SpEQ: Translation of Sparse Codes using Equivalences

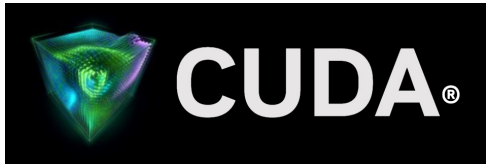
Avery Laird[*], Bangtian Liu[*], Nikolaj Bjorner[†], Maryam Mehri Dehnavi[*]

[*]University of Toronto

[†]Microsoft Research

Optimized Libraries and DSLs

Less General



cuSPARSE

+ Portable

– Harder to optimize

+ Better Performance

– Less Portable

SpEQ: Translation of Sparse Codes using Equivalences

Example Input Code (NPB Benchmarks)

```
for (j = 0; j < lastrow - firstrow + 1; j++) {
  sum = 0.0;
  for (k = rowstr[j]; k < rowstr[j+1]; k++)
    sum = sum + a[k]*p[colidx[k]];
  q[j] = sum;
}
```

SpEQ Transformation

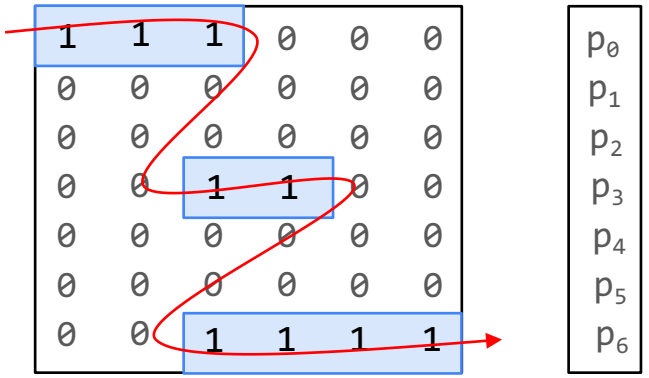
```
if (preconditions(a, rowstr, colidx)) {

  very_fast_spmv(a, rowstr, ..., p, q);

} else {
  // execute original code
}
```

Implicit Precondition:

(a, rowstr, colidx) represent a matrix.



Compressed Sparse Row (CSR) storage format

OpenMP

intel

Sparse and Dense Relationship

Sparse = Computation + Formats

Dense = Computation

Sparse – Formats = Dense = Computation

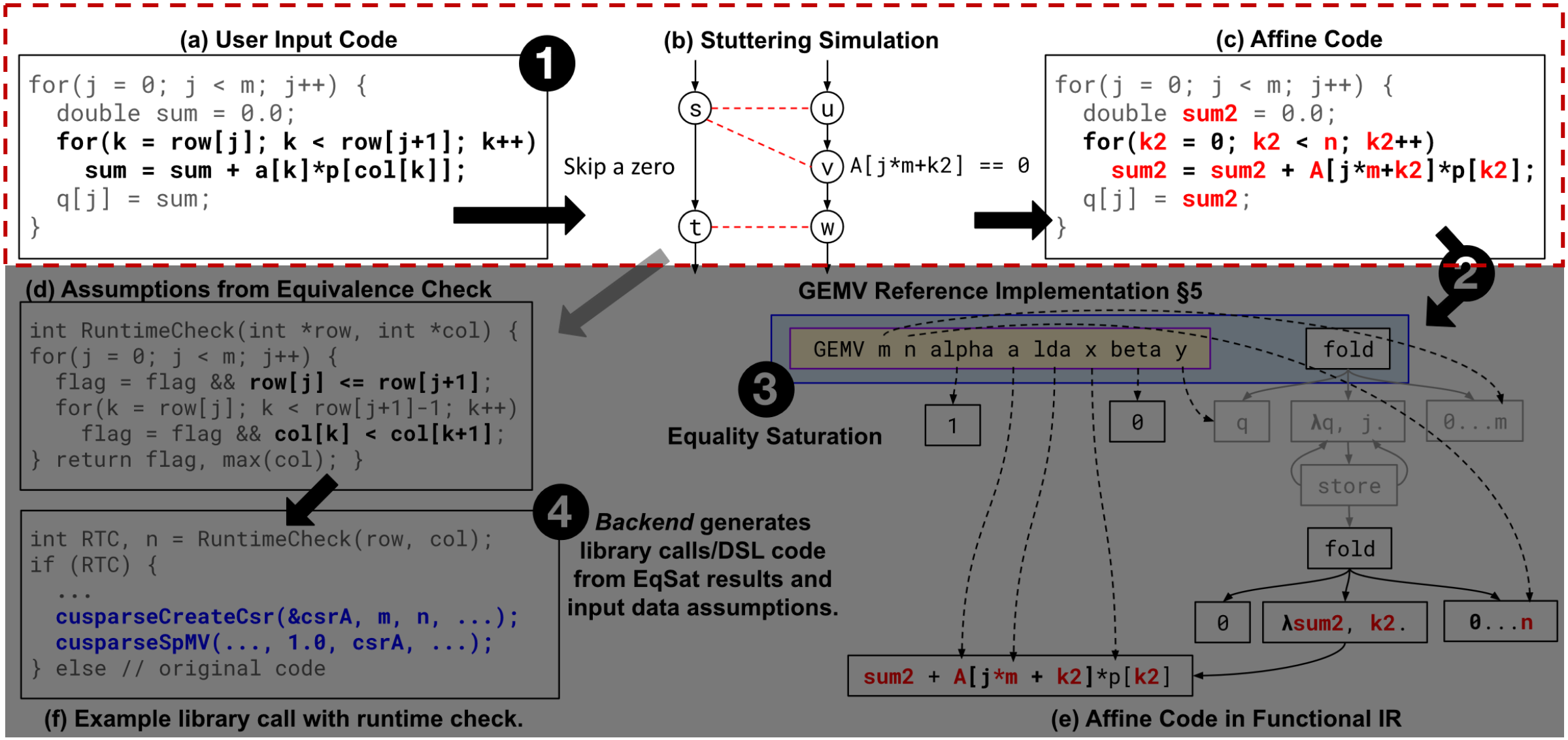
Sparse and Dense Relationship

$SpMV_{CSR} = GEMV + CSR$

$GEMV = GEMV$

$SpMV_{CSR} - CSR = GEMV$

SpEQ: Translation of Sparse Codes using Equivalences

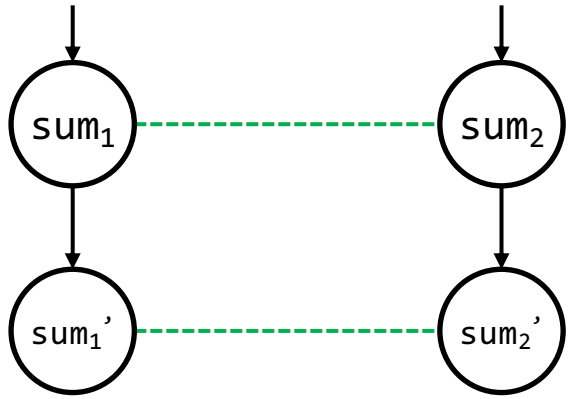


Inferring Storage Format through Equivalences

Create a *precondition-free* version **B**

```
for (j = 0; j < lastrow - firstrow + 1; j++) {
  sum1 = 0.0;
  for (k1 = rowstr[j]; k1 < rowstr[j+1]; k1++)
    sum1 = sum1 + a[k]*p[colidx[k1]];
  q[j] = sum1;
}
```

```
for (j = 0; j < lastrow - firstrow + 1; j++) {
  sum2 = 0.0;
  for (k2 = 0; k2 < M; k2++)
    sum2 = sum2 + A[j][k2]*p[k2];
  q[j] = sum2;
}
```

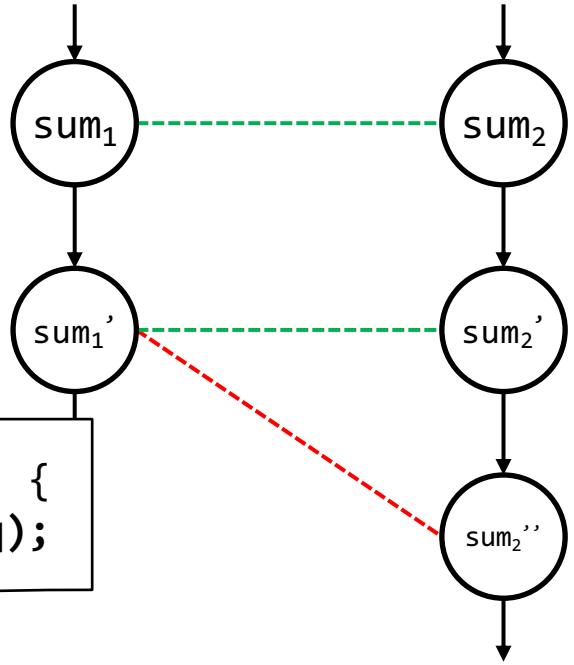


Inferring Storage Format through Equivalences

Create a *Dense* version **B**

```
for (j = 0; j < lastrow - firstrow + 1; j++) {
  sum1 = 0.0;
  for (k1 = rowstr[j]; k1 < rowstr[j+1]; k1++)
    sum1 = sum1 + a[k1]*p[colidx[k1]];
  q[j] = sum1;
}
```

```
for (j = 0; j < lastrow - firstrow + 1; j++) {
  sum2 = 0.0;
  for (k2 = 0; k2 < M; k2++)
    sum2 = sum2 + A[j][k2]*p[k2];
  q[j] = sum2;
}
```



Either, both codes process a non-zero element...

$a[k_1] == A[j][k_2]$
$colidx[k_1] == k_2$

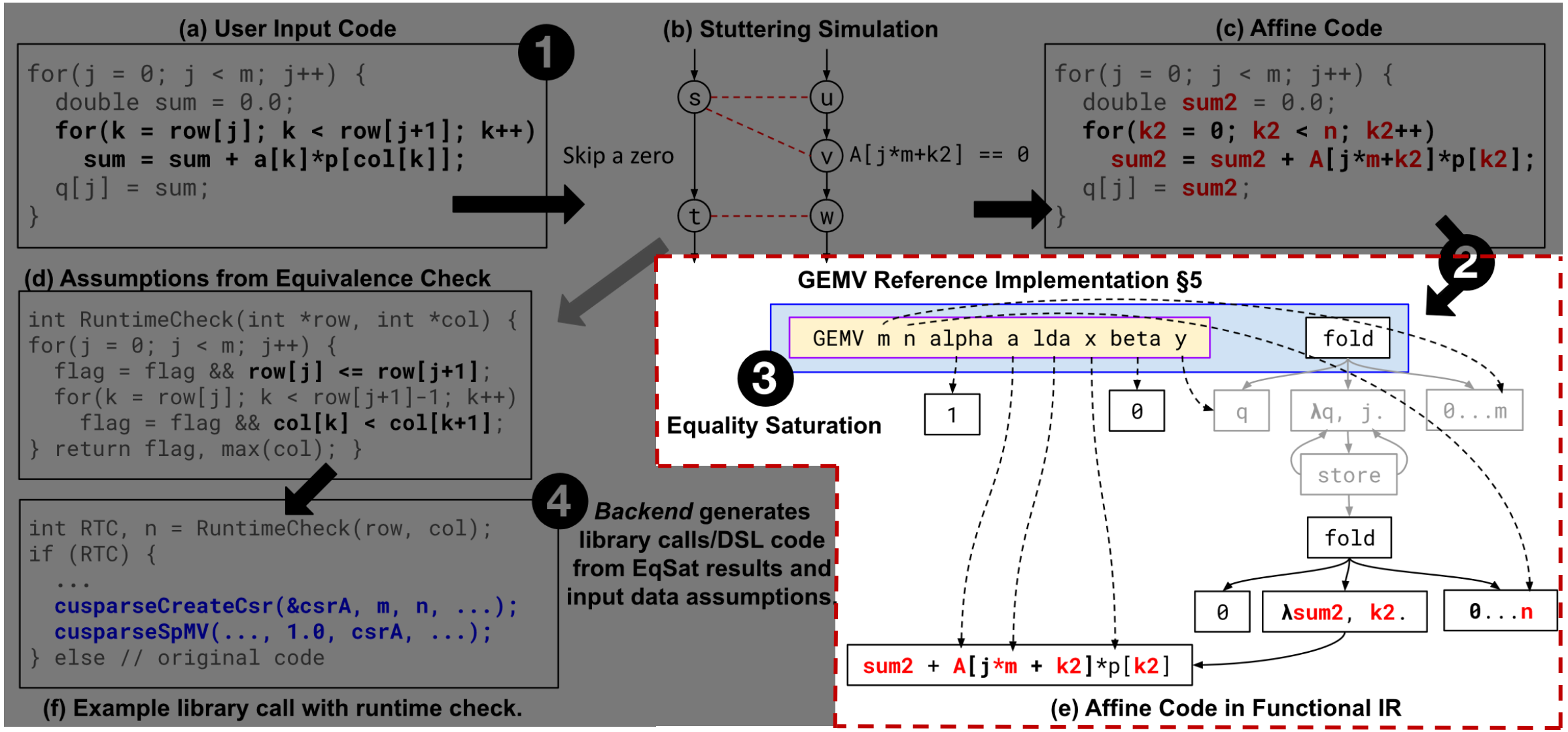
...or, version **B** processes a zero (stuttering)

$A[j][k_2] == 0$

```
if (preconditions(a, rowstr, colidx)) {
  very_fast_spmv(a, rowstr, ..., p, q);
```

Implicit preconditions: the necessary conditions such that both codes are equal.

SpEQ: Translation of Sparse Codes using Equivalences



Identifying Computation Example: GEMV

```
for (j = 0; j < lastrow - firstrow + 1; j++) {
  sum = 0.0;
  for (k2 = 0; k2 < M; k2++)
    sum2 = sum2 + A[j][k2]*p[k2];
  q[j] = sum;
}
```

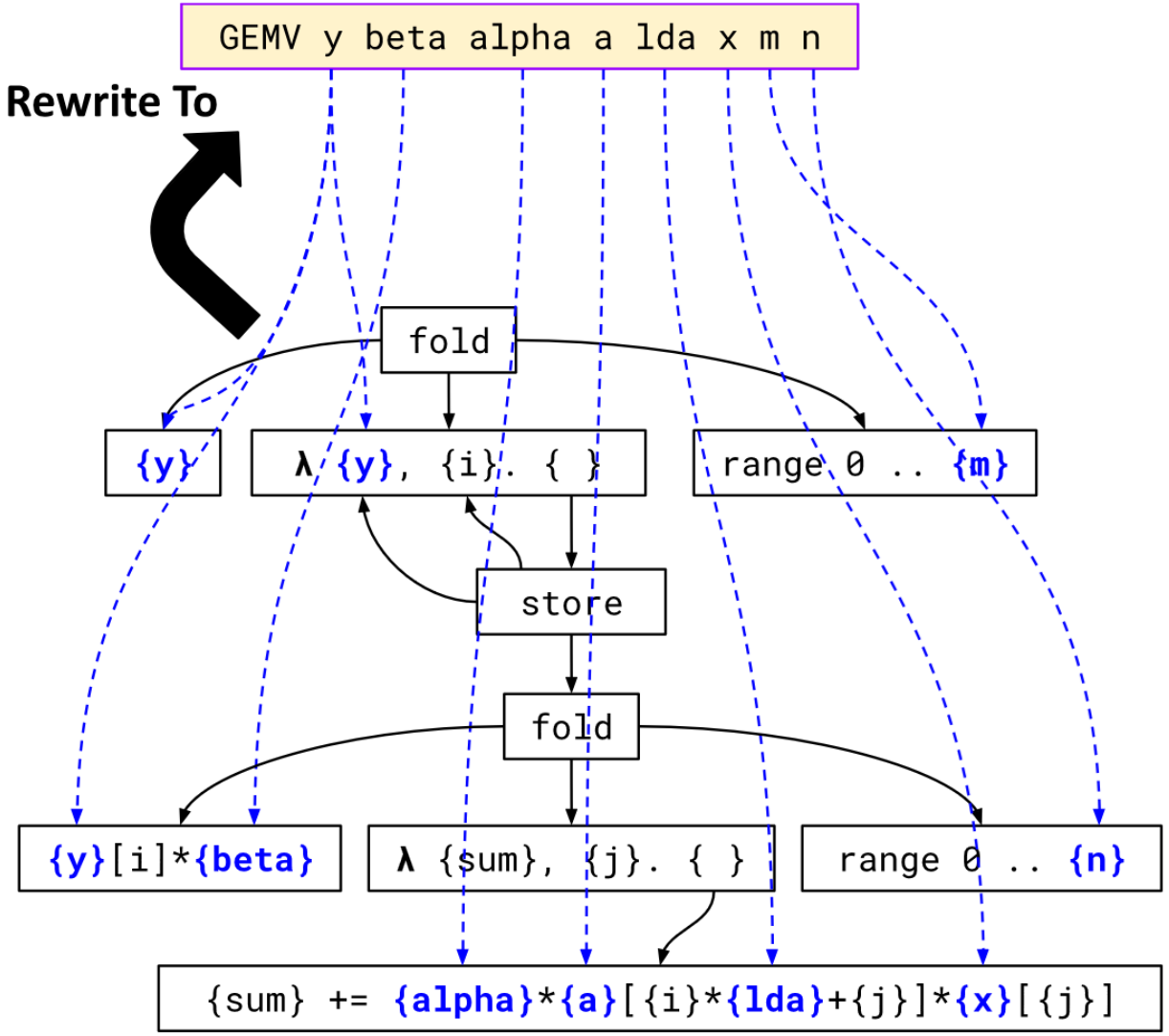
Dense = (GEMV + CSR) – CSR = GEMV

Identifying Computation Example: GEMV

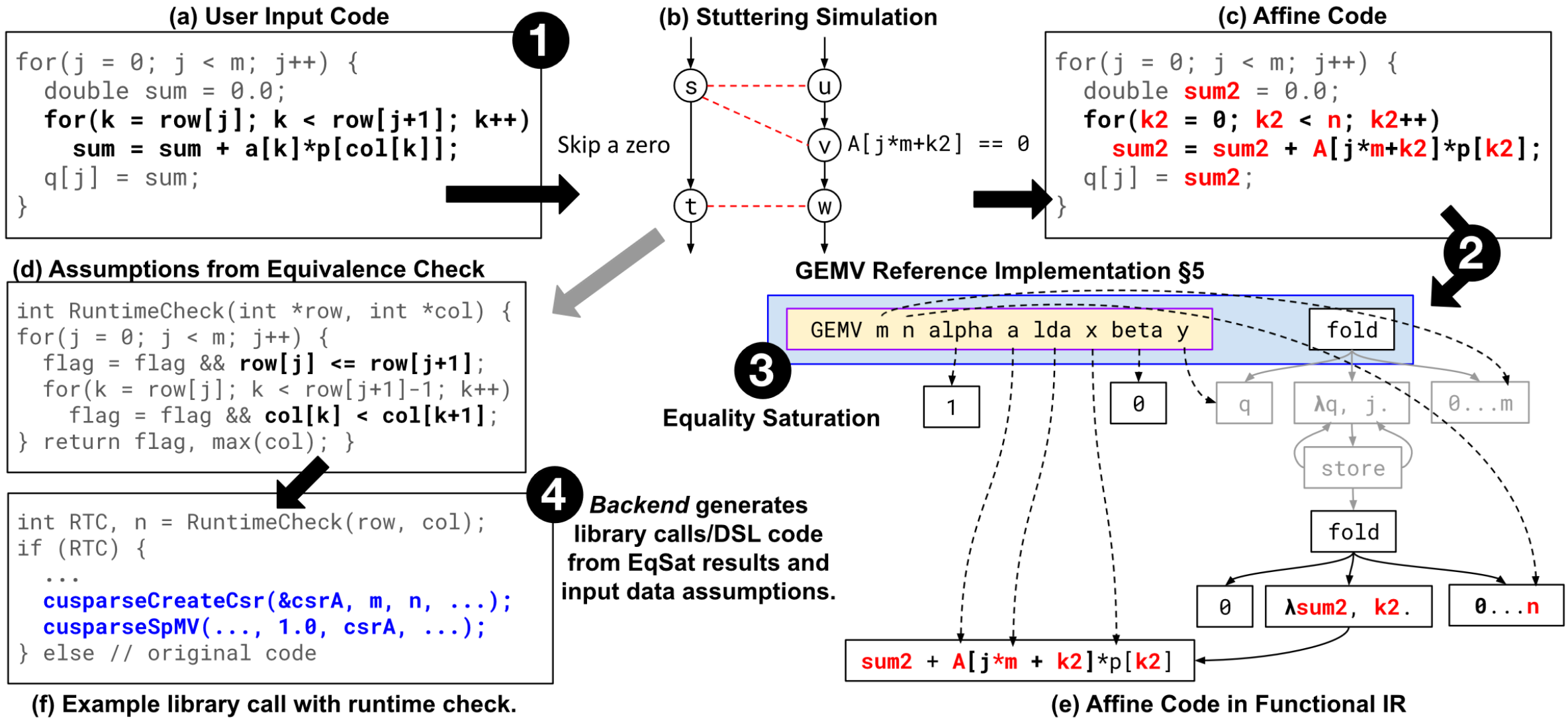
The Computation I want to Recognize

```
void gemv(double *y,
          double beta,
          double alpha,
          double *a,
          int lda,
          double *x,
          int m,
          int n) {
  for (int i = 0; i < m; ++i) {
    double sum = y[i]*beta;
    for (int j = 0; j < n; ++j)
      sum += alpha * a[i*lda + j] * x[j];
    y[i] = sum;
  }
}
```

Function Body Rewrites to Computation “Name”



Putting it All Together

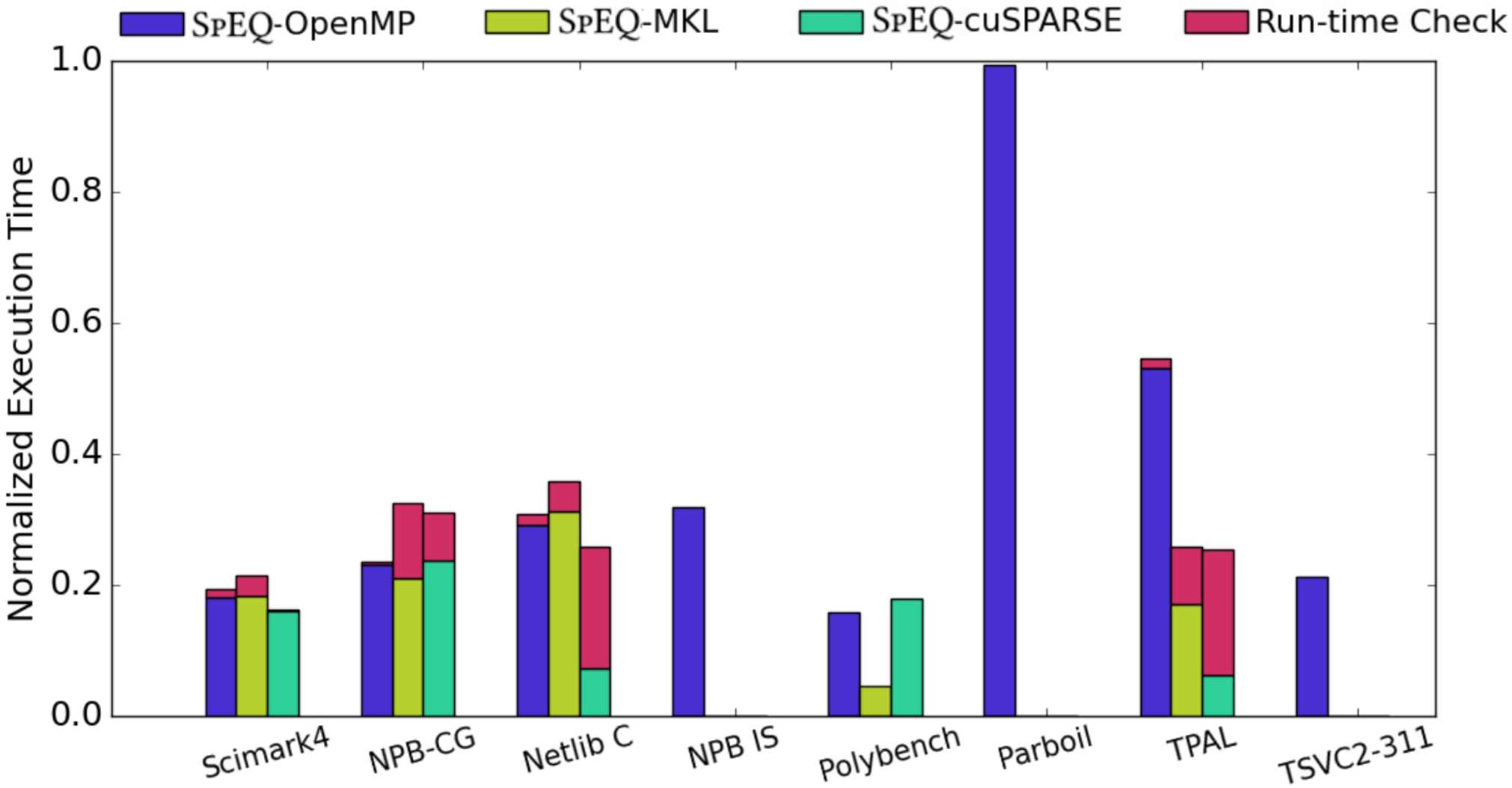


Putting it All Together

LLVM[1]	static analysis/functional representation
Z3[2]	equivalence check
egglog[3]	equality saturation

[1] https://llvm.org/
[2] https://github.com/Z3Prover/z3
[3] https://egglog-python.readthedocs.io/latest/

Performance Benefits



SpEQ automatically achieves up to **8.04x** speedup on average (GPU[1]), with low run-time overhead.

[1] NVIDIA GeForce RTX 3080 Ti GPU

Flexibility and Safety

Tool	GEMM	GEMV		Histogram	Reduction
		SpMV+CSC	SpMV+CSR		
SpEQ	●	●	●	●	●
KernelFaRer	●				
LiLAC			○	●	●

● = correct detection

○ = unsafe detection (preconditions ignored)

Benchmark	Computation	Kernel
Polybench	GEMM	GEMM
TACO-Gen	SpMV+CSC	GEMV
CSparse	SpMV+CSC	GEMV
Scimark4	SpMV+CSR	GEMV
NPB CG	SpMV+CSR	GEMV
Netlib C	SpMV+CSR	GEMV
TPAL	SpMV+CSR	GEMV
NPB IS	Histogram	Histogram
Parboil	Histogram	Histogram
TSVC2	Reduction	Reduction

Thank you!