

Sparse Matrix Vector Multiplication on the SRC MAPstation

Sreesa Akella

Computer Science and Engineering
University of South Carolina

**Melissa C. Smith, Richard T. Mills,
Sadaf R. Alam, Richard F. Barrett,
Jeffrey S. Vetter**

Oak Ridge National Laboratory

Outline

- Introduction
- Software implementation
- SRC implementations
- Analysis
- Other architectures
- Current Status
- Future work

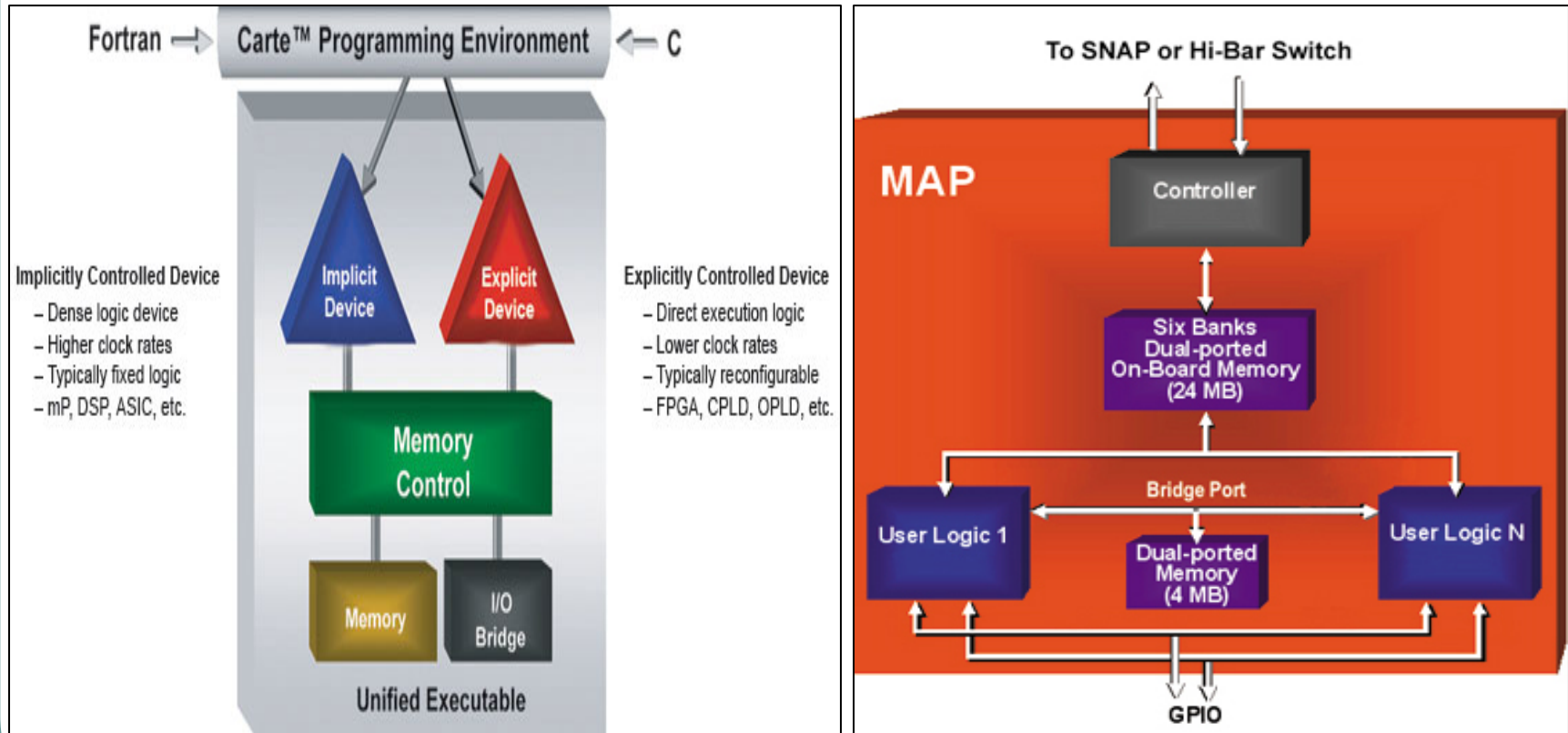
FP Sparse Matrix Vector Multiplication

- Sparse Matrix Vector Product
 - $Ax = y$
 - A: Matrix with very few non-zero elements
 - x: Vector
 - y: Result vector
- Used in iterative solvers for linear systems
- Not efficient on general purpose microprocessor systems
 - High cache miss rate due to poor data locality
 - Low utilization of floating point units due to high ratio of load/store to floating point operations

SpMatVec on FPGAs

- FPGAs
 - Effectively implement floating point applications
 - High density can be utilized to implement multiple floating point units
- SRC-6 MAPstation architecture
 - Local distributed memory banks (six 4 MB banks)
 - High density FPGAs (Virtex-II 6000)
 - High speed host- μ p to FPGA communication (peak 1400 MB/s)

SRC-6 MAPstation



Sparse Matrix Storage Format

- CSR – Compressed Sparse Row
- Example:
 - Non-zero elements: NZ = [1, 2, 5, 4, 5, 3, 2, 7, 8, 1]
 - Column Indices: CI =[1, 3, 2, 5, 1, 4, 2, 4, 3, 5]
 - Row Pointers: PT = [1, 3, 5, 7, 9, 11]
 - Get the Row lengths (RL) from PT
- Other formats:
 - ELL, JAD, CSRPerm, CCS

1	0	2	0	0
0	5	0	0	4
5	0	0	3	0
0	2	0	7	0
0	0	8	0	1

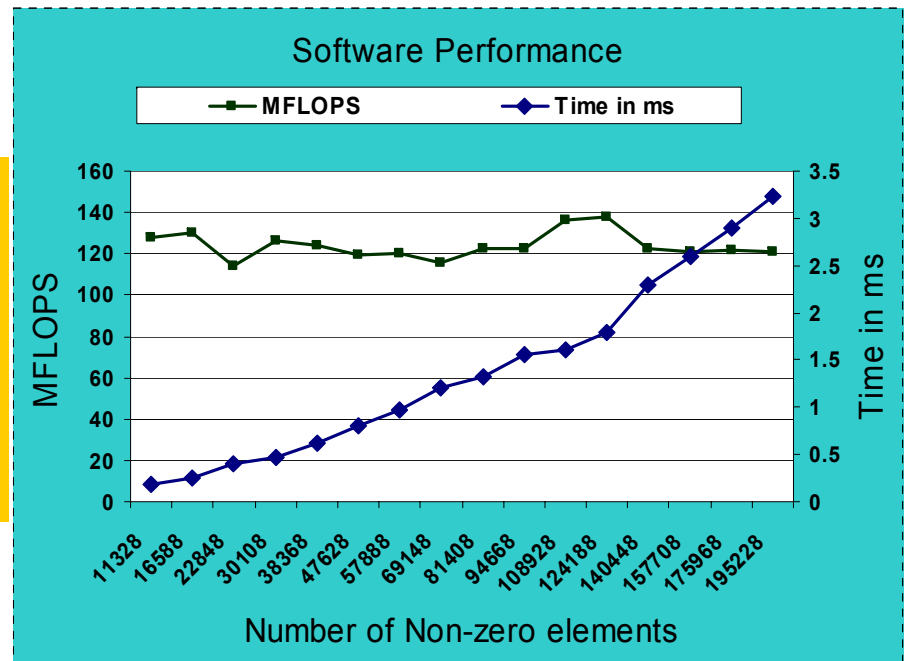
Software implementation - SPARSKIT

- Fortran 77 code rewritten in C
- Uses the nested loop structure
- On SRC-6
 - Dual Intel Xeon at 2.8 GHz

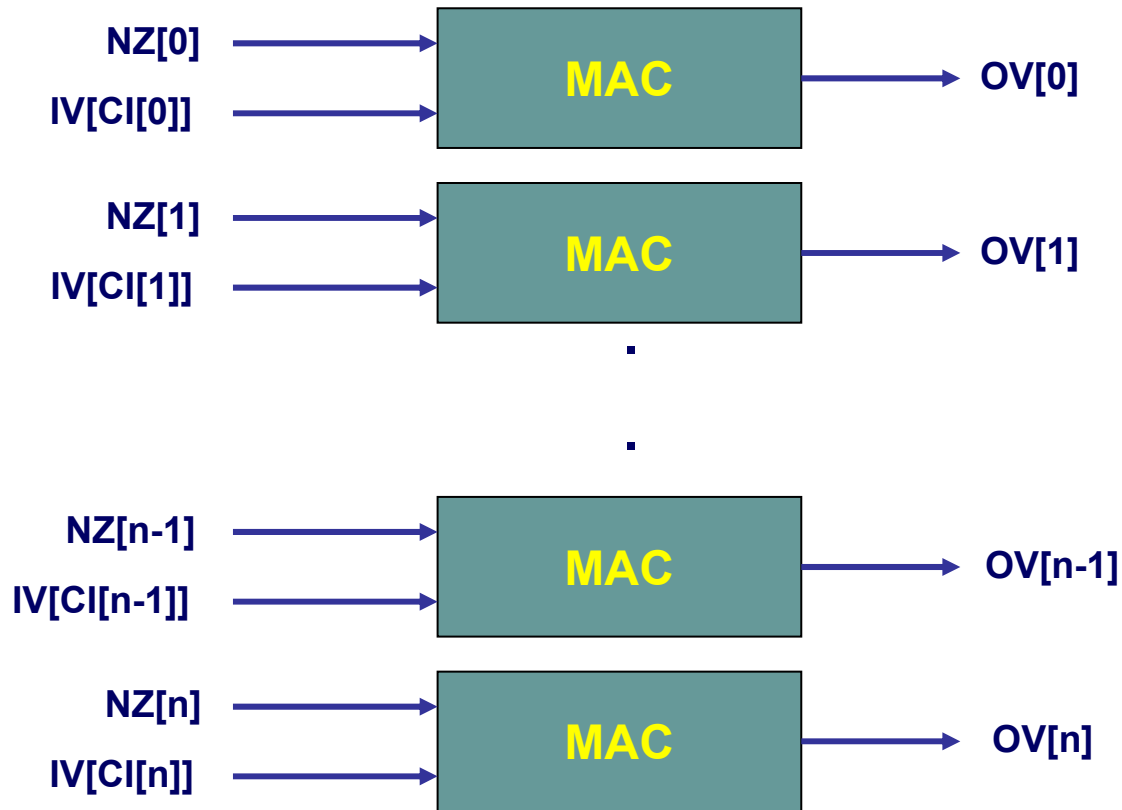
```

for(i=0;i<nrow;i++){
  // compute the inner product of row
  // with vector x
  t = 0.0;
  for(k=PT[i];k<PT[i+1];k++)
    t = t + NZ[k]*IV[CI[k]];
  //store result in y(i)
  OV[i] = t;
}

```



Basic Architecture of the kernel



NZ – Non-zero element vector, CI – Column indices vector,
IV- Input vector, OV- Output vector

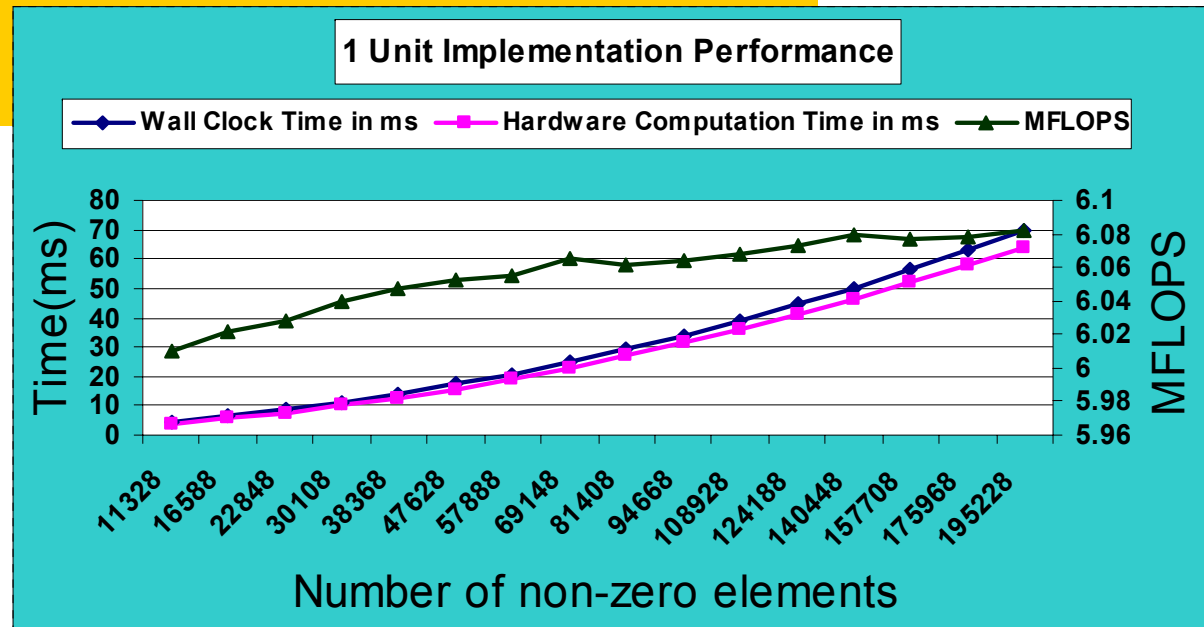
SRC-6 Implementation: 1-Unit

- Written in C Map
- The NZ, CI, RL, IV, and OV data are stored on the OBM banks

```

for(i=0;i<nrow;i++){// OV element index
  l = CL[i];
  for(j=k;j<k+l;j++){
    fp_mac_64 (NZ[j], IV[CI[j]], j==k+l-1, 1, j==k, &OV[i], &err);
  }
  k = k + l;
}

```

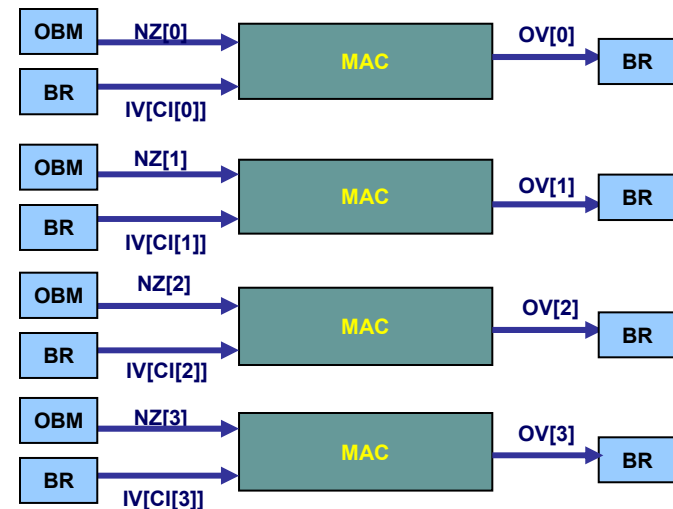
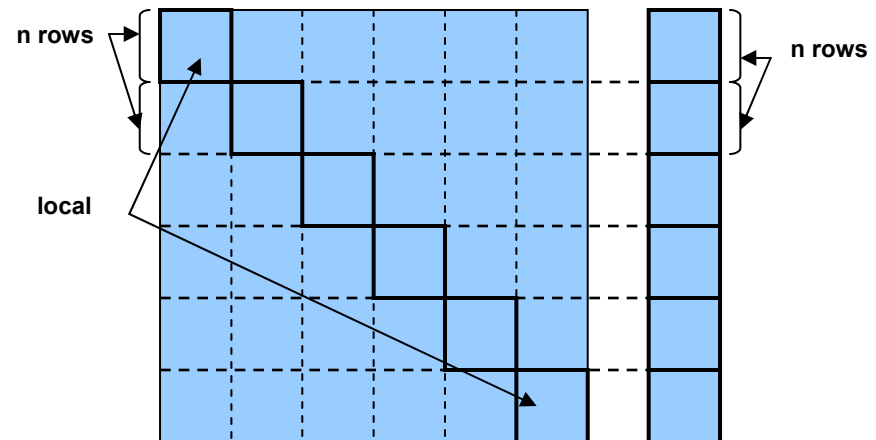


SRC-6 Implementation: 1-Unit Analysis

- Random access to memory bank for IV elements
- Each read to the OBM has a 4 cycle latency
- Random reads to the OBM lead to lot of latency cycles
- BRAM read take only 1 cycle
- Moving IV to BRAM would reduce the latency
- Multiple BRAMs for storing IV would facilitate parallelism

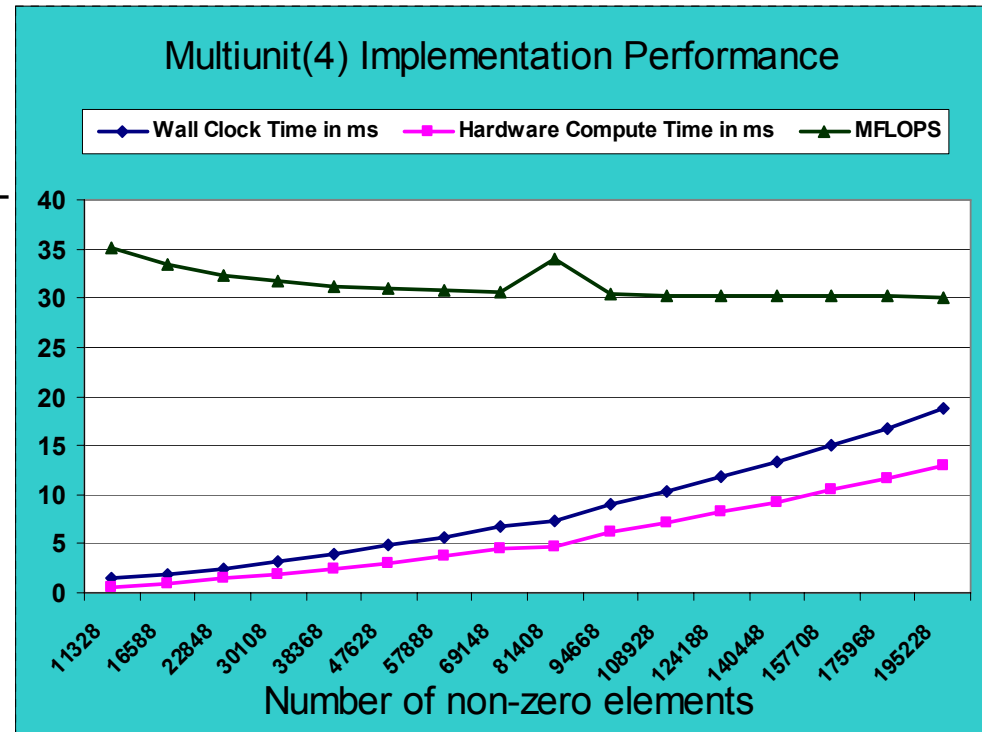
Multi-unit (4) Local/Non-Local (LNL) blocks implementation

- Non-duplication of IV
 - Local – FPGA, Non-local & Sub-product sum - CPU
 - Can be run for matrices of size 40000x40000
 - CI and RL share a single OBM bank
 - Each shares half – 262144 words
 - Limits the number of non zeroes that can be transferred at one time
 - Can look at local on FPGA1 and non-local on FPGA2, sum on CPU



Variations/Improvements & Performance

- Multiplier-add instead of MACs
 - **Cycles improvement: 47%** for 3600x3600 matrix
- Different 'for loops' for each Mul-add unit
 - Each Mul-Add runs for a different length of iterations
 - Each loop in a different parallel section
 - Outer loop moved into each parallel section
 - **Cycles improvement: 20%** for 3600x3600 matrix



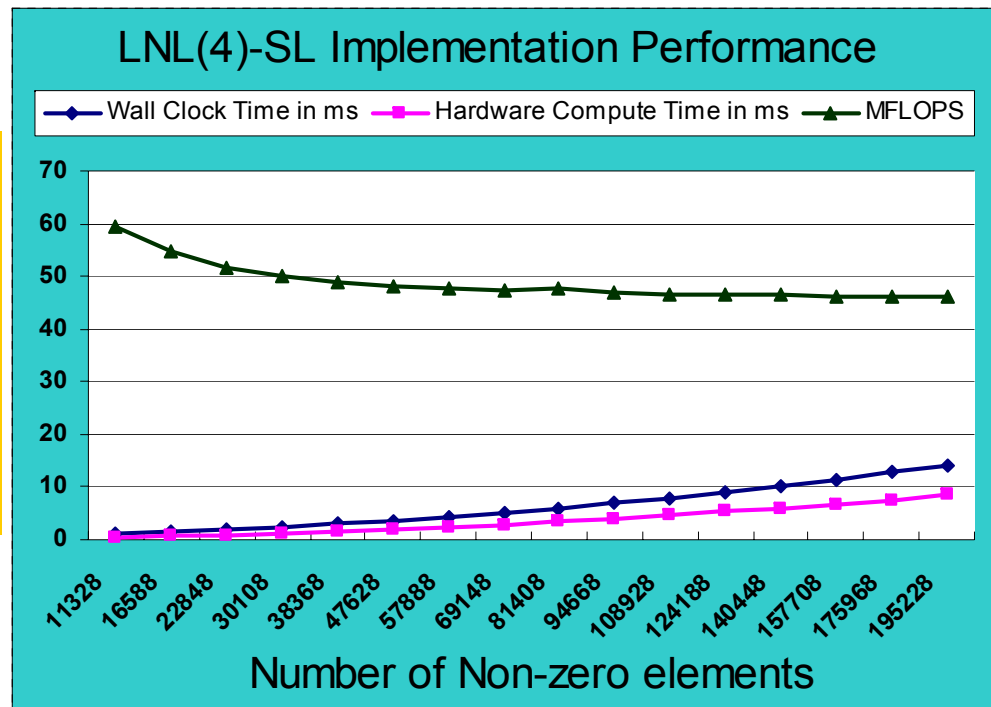
LNL Single Loop (SL) design

- The two loop structure is made into a single loop structure
- 1-unit SL is **3 times faster** than 1-unit nested loop design
- 4-unit LNL-SL is **7.8 times faster** than 1-unit design

```

for (i=0;i<bank2_length;i++){// row index
  temp += BL[i+b1+n3]*IVec2[ci_bram2[i]-n2];
  if (j==l2-1){
    OVec2[k] = temp;
    temp = 0;
    j = 0;
    k++;
    l2 = rl_bram2[k];
  }
  else
    j++;
}

```



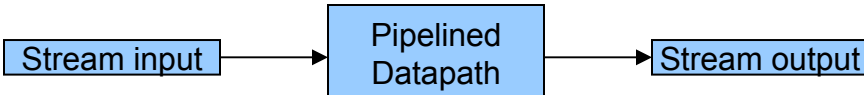
NIST Sparse BLAS toolkit implementations

- Implement simple Sparse Matrix-Vector multiply written in C code
 - Sun Sparc 20
 - IBM RS6000
 - <http://math.nist.gov/spblas/>
- Preliminary implementations yield a performance of:
 - 17 MFLOPs on Sparc 20
 - 27 MFLOPs on IBM RS6000
- Our best implementation has a peak performance of 59 MFLOPs for small datasets and 46 MFLOPs for larger datasets.

Analysis - Parallelism

- Parallelism
 - Obtain 7.8 times improvement over 1-unit by having 4 parallel units
 - More parallel units can be implemented using the second FPGA
 - Four FP units take about 89% of the Virtex-II 6000 chip including the memory interface
 - Limitations:
 - Few memory banks – 4 for NZ, 1 for CI&RL, 1 for IV&OV
 - Forced to use on-chip BRAMs
 - Low On-chip memory capacity – limits the amount of data transferred at a time
 - Time to transfer from OBM banks to BRAMs
 - Multi-FPGA implementations would improve performance
 - Need more distributed memory units
- Memory Bandwidth
 - 4.8 GB/s to a single FPGA (all six banks dedicated to one FPGA)
 - 2.4 GB/s each to two FPGAs (3 banks dedicated to each FPGA)
 - More bandwidth necessary to implement parallel units without using BRAMs
 - About 9.6 GB/s for obtaining the 4 values of NZ, CI, & IV in parallel

Larger datasets and other architectures

- Streaming of the data arrays
 - Output Stream is possible in Carte 2.0
- 
- ```
graph LR; A[Stream input] --> B[Pipelined Datapath]; B --> C[Stream output]
```
- Parallel DMA and Computation sections
    - Would overlap/hide the computation sections over the DMAs
  - Multi-FPGA implementations
    - partition large dataset into multiple smaller ones
  - Other architectures and storage formats:
    - Zhou, et. al. (FPGA 2005) look at tree based structure
      - k multipliers, k-1 adders, tree structure
      - CSRPerm – variant of CSR
    - DeHon et. al.'s (FPGA 2005) architecture.
      - Multi-FPGA implementation
      - Simple mul+add datapath per PE
      - Bidirectional ring for communication



# Current Status & Lessons Learned

- Current Status:
  - 4-unit, single loop implementation performance is still about **2-2.55 times slower** than software
  - More parallel units
    - Need multiple FPGAs and distributed memory banks
- Lessons Learned:
  - SRC-6 provides moderate performance for SpMatVec multiplication operation
  - Need better data transfer mechanism
    - **Data transfer time is more than the total software operation time**
    - **Hide data transfer behind computation**
  - FPGAs can perform floating-point operations fast if:
    - **Feed data at high bandwidth**
    - Have **many parallel floating point units**
    - Need **more memory units** for parallelism

# Future Work

- Parallel DMA/computation sections
- Combining transfer of NZ, CI and IV, RL to achieve maximum DMA transfer bandwidth
- Multi-FPGA Implementations to extract more parallelism
- Other Architectures and different storage formats
- Cray XD1 architecture
  - 2 Dual Opterons & 1 Virtex-II Pro FPGA per blade
  - Four 36-bit word QDR-II SRAMs per FPGA
  - 3.2 GB/s data transfer between host and FPGAs
  - Six FPGAs per chassis suitable for Multi-FPGA implementation
- SRC-7 coming out in 1<sup>st</sup> quarter of next year
  - Two 30 MGates user logic chips/One 30 MGates and One Field programmable FP device
    - DP - 30 GFLOPs
    - SP - 60 GFLOPs
  - Ten 64-bit word SRAMs on board
  - Two I/P and two O/P streaming capability
  - 14.4 GB/s data transfer between host and FPGAs