

# StreamSPAS on Merrimac

Jung Ho Ahn

# StreamSPAS

- Sparse Matrix-Vector product implementations
- Contains four common ways
  - Compressed sparse row (CSR) storage
    - utilizing elementary row operations together with indexed memory gathers
  - Compressed sparse column (CSC) storage
    - utilizing elementary column operations together with scatter-adds
  - Hypergraph edge storage (HES) for pattern symmetric matrices
    - utilizing both gather and scatter-adds but with fixed record sizes
  - Element-by-element storage (EBES) for FEM matrices
    - utilizing both gather and scatter-adds with fixed record sizes
- Originally implemented in C++

# StreamSPAS - CSC

$$\begin{bmatrix} a & b & c & 0 \\ d & e & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & 0 & 0 & g \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{pmatrix} a, d, f & \alpha \\ & b, e & \beta \\ & & c & \gamma \\ & & & g & \delta \end{pmatrix} \begin{pmatrix} 3 \\ 5 \\ 6 \\ 7 \end{pmatrix}$$

- $sX$  (input vector) =  $( \alpha \ \beta \ \gamma \ \delta )$
- $sAsp$  (matrix data) =  $( a \ d \ f, \ b \ e, \ c, \ g )$
- $sIsp$  (column sync) =  $( 0 \ 3 \ 5 \ 6 \ 7 )$
- $sOut$  (output) =  $( a\alpha \ d\alpha \ f\alpha, \ b\beta \ e\beta, \ c\gamma, \ g\delta )$
- $sY$  (output vector) =  $( a\alpha+b\beta+c\gamma \ d\alpha+e\beta \ f\alpha \ g\delta )$

# StreamSPAS - CSR

$$\begin{bmatrix} a & b & c & 0 \\ d & e & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & 0 & 0 & g \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{pmatrix} a,b,c & 0,1,2 \\ d,e & 0,1 \\ f & 0 \\ g & 3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

- sXIndex (input vector) = (  $\alpha \ \beta \ \gamma, \alpha \ \beta, \alpha, \delta$  )
- sAsp (matrix data) = (  $a \ b \ c, d \ e, f, g$  )
- sIsp (column sync) = (  $0 \ 3 \ 5 \ 6 \ 7$  )
- sY (output vector) = (  $a\alpha+b\beta+c\gamma \ d\alpha+e\beta \ f\alpha \ g\delta$  )

# StreamSPAS - HES

$$\begin{bmatrix} a & 0 & e & 0 \\ 0 & b & 0 & g \\ f & 0 & c & 0 \\ 0 & h & 0 & d \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{matrix} a, b, c, d \\ (e, f), (g, h) \\ \alpha, \beta, \gamma, \delta \end{matrix}$$

- sX (input vector) = (  $\alpha$  ,  $\beta$  ,  $\gamma$  ,  $\delta$  )
- sAspDiag (diagonal matrix data) = ( a , b , c , d )
- sAsp (matrix data) = ( (e, f) (g, h) )
- sImat (matrix index) = ( (0, 2) (1, 3) )

# StreamSPAS - EBES

$$\begin{bmatrix} a & b & 0 & 0 \\ c & 0 & 0 & 0 \\ 0 & 0 & d & e \\ 0 & 0 & 0 & f \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{matrix} (a,b,c,0), (d,e,0,f) \\ (0,0), (2,2) \\ \alpha, \beta, \gamma, \delta \end{matrix}$$

- sX (input vector) = (  $\alpha$  ,  $\beta$  ,  $\gamma$  ,  $\delta$  )
- sAmat (matrix data) = ( (a, b, c, 0) (d, e, f, 0) )
- sImat (matrix index) = ( (0, 1) (2, 3) )

# Direct Conversion

- Didn't touch data structures for vectors & matrices
- Use predication to deal with variable length rows and columns
- Didn't do software pipelining explicitly

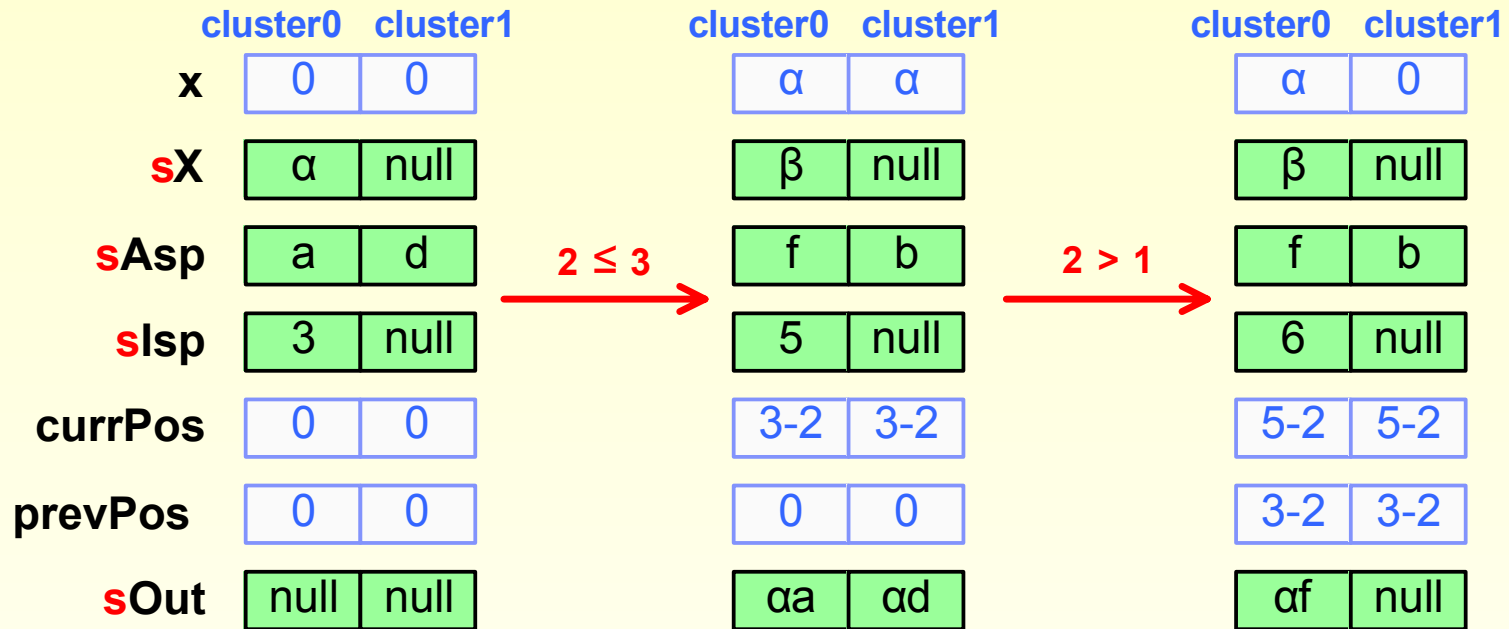
# New Implementation

- Modified data structures for better parallelization
  - Zero padding to make sizes equal for variable length rows and columns
  - Made a stream to store the maximum length of 16 columns/rows of a matrix
- Explicit software pipelining
- Used Stream cache for vectors

# CSC – Direct Conversion

$$\begin{bmatrix} a & b & c & 0 \\ d & e & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & 0 & 0 & g \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{pmatrix} a, d, f & \alpha \\ b, e & \beta \\ c & \gamma \\ g & \delta \end{pmatrix} \begin{pmatrix} 3 \\ 5 \\ 6 \\ 7 \end{pmatrix}$$

- $sX = ( \alpha \ \beta \ \gamma \ \delta )$
- $sAsp = ( a \ d \ f, \ b \ e, \ c, \ g )$
- $sJsp = ( 0 \ 1 \ 2, \ 0 \ 1, \ 0, \ 3 )$
- $sIsp = ( 0 \ 3 \ 5 \ 6 \ 7 )$
- $sOut = ( a\alpha \ d\alpha \ f\alpha, \ b\beta \ e\beta, \ c\gamma, \ g\delta )$
- $sY = ( a\alpha + b\beta + c\gamma \ d\alpha + e\beta \ f\alpha \ g\delta )$

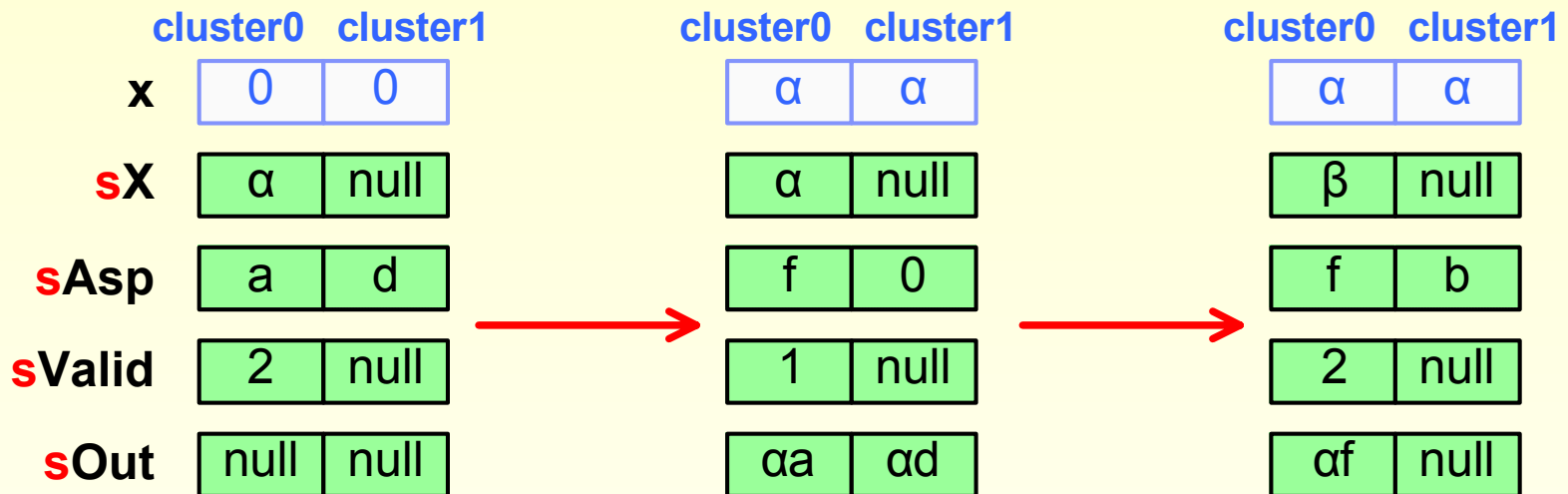


scatter-add **sOut** with **sJsp**

# CSC – New Implementation

$$\begin{bmatrix} a & b & c & 0 \\ d & e & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & 0 & 0 & g \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{pmatrix} a,d,f & \alpha \\ b,e & \beta \\ c & \gamma \\ g & \delta \end{pmatrix} \begin{pmatrix} 3 \\ 5 \\ 6 \\ 7 \end{pmatrix}$$

- $sX = ( \alpha \ \beta \ \gamma \ \delta )$
- $sAsp = ( a \ d \ f, \ b \ e, \ c, \ g )$
- $sJsp = ( 0 \ 1 \ 2, \ 0 \ 1, \ 0, \ 3 )$
- $sIsp = ( 0 \ 3 \ 5 \ 6 \ 7 )$
- $sOut = ( a\alpha \ d\alpha \ f\alpha, \ b\beta \ e\beta, \ c\gamma, \ g\delta )$
- $sY = ( a\alpha+b\beta+c\gamma \ d\alpha+e\beta \ f\alpha \ g\delta )$

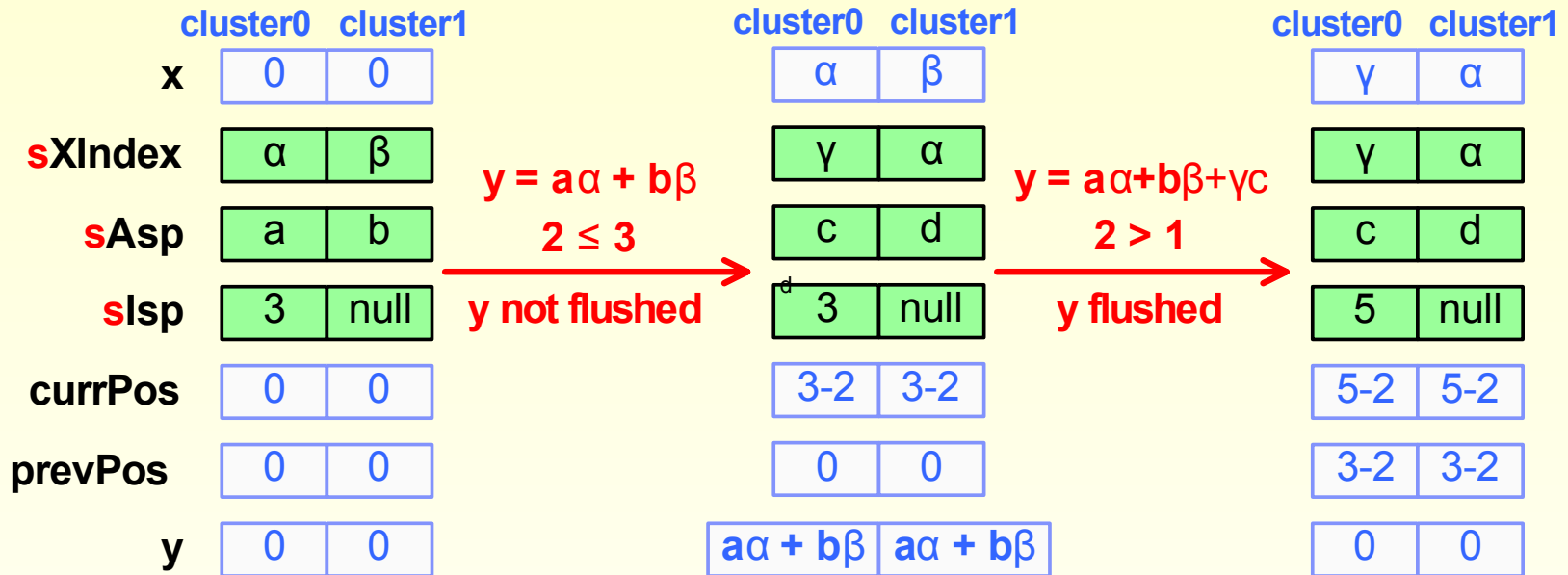


scatter-add **sOut** with **sJsp**

# CSR – Direct Conversion

$$\begin{bmatrix} a & b & c & 0 \\ d & e & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & 0 & 0 & g \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{pmatrix} a,b,c & 0,1,2 \\ d,e & 0,1 \\ f & 0 \\ g & 3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

- $sXIndex = (\alpha \beta \gamma, \alpha \beta, \alpha, \delta)$
- $sAsp = (a \ b \ c, d \ e, f, g)$
- $sJsp = (0 \ 1 \ 2, 0 \ 1, 0, 3)$
- $sIsp = (0 \ 3 \ 5 \ 6 \ 7)$
- $sY = (a\alpha+b\beta+c\gamma \ d\alpha+e\beta \ f\alpha \ g\delta)$



# CSR – New Implementation

$$\begin{bmatrix} a & b & c & 0 \\ d & e & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & 0 & 0 & g \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{pmatrix} a,b,c & 0,1,2 \\ d,e & 0,1 \\ f & 0 \\ g & 3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

- $sXIndex = ( \alpha \beta \gamma, \alpha \beta, \alpha, \delta )$
- $sAsp = ( a \ b \ c, d \ e, f, g )$
- $sJsp = ( 0 \ 1 \ 2, 0 \ 1, 0, 3 )$
- $sIsp = ( 0 \ 3 \ 5 \ 6 \ 7 )$
- $sY = ( a\alpha + b\beta + c\gamma \ d\alpha + e\beta \ f\alpha \ g\delta )$

**sXIndex**

cluster0    cluster1

$\alpha$	$\alpha$
$\beta$	$\beta$
$\gamma$	0

**sAsp**

a	d
b	e
c	0

**y**  $a\alpha + b\beta + \gamma c$   $d\alpha + e\beta$

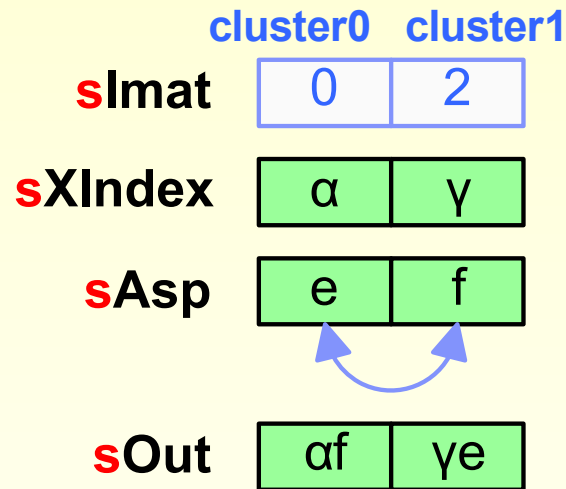
# HES - Implementation

$$\begin{bmatrix} a & 0 & e & 0 \\ 0 & b & 0 & g \\ f & 0 & c & 0 \\ 0 & h & 0 & d \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{matrix} a, b, c, d \\ (e, f), (g, h) \\ \alpha, \beta, \gamma, \delta \end{matrix}$$

- $sX = ( \alpha , \beta , \gamma , \delta )$
- $sAspDiag = ( a , b , c , d )$
- $sAsp = ( (e, f) (g, h) )$
- $sImat = ( (0, 2) (1, 3) )$

Step 1 :  $y = (a\alpha, b\beta, c\gamma, d\delta)$

Step 2 :

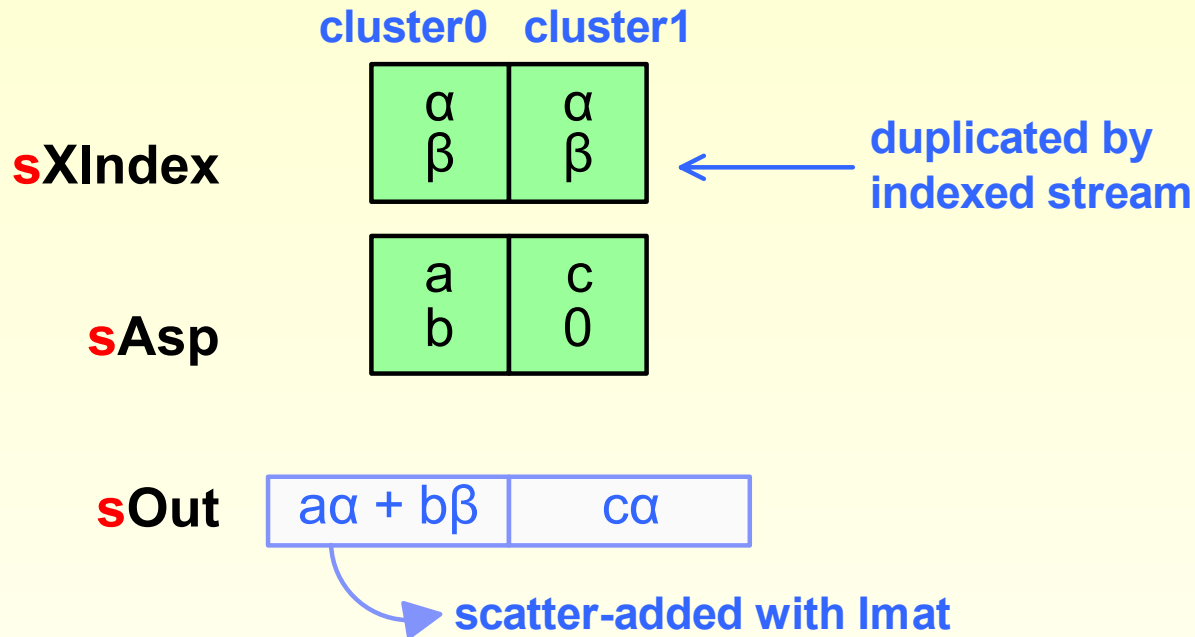


$y[0] += f\alpha, y[2] += e\gamma$  (scatter-add)

# EBES – Direct Conversion

$$\begin{bmatrix} a & b & 0 & 0 \\ c & 0 & 0 & 0 \\ 0 & 0 & d & e \\ 0 & 0 & 0 & f \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \rightarrow \begin{matrix} (a,b,c,0), (d,e,0,f) \\ (0,0), (2,2) \\ \alpha, \beta, \gamma, \delta \end{matrix}$$

- $sX = ( \alpha , \beta , \gamma , \delta )$
- $sAsp = ( (a, b, c, 0) (d, e, 0, f) )$
- $sImat = ( (0, 1) (2, 3) )$



# EBES – New Implementation

$$\begin{bmatrix} a & b & 0 & 0 \\ c & 0 & 0 & 0 \\ 0 & 0 & d & e \\ 0 & 0 & 0 & f \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix}$$

$$(a, b, c, 0), (d, e, 0, f)$$

$$(0, 0), (2, 2)$$

$$\alpha, \beta, \gamma, \delta$$

- $sX = (\alpha, \beta, \gamma, \delta)$

- $sAsp = ((a, b, c, 0) (d, e, 0, f))$

- $sImat = ((0, 1) (2, 3))$

**sXIndex**

cluster0 cluster1

$\alpha$	$\gamma$
$\beta$	$\delta$

**sAsp**

$a$	$d$
$b$	$e$
$c$	$0$
$0$	$f$

**sOut**

$y[0] = a\alpha + b\beta$	$y[2] = d\gamma + e\delta$
$y[1] += c\alpha$	$y[3] += f\delta$

# StreamSPAS – Data set

Number of tetrahedra	Polynomial	Number of rows	Number of non zeros	Average nnz/row
1916	1	471	5846	12.41
1916	2	3158	80372	25.45
1916	3	9978	441676	44.26
3787	1	846	11050	13.06
3787	2	5948	155810	26.20
3787	3	19094	863098	45.20

# Merrimac – simulation spec.

## ■ Computation unit

- 16 clusters
- 4 ALUs per cluster (multiplier or adder)
- Operating at 1GHz – 64 GFLOPS

## ■ SRF

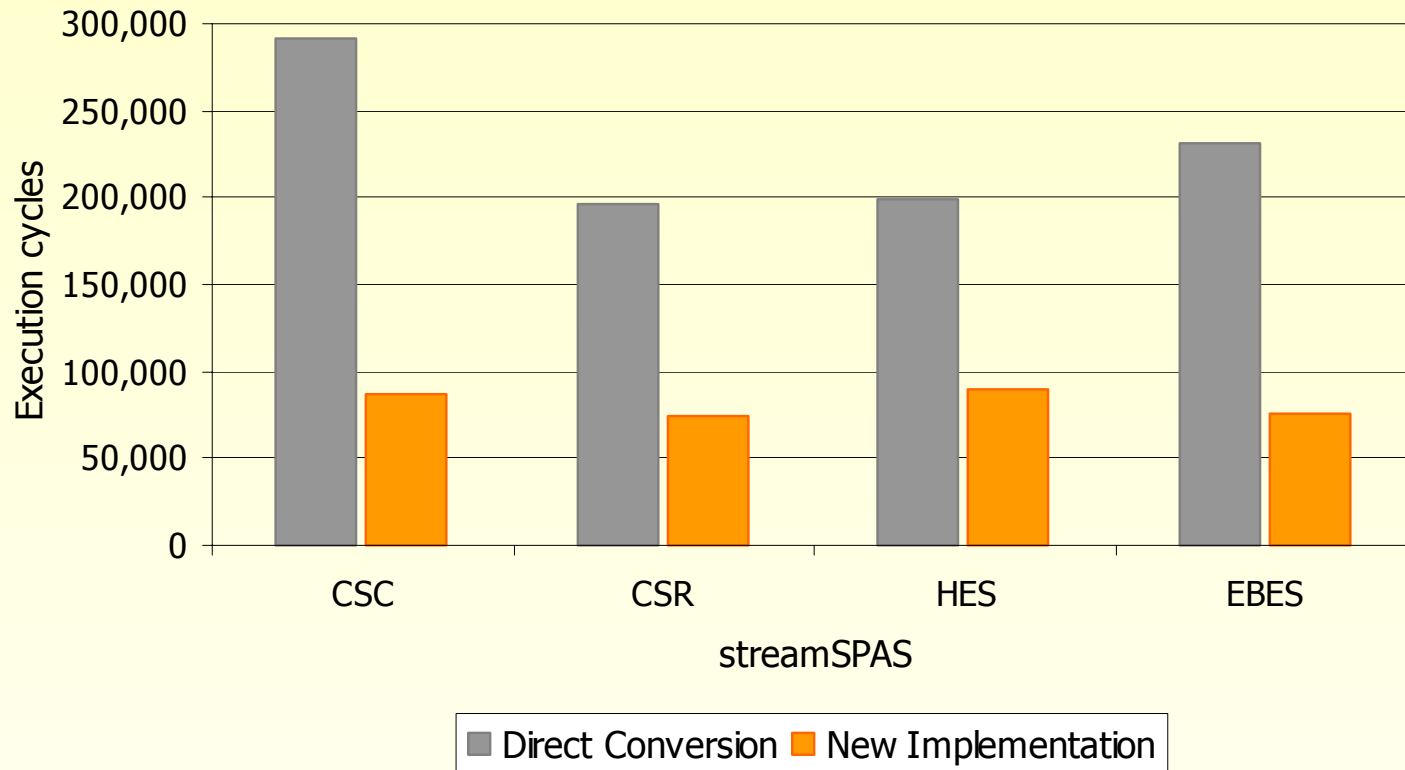
- 64K words
- No indexable SRF support

## ■ Memory

- 64K words stream cache
- 8 stream cache banks
- 16 DRAM interfaces
- 4.6 words per cycle DRAM bandwidth (38.4 GB/s)

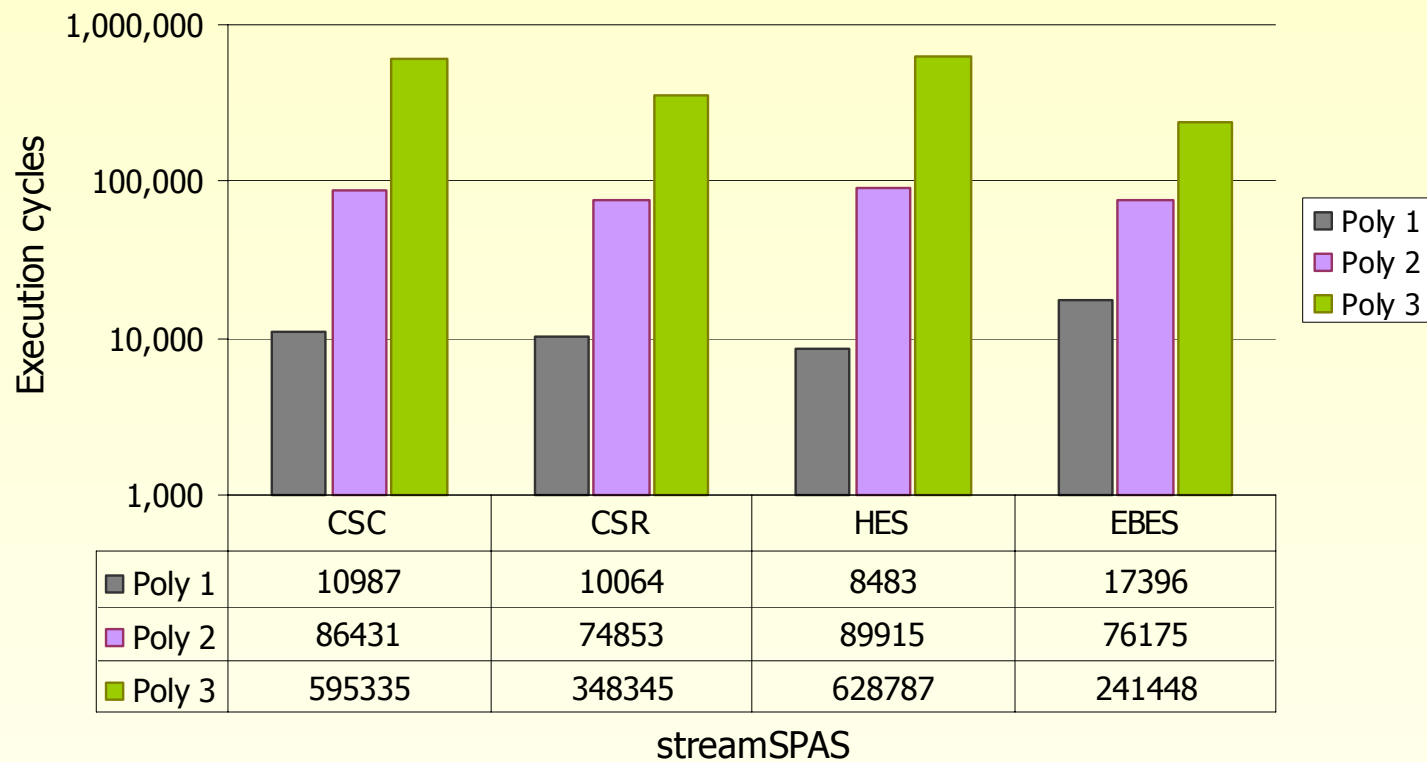
# Execution time comparison

## Run time comparison (1918 - poly order 2)



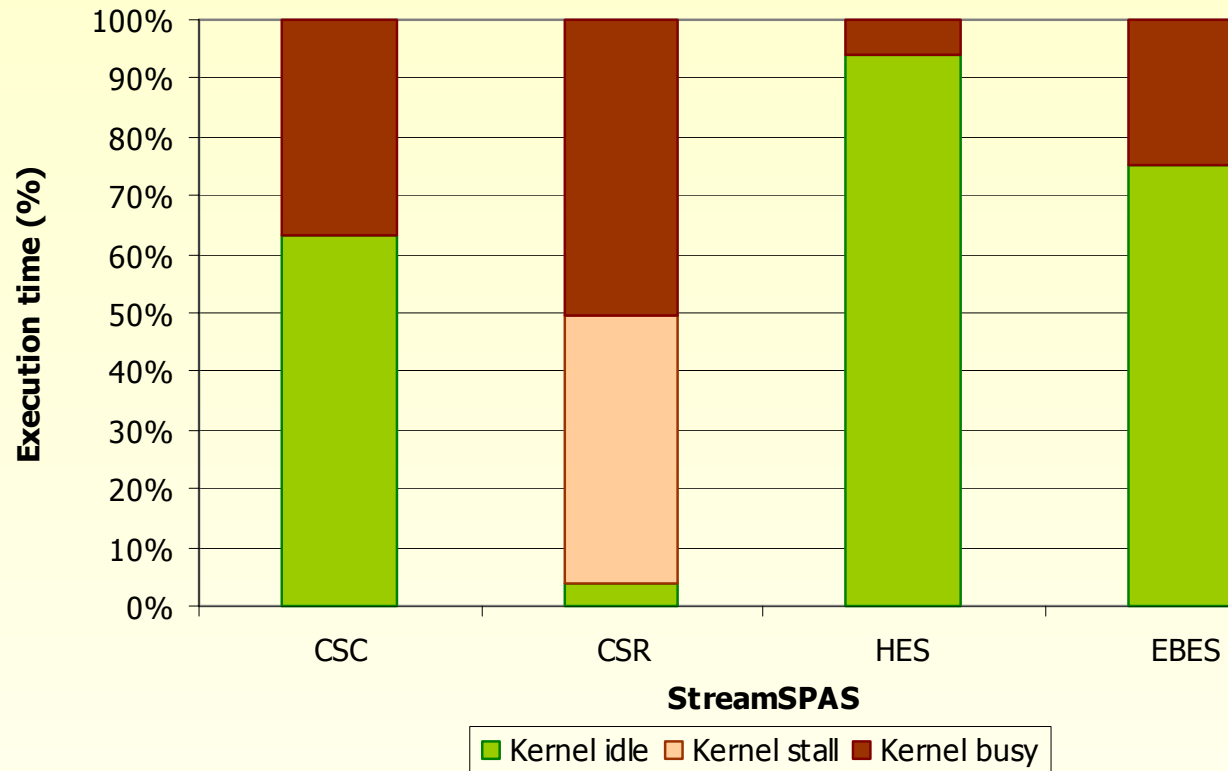
# New version – Execution time

(data - 1918)



# New version – Kernel Occupancy

(Mesh 1918, poly 3)



# New version – Performance Highlights

Multiplication performance (GFLOPS) – 64GFLOPs peak

Polynomial	CSC	CSR	HES	EBES
1	0.813	1.046	0.724	1.781
2	1.213	1.834	0.924	2.689
3	0.834	1.861	0.724	3.976

SRF bandwidth (GB/s) – 512GB/s peak

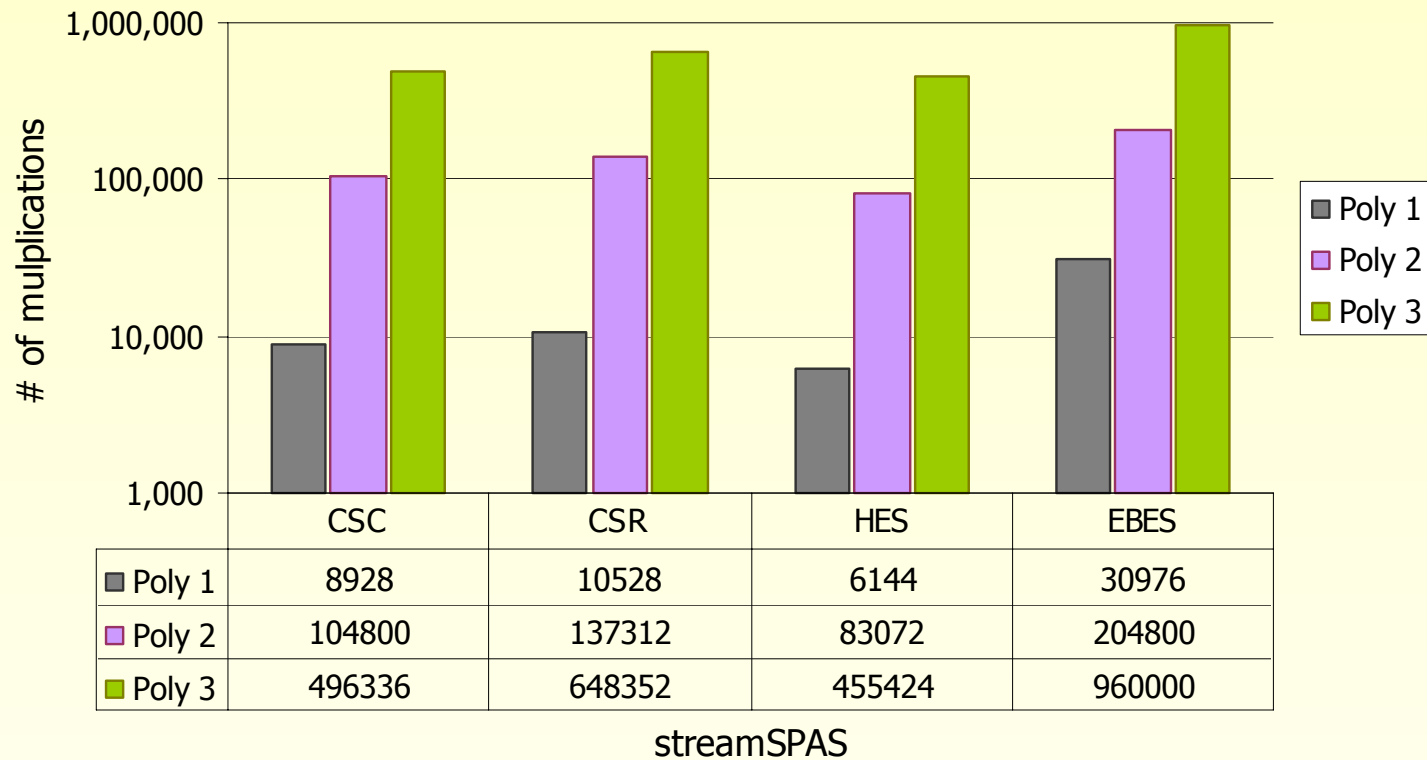
Polynomial	CSC	CSR	HES	EBES
1	32.48	42.12	53.60	56.51
2	51.18	80.69	70.50	56.46
3	45.57	84.33	55.71	61.07

Memory bandwidth (GB/s) – 64GB/s peak for cache, 38.4GB for DRAM

Polynomial	CSC	CSR	HES	EBES
1	16.14	20.93	26.80	24.72
2	25.26	40.32	35.25	28.23
3	25.71	42.16	27.85	30.83

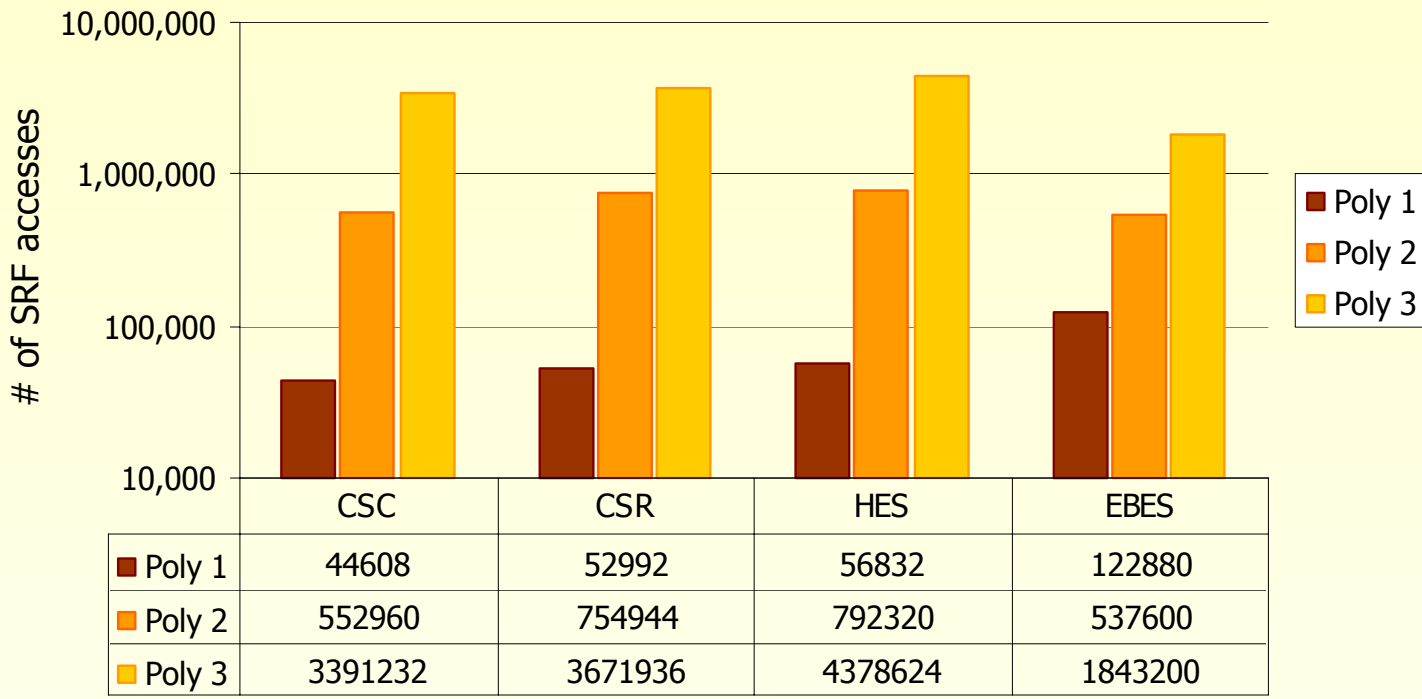
# New Version - Muplication

(data - 1918)



# New Version – SRF Accesses

(Data - 1918)

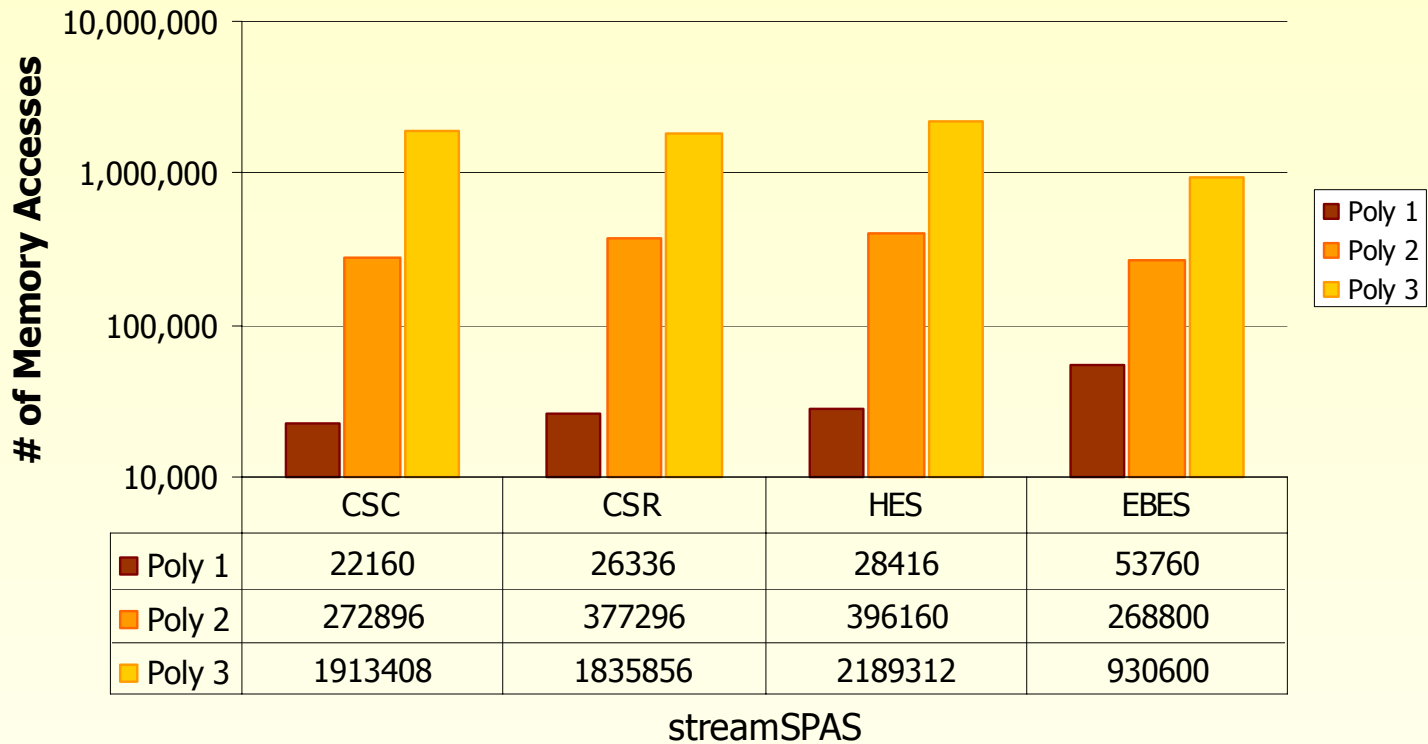


	CSC	CSR	HES	EBES
■ Poly 1	44608	52992	56832	122880
■ Poly 2	552960	754944	792320	537600
■ Poly 3	3391232	3671936	4378624	1843200

streamSPAS

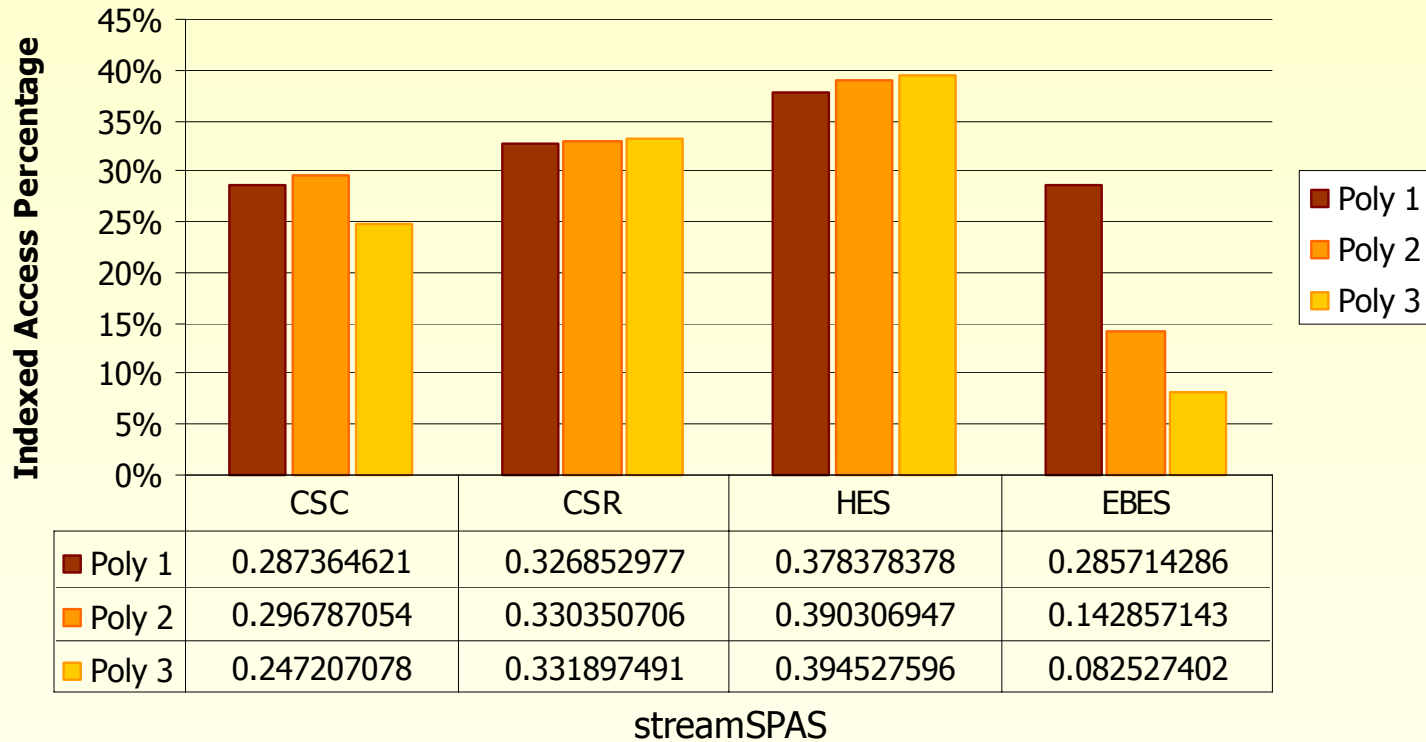
# New Version – Memory Accesses

(Data - 1918)



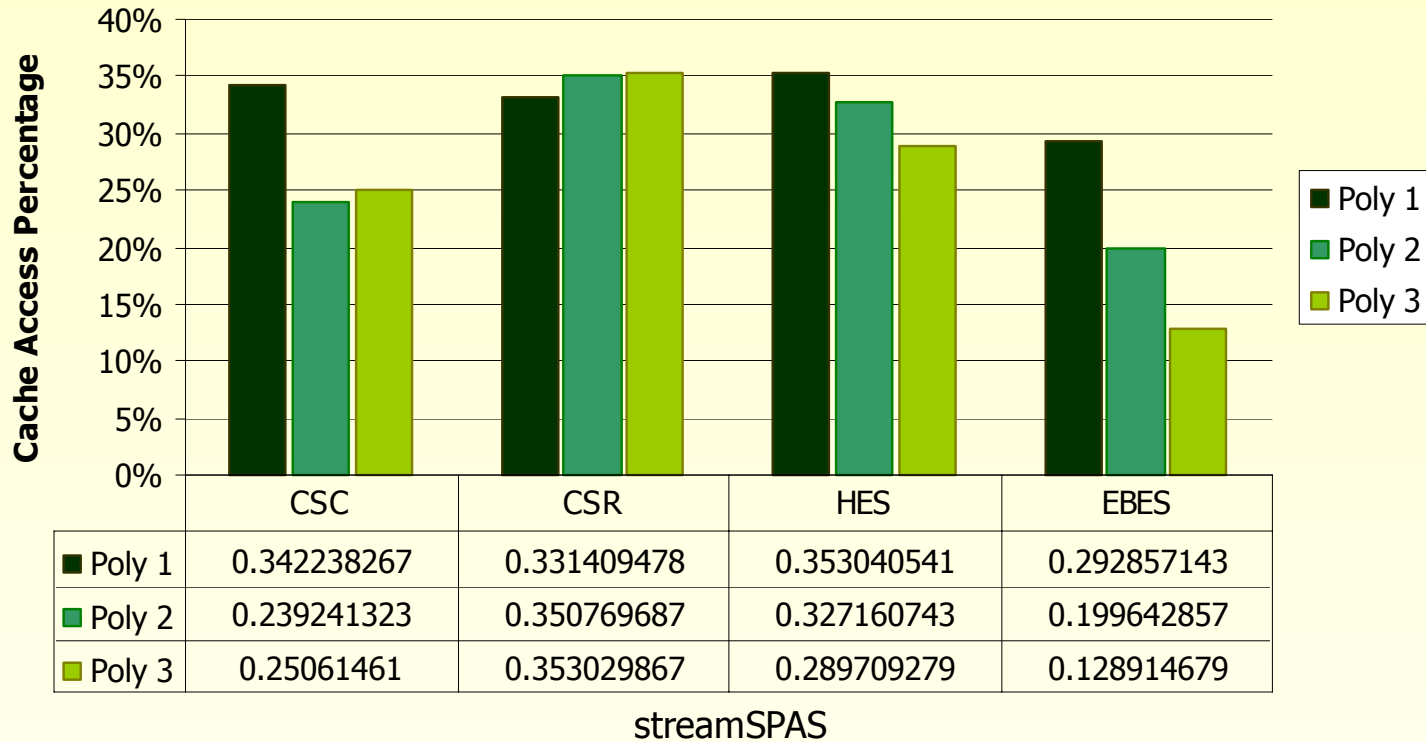
# New Version – Idx Mem Accesses

(Data - 1918)



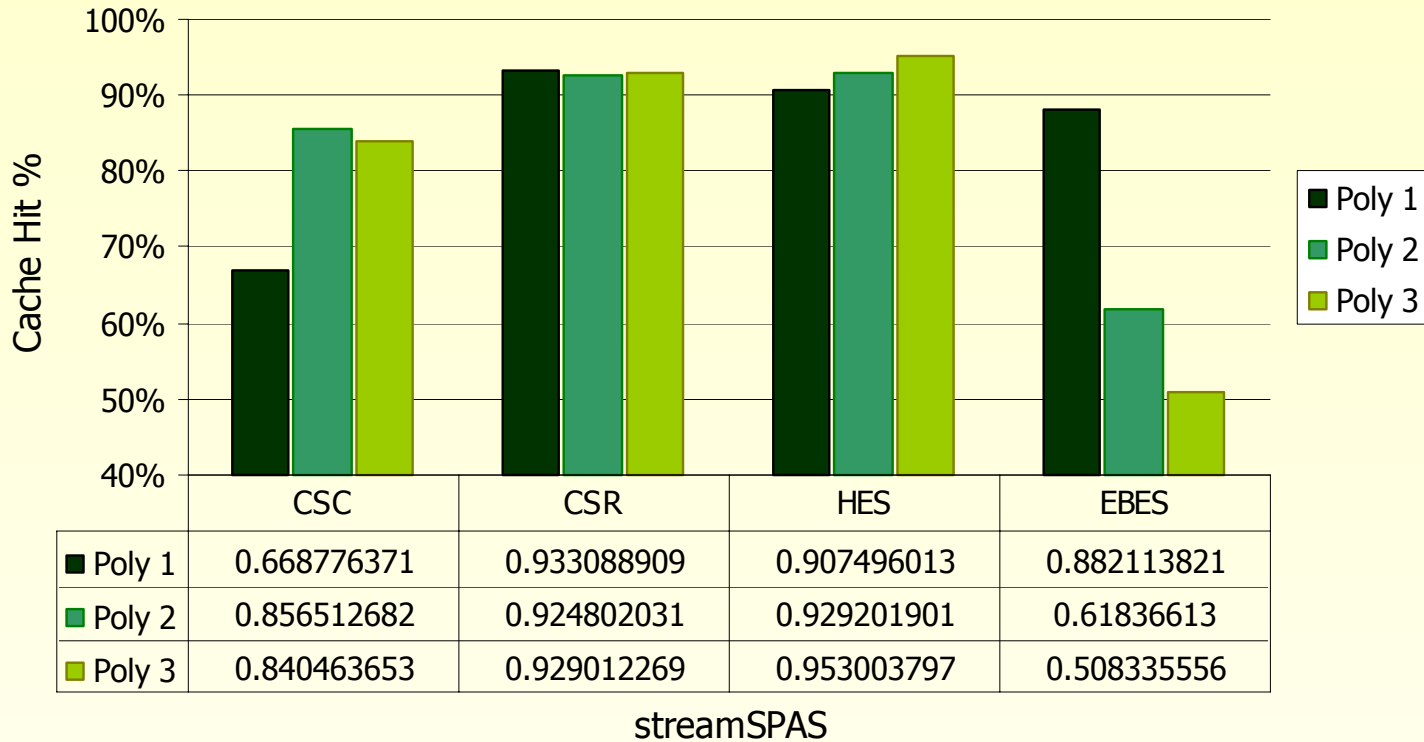
# New Version – Cache Accesses

(Data - 1918)



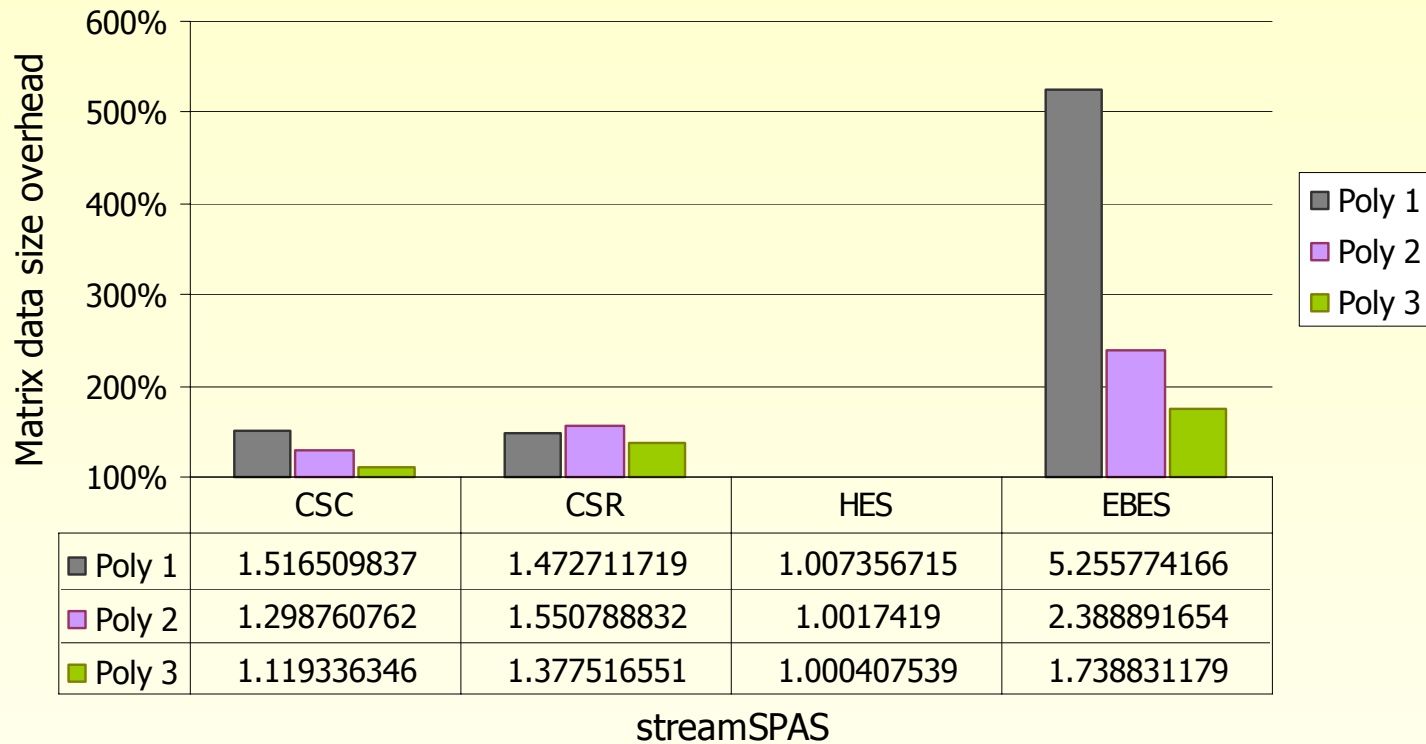
# New Version – Cache Hit Rate

(Data - 1918)



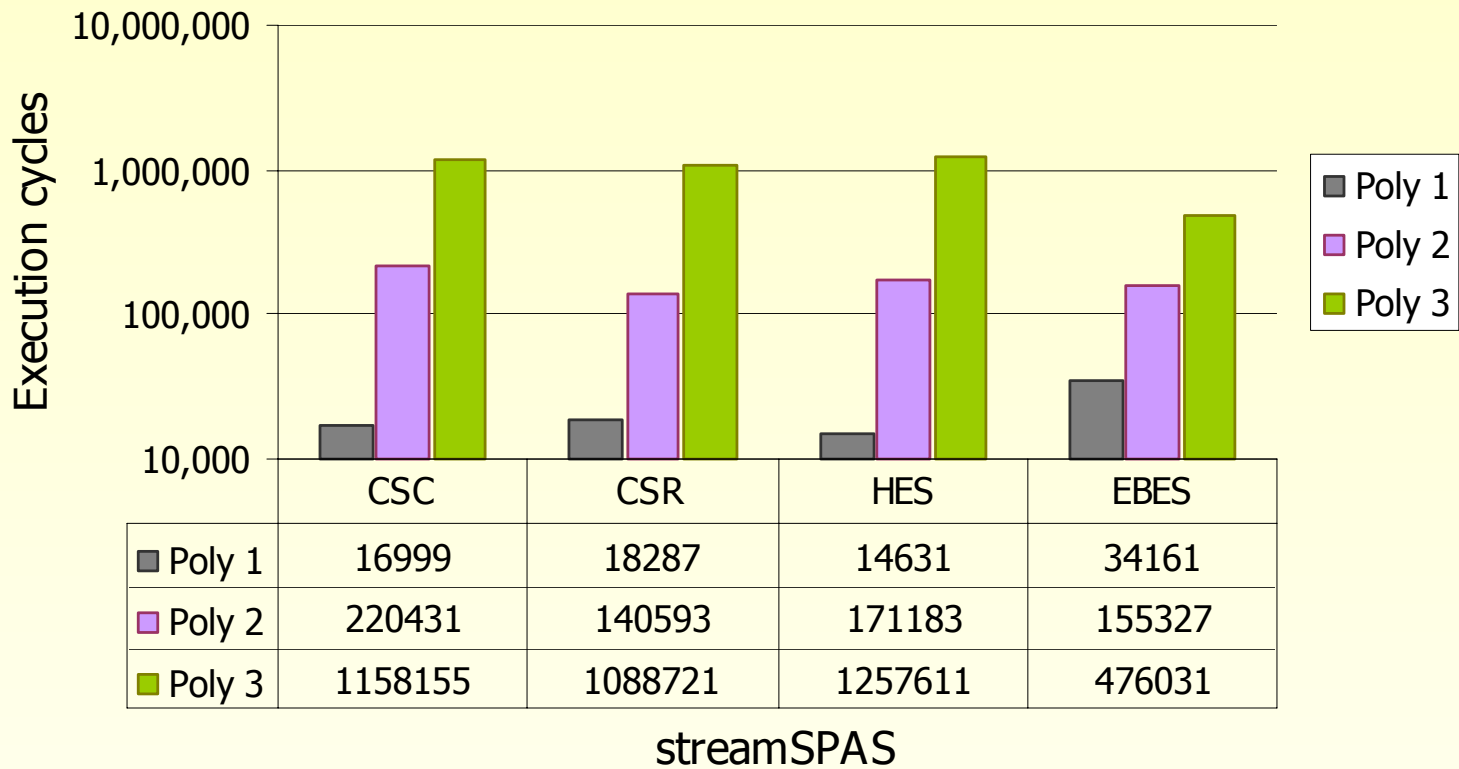
# New Version – Matrix Data Overhead

(Data - 1918)



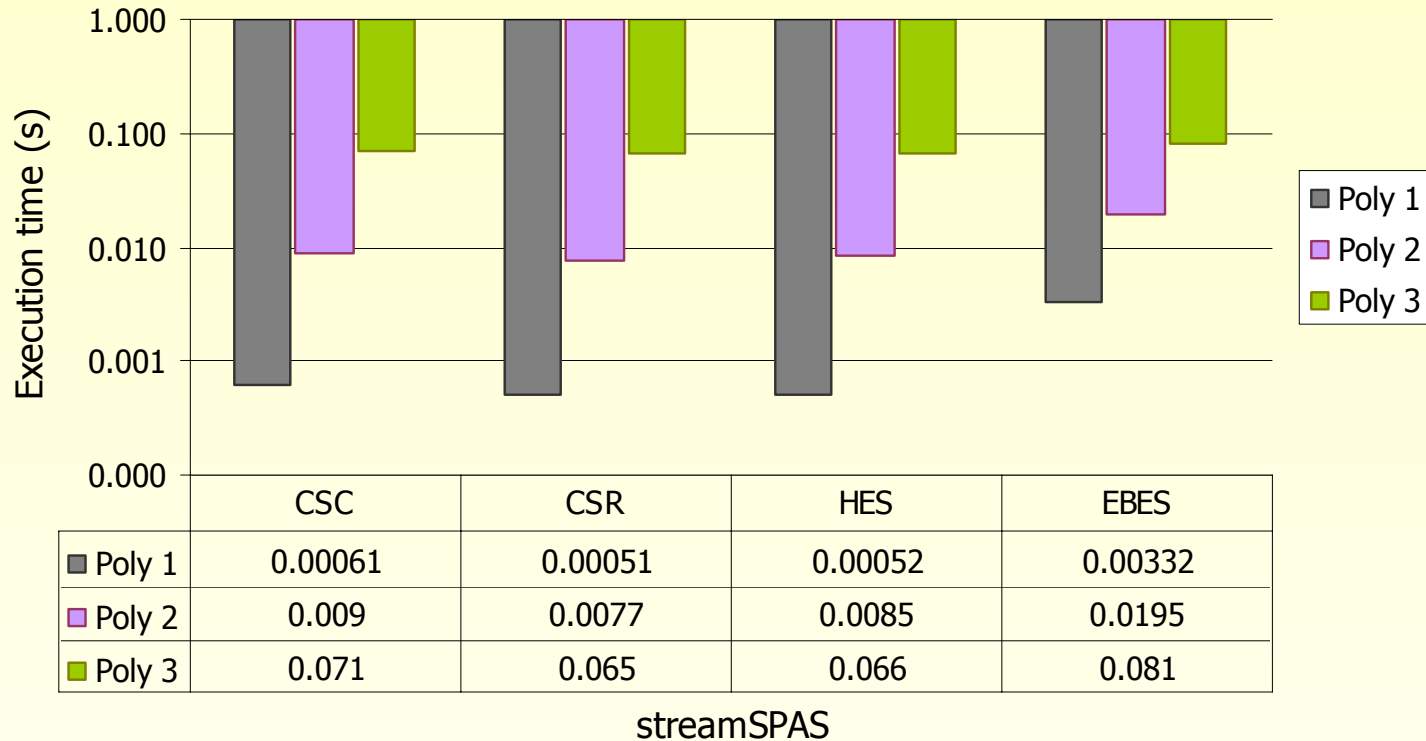
# New version – Execution time(3787)

(Data - 3787)



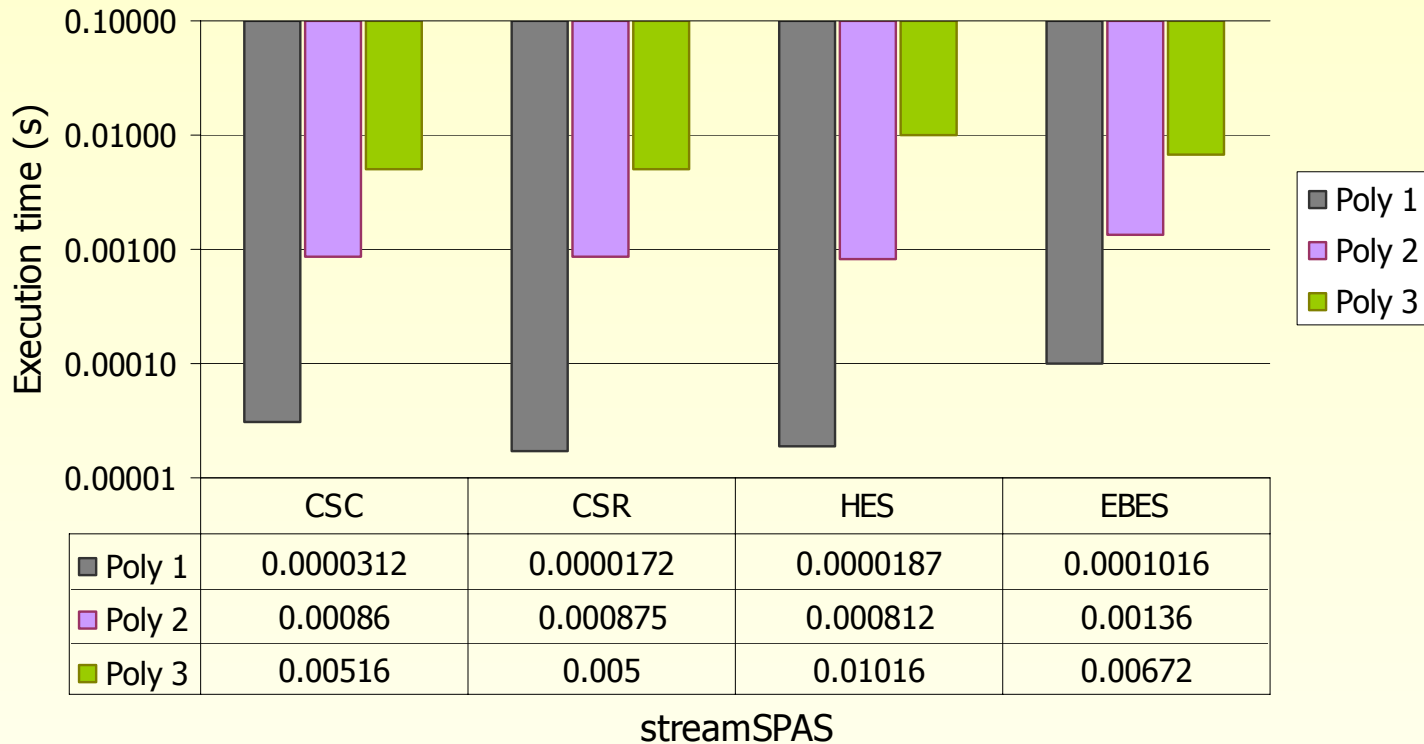
# C++ version at Dim-Sum

(Data - 1918)



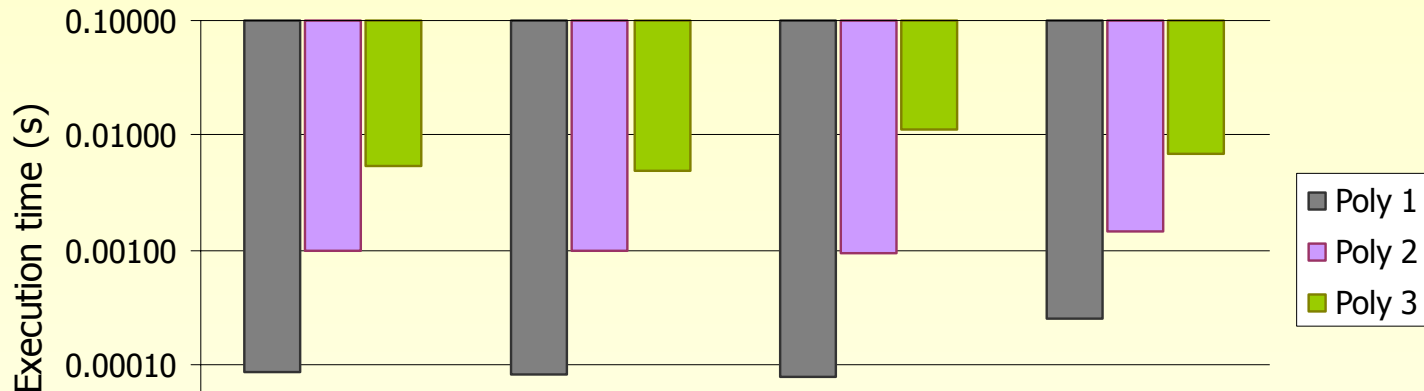
# C++ version at Pentium 4

(Data - 1918, without SSE2)



# C++ version at Pentium 4

(Data - 1918, without SSE2, with flushing)



	CSC	CSR	HES	EBES
■ Poly 1	0.0000843	0.0000797	0.0000766	0.0002531
■ Poly 2	0.000953	0.000953	0.000937	0.001453
■ Poly 3	0.00547	0.00499	0.01093	0.00671

streamSPAS

# C++ version at Pentium 4

## ■ Compiler

- Intel C++ compiler 7.1
- Used intrinsic for coding SSE2 instructions

## ■ Optimization options

- Full optimization (/Ox)
- Global Optimization (/Og)
- Favor Fast Code (/Ot)
- Optimize for Pentium® 4 or Above (/G7)
- Use Processor Extensions : Pentium® 4 instructions (/QaxW)
- Enable Parallelization (/Qparallel)
- Loop Unrolling : up to 4

# Pentium 4 – SSE2

## ■ Multimedia Extensions

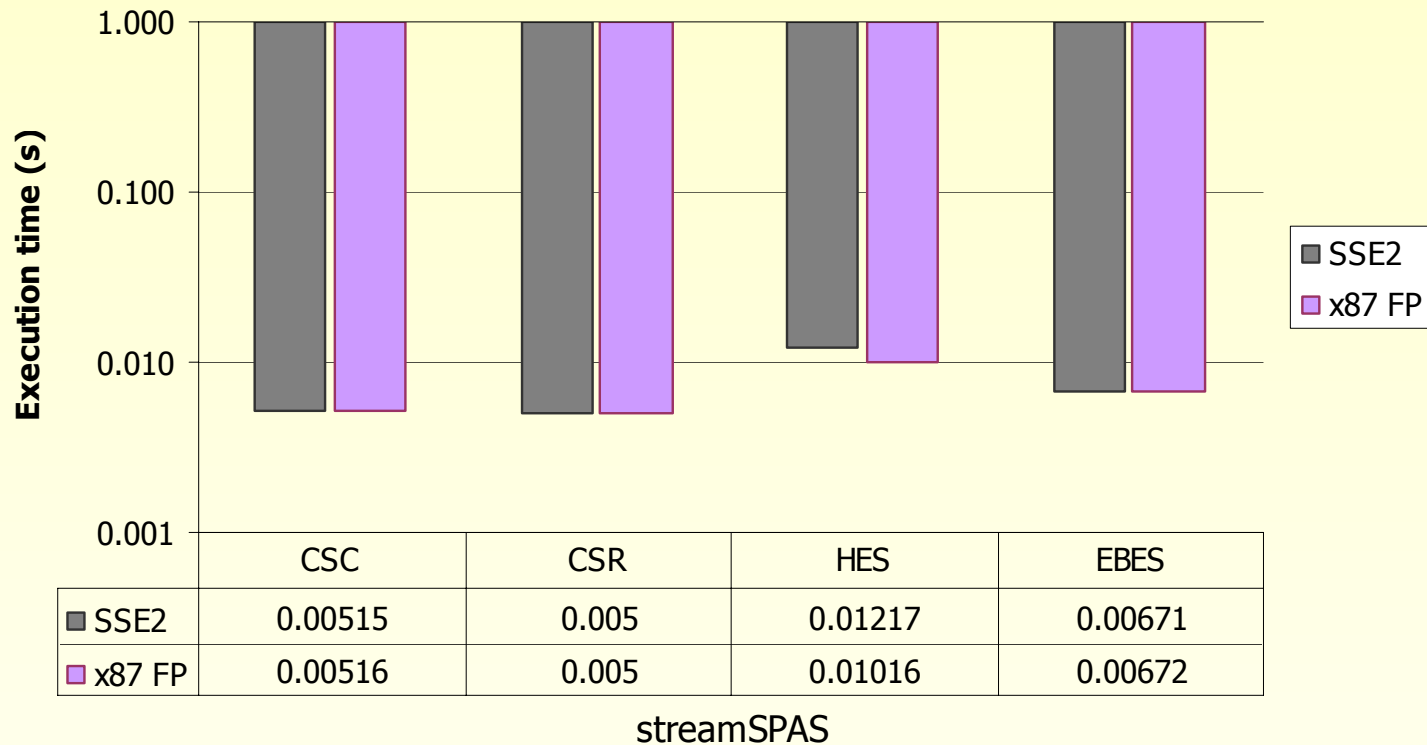
- PowerPC – AltiVec
- Sun – VIS
- Intel – MMX, SSE, SSE2

## ■ Intel SIMDs

- MMX
  - 64 bit, integer, sub word
- SSE (Streaming SIMD Extensions)
  - 128 bit, single precision FP
- SSE2 (Streaming SIMD Extensions 2)
  - 128 bit, double precision FP

# Pentium 4 – SSE2 vs. x87 FP

(Data - 1918, Poly - 3)



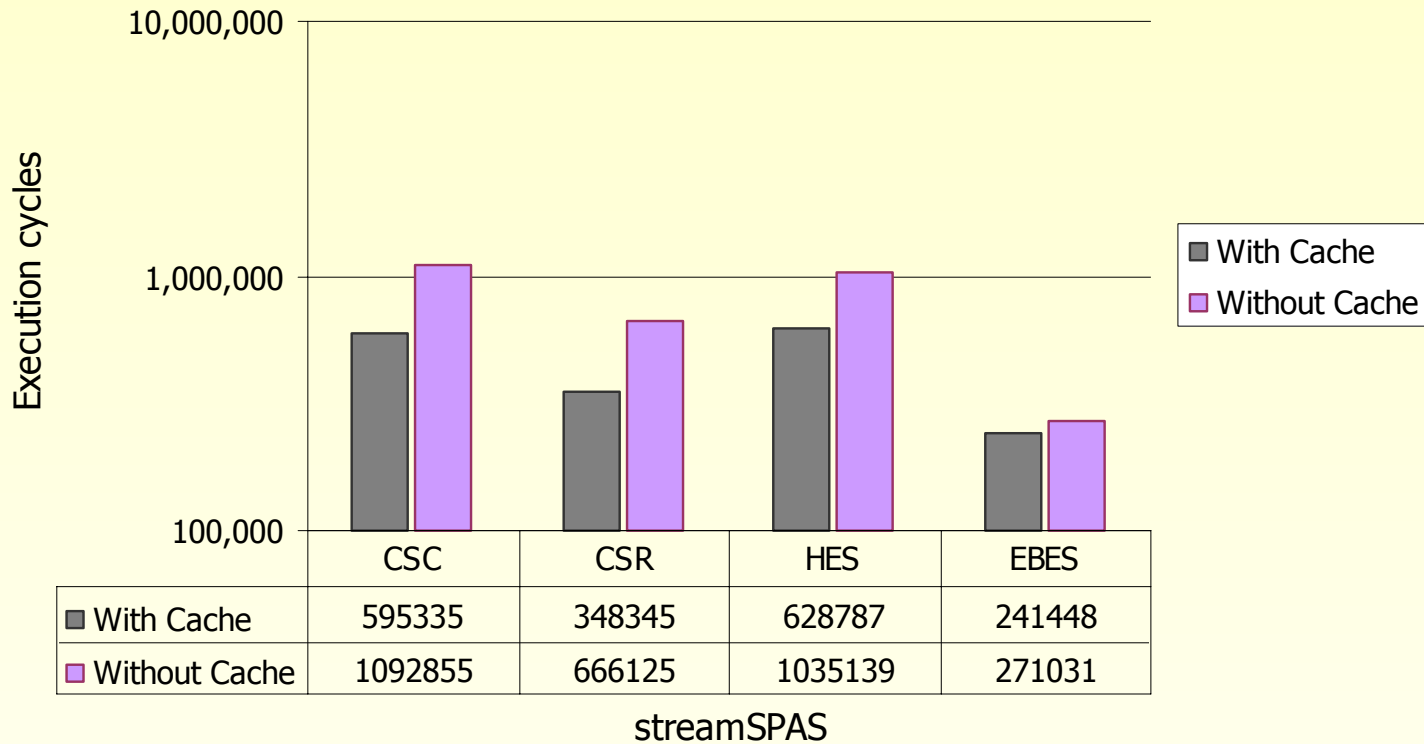
# Pentium 4 – SSE2 vs. x87 FP

## ■ Why no difference?

- Matrix data comes from off-chip DRAM
- At least one global memory access for every 2 FP operations
  - one multiplication, one addition
  - 2.1 GB/s peak DRAM bandwidth
    - sequential access gave 1.25GB/s
    - $1.25\text{GB/s} = 156\text{MW/s}$
    - $2.8\text{GHz} / 156(\text{MW/s}) = 18 \text{ cycle} / \text{word}$
  - 2 FP definitely takes less than 18 cycles
- So execution time depends on total number of off-chip DRAM accesses

# With vs. Without Cache

(Data - 1918, Poly - 3)



# Conclusion

- Reorganizing data boosts performance
- EBES vs. CSR
  - EBES needs redundant computation and communication – worse for small data sets (dominant factor)
  - EBES needs smaller memory and SRF accesses for large data sets – memory bandwidth is the dominant factor for these cases.
- CSC vs. HES
  - C versions of these algorithms have the same amount of computation.
  - StreamC version of CSC has redundancy for better parallelization.
  - But HES takes more time for bigger matrixes because it has bigger size of input data from memory (indexed stream).

# Conclusion (Cont.)

## ■ CSR vs. CSC

- C versions of CSR, CSC and HES algorithms have the same amount of computation.
- StreamC version of CSR has a little bit more redundancy than CSC for better parallelization.
- But CSR takes less time because it doesn't have a scatter-add operation to accumulate output. it happens in clusters. CSR achieves near-peak performance of the memory system because two of three memory loads are sequential accesses, and the other gather operation has around 95% of cache hit.
- EBES also has a scatter-add operation, but because the size of the stream to be scatter-added are 1/10 of CSC and HES cases, it doesn't affect the performance that much.

# Future Work

- Testing more data set other than 1918, 3787
- More fine tuning on each algorithm
  - CSR is suffering from the short stream effect due to the poor performance of the double-buffering
  - Kernel restarting instead of invoking a new kernel per strip will improve performance on EBES
  - Temporary output values are stored back to memory which is not necessary – HES, CSC
- Reverse Cuthill-McKee Ordering