemented standard affine analyses</li> <li>Next step: To target affine mapping to Merrimac</li> </ul>	<p><b>Streaming on Conventional Architectures</b></p> <ul style="list-style-type: none"> <li>Mapped stream programming model on Pentium 4</li> <li>Evaluated the effectiveness using micro-benchmarks</li> <li>Next step: Evaluate Merrimac Applications on Pentium 4</li> </ul>