Exploring QR Factorization on GPU for Quantum Monte Carlo Simulation

Tyler McDaniel
Ming Wong

Mentors: Ed D’Azevedo, Ying Wai Li, Kwai Wong

Quantum Monte Carlo Simulation

Slater Determinant for N-electrons system

$$\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\mathbf{x}_1) & \chi_2(\mathbf{x}_1) & \cdots & \chi_N(\mathbf{x}_1) \\ \chi_1(\mathbf{x}_2) & \chi_2(\mathbf{x}_2) & \cdots & \chi_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(\mathbf{x}_N) & \chi_2(\mathbf{x}_N) & \cdots & \chi_N(\mathbf{x}_N) \end{vmatrix}$$

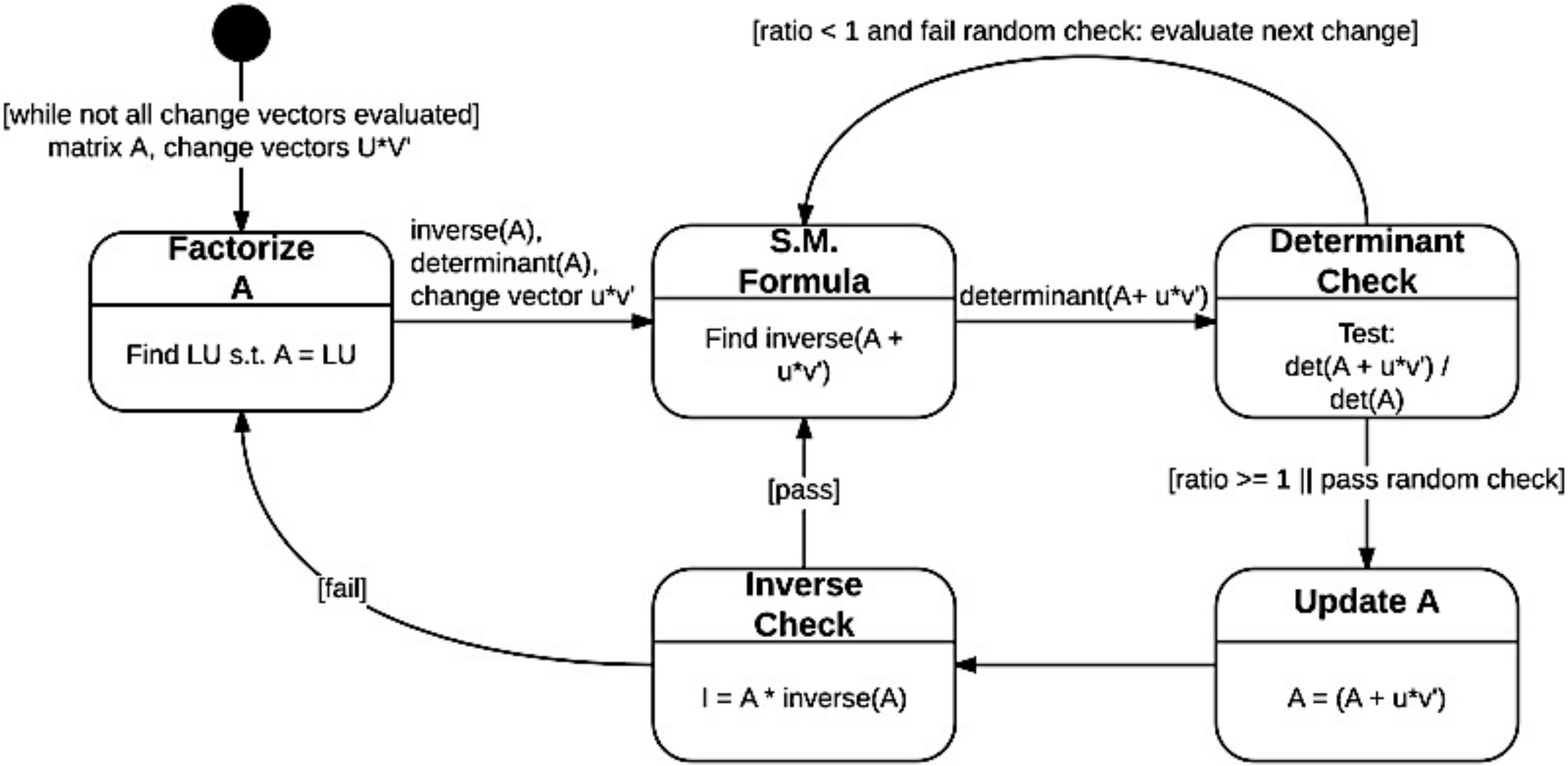
What is QMCPACK?

- Open-Source scientific software for quantum Monte Carlo simulation
- Written in C++ with CUDA kernels
- Utilizes CUDA (acceleration) and openMP (parallelization)

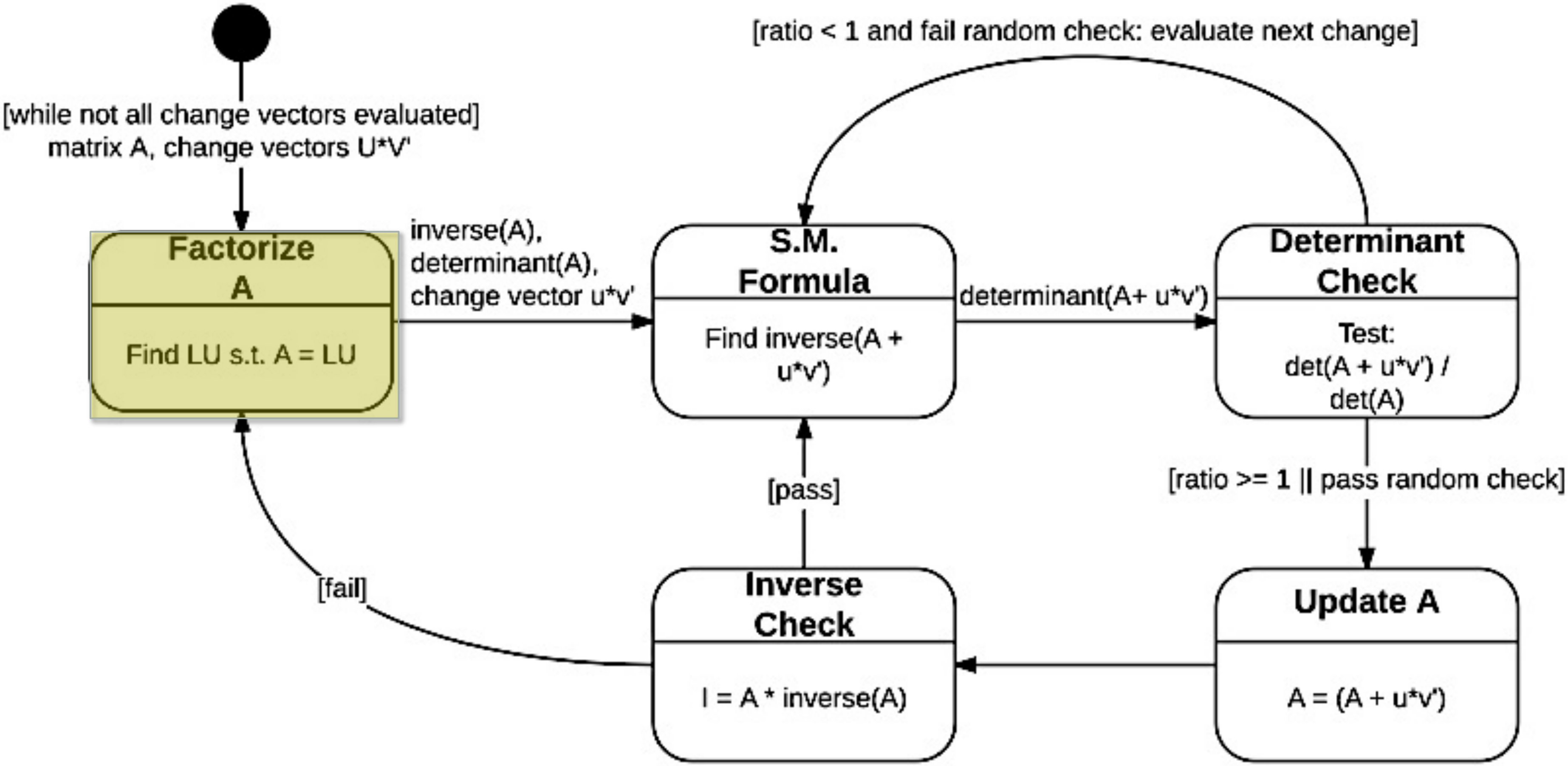
Purpose

- To Improve on the existing method in QMCPACK for evaluating single-particle updates to a system’s electron configuration

Current QMC implementation



Current QMCPACK implementation

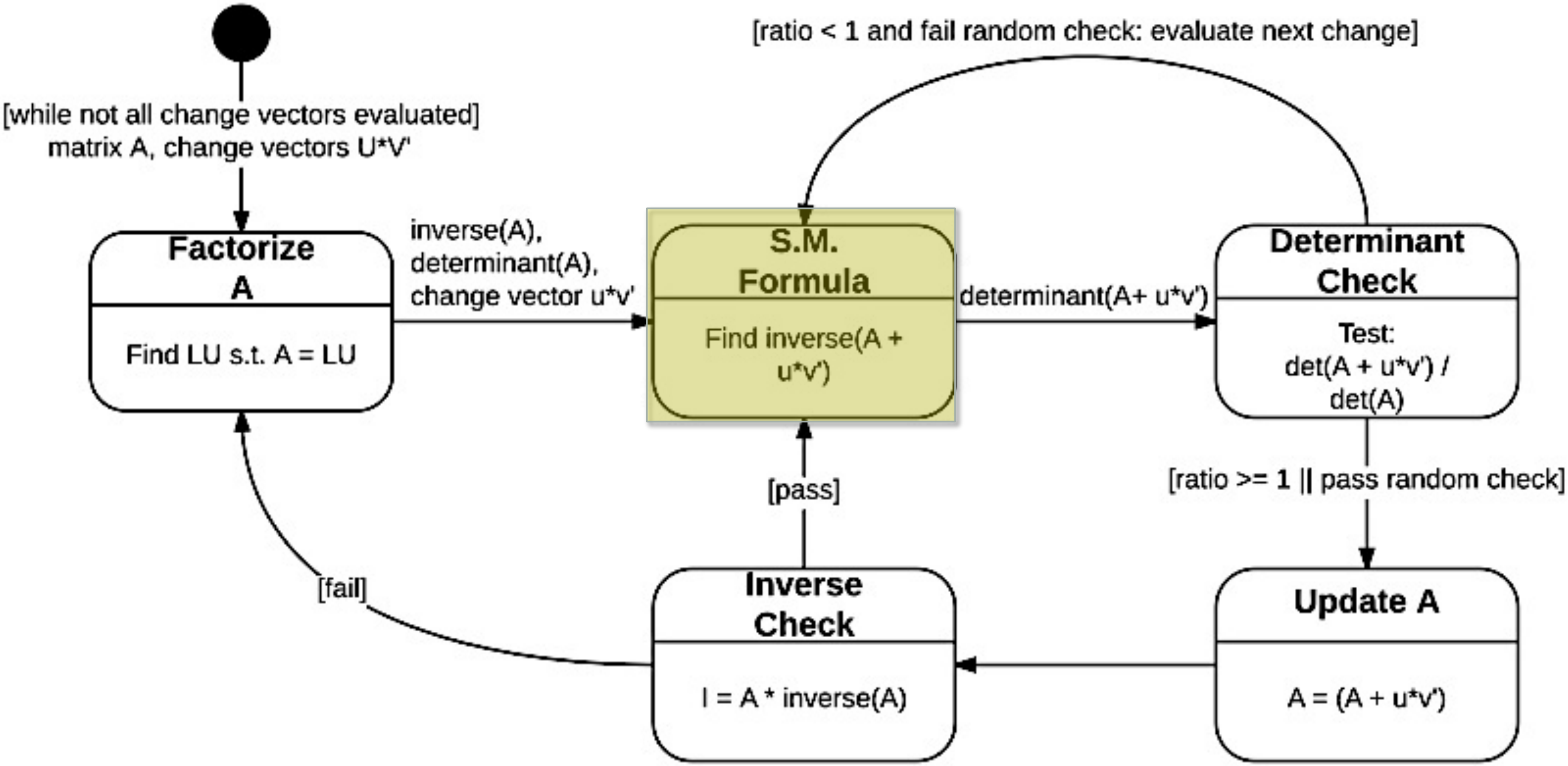


LU Decomposition

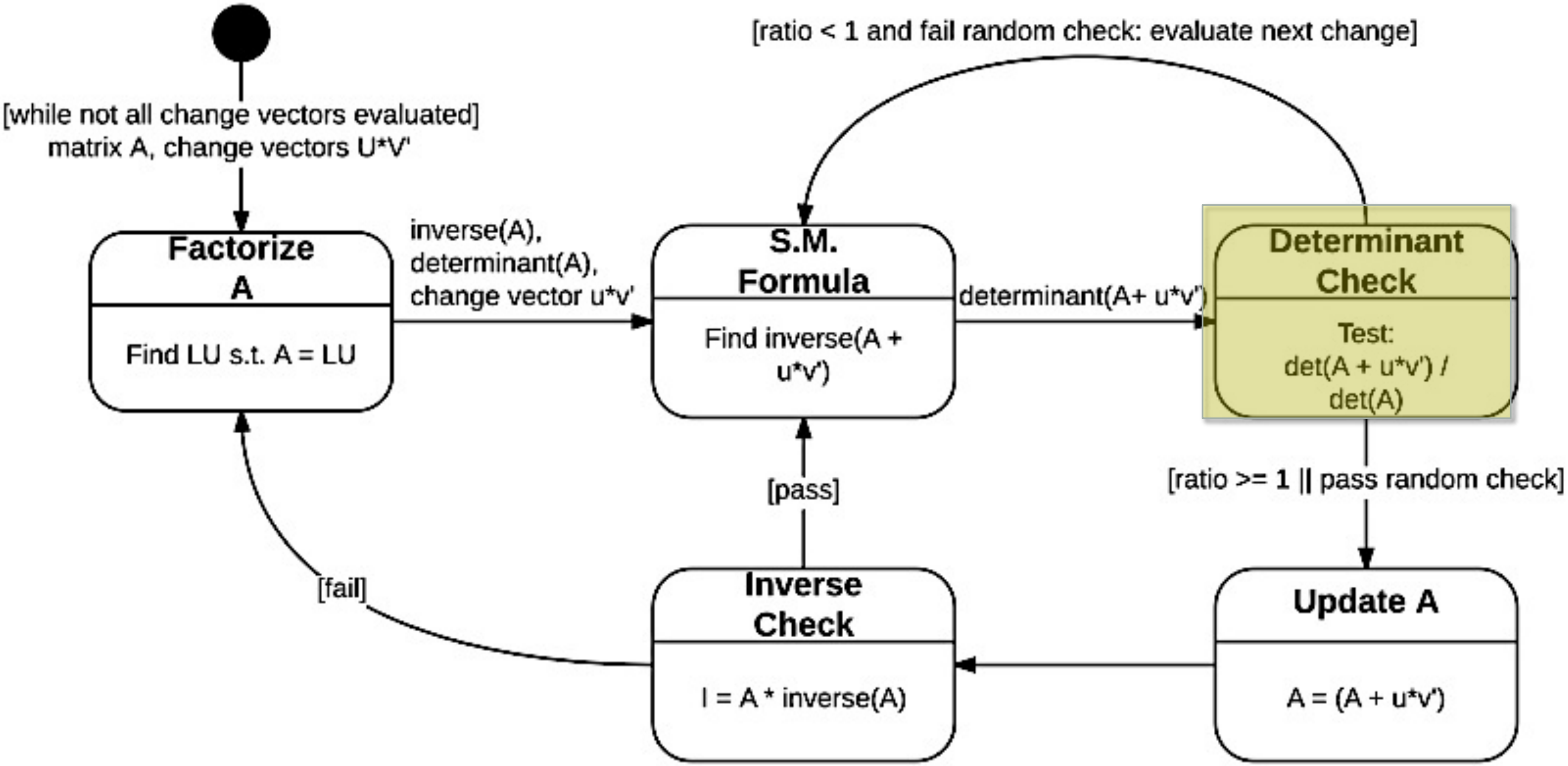
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

$$A \quad = \quad L \quad * \quad U$$

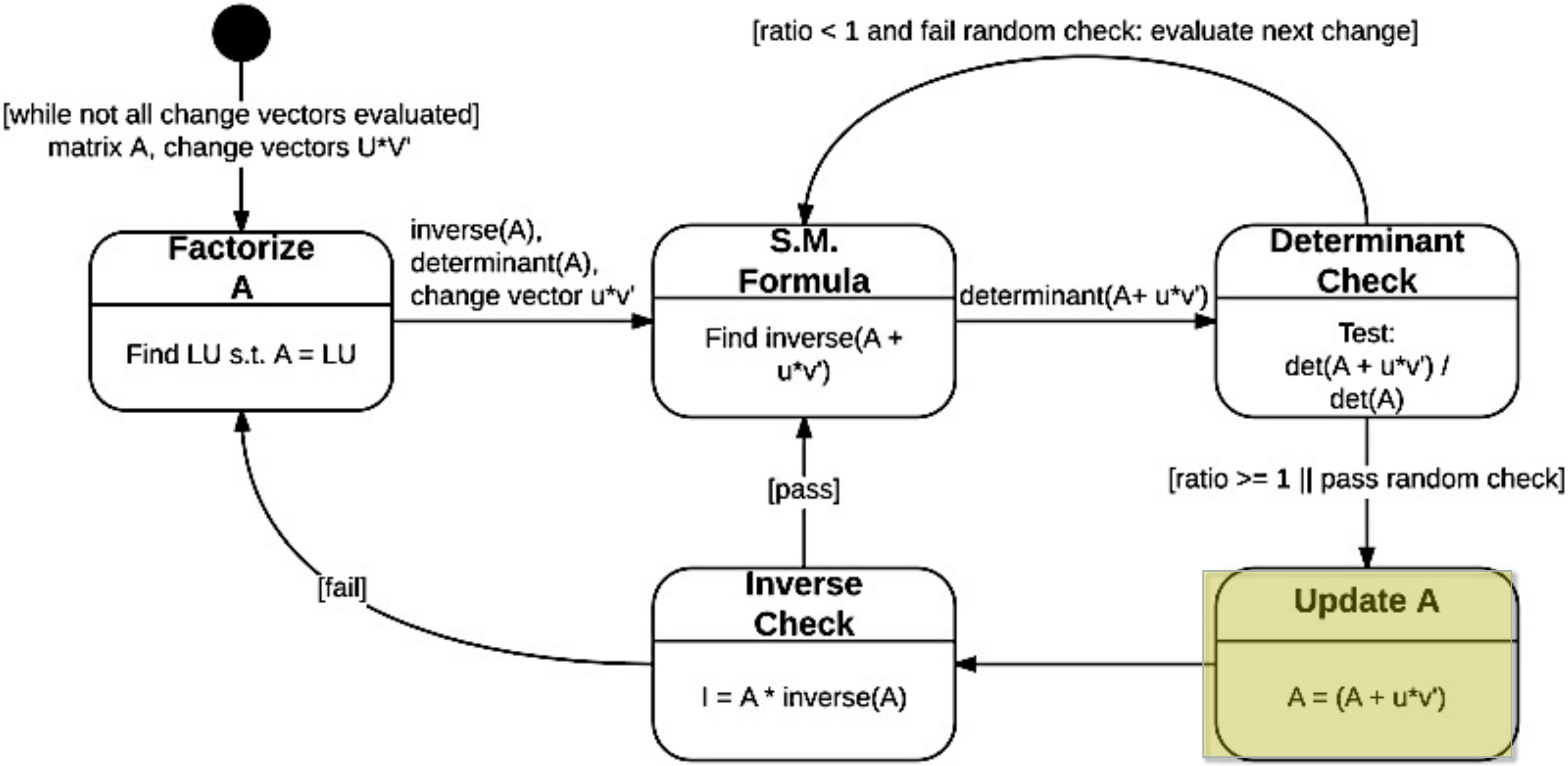
Current QMCPACK implementation



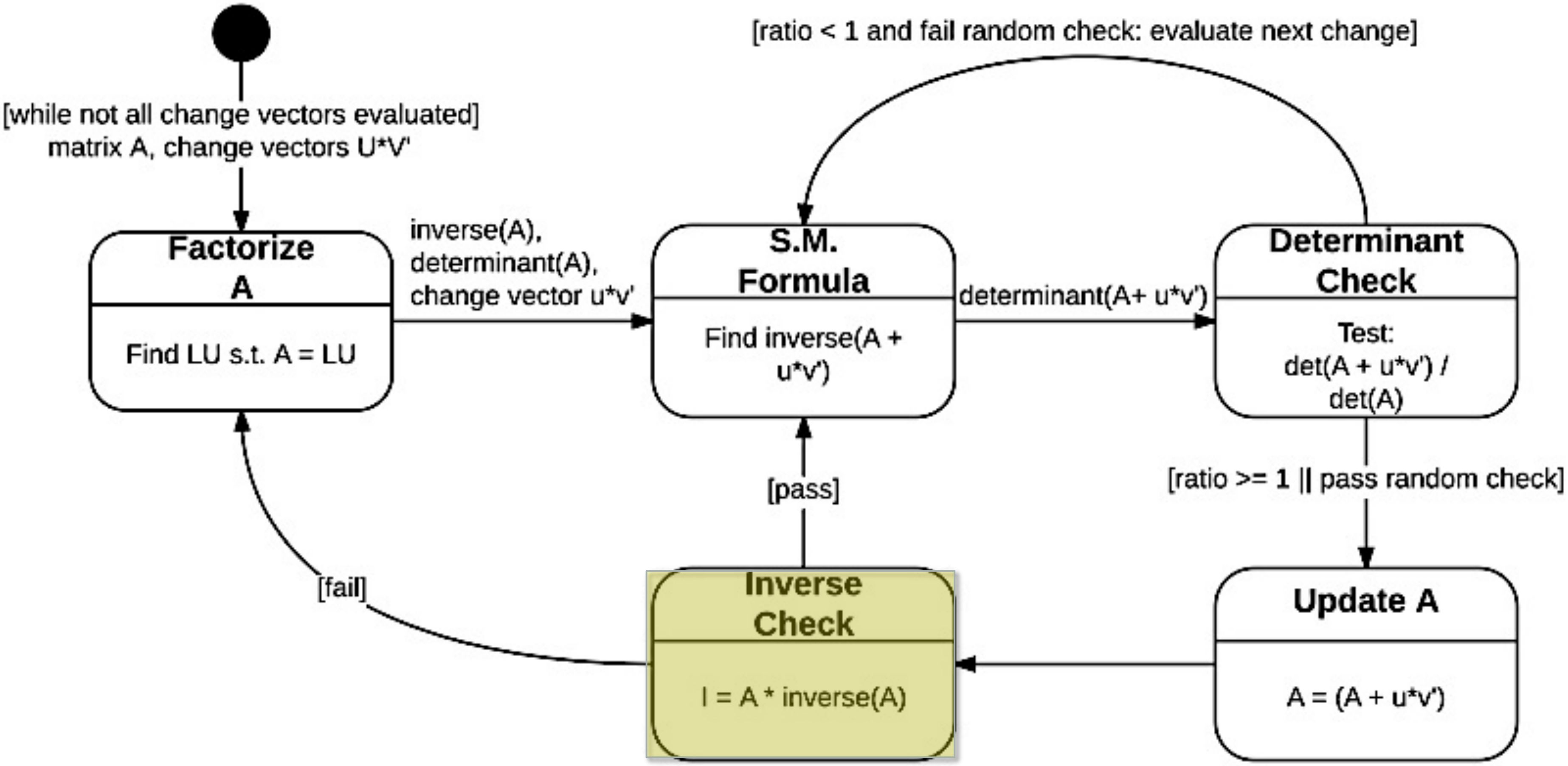
Current QMCPACK implementation



Current QMCPACK implementation



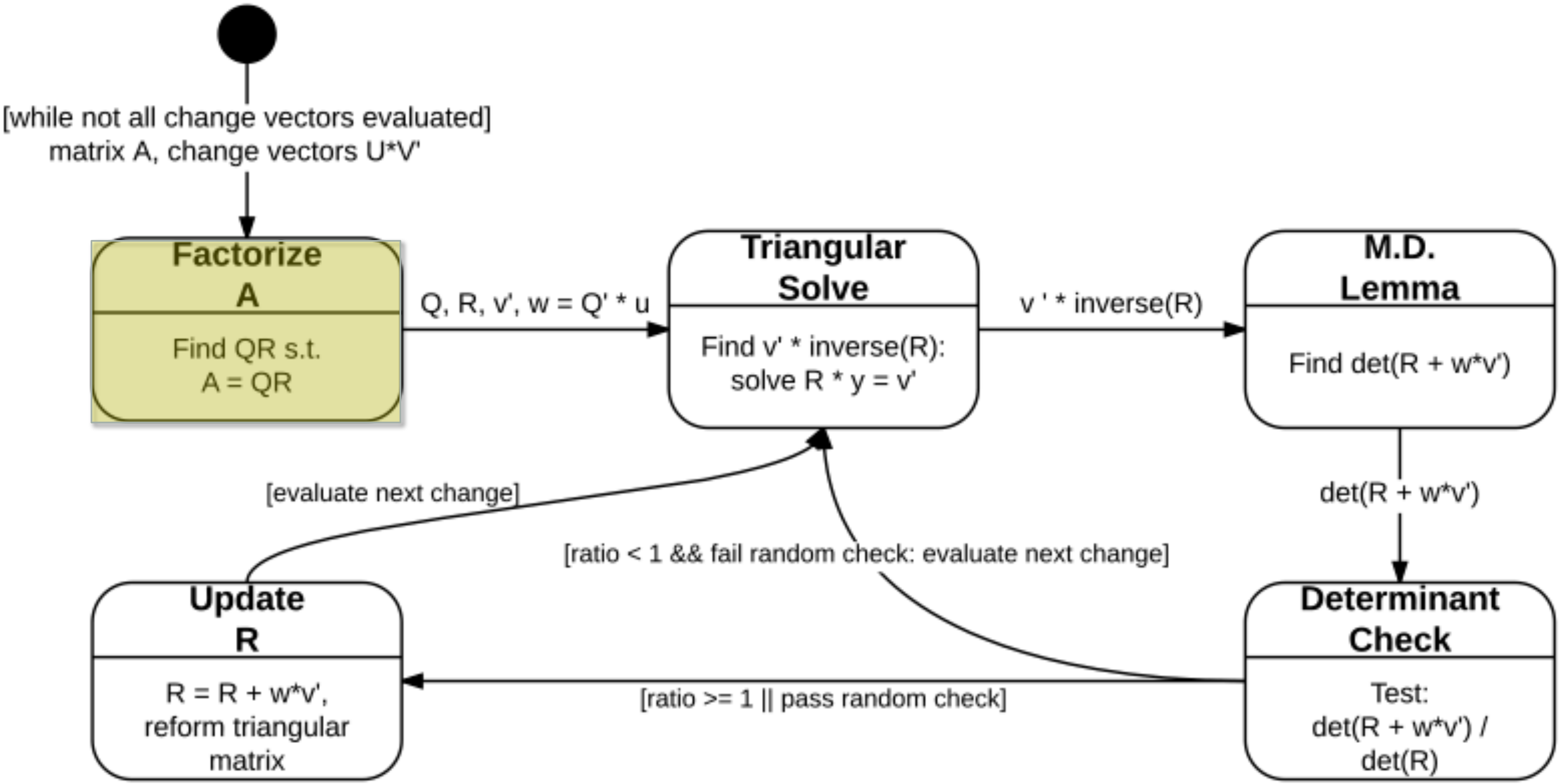
Current QMCPACK implementation



Proposed Implementation

- Using QR factorization versus LU factorization
- Rank-k update versus Rank-1 update
- Triangular solve versus Sherman Morrison formula

Proposed Implementation



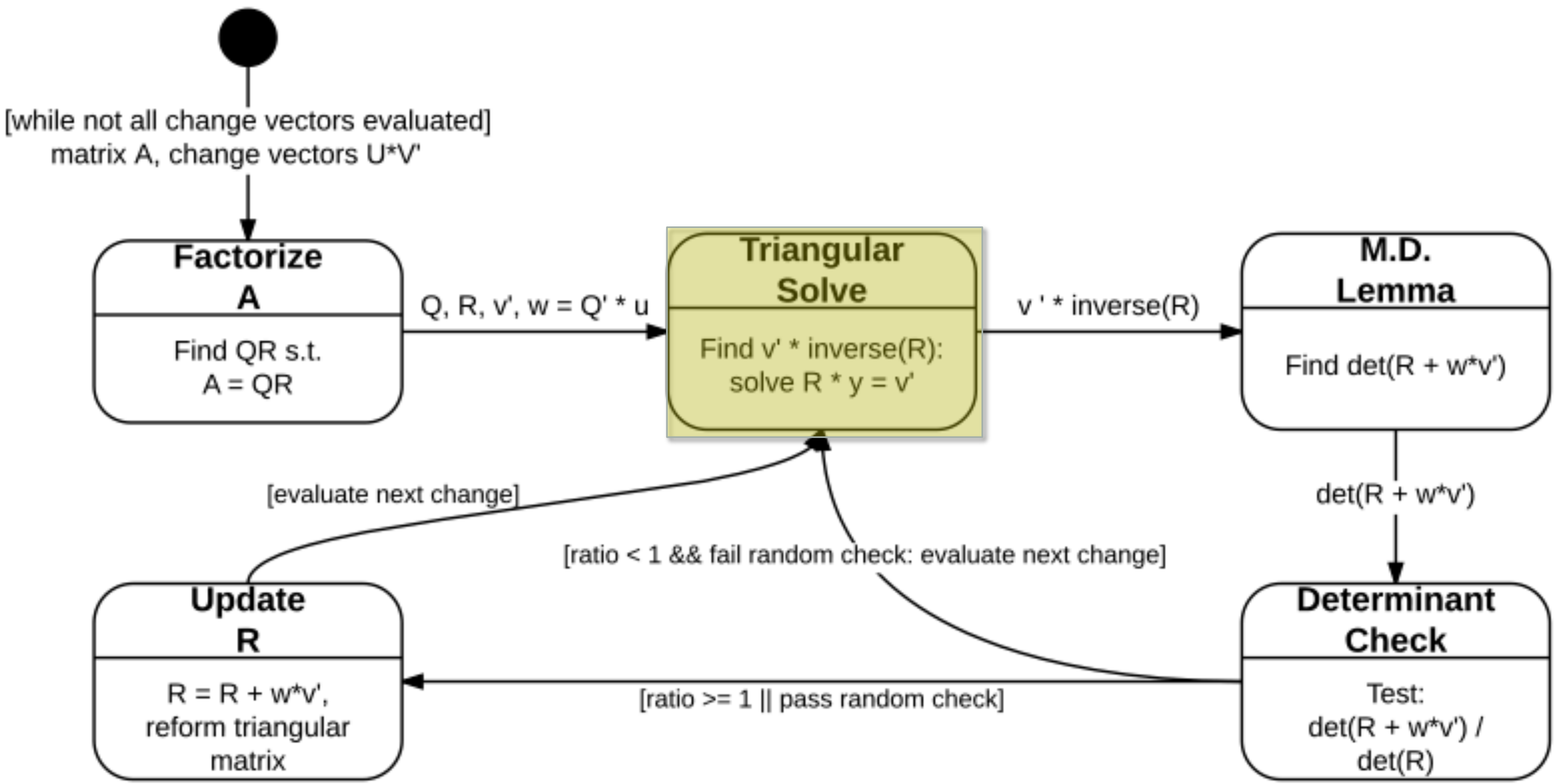
QR Decomposition

$$A = QR,$$

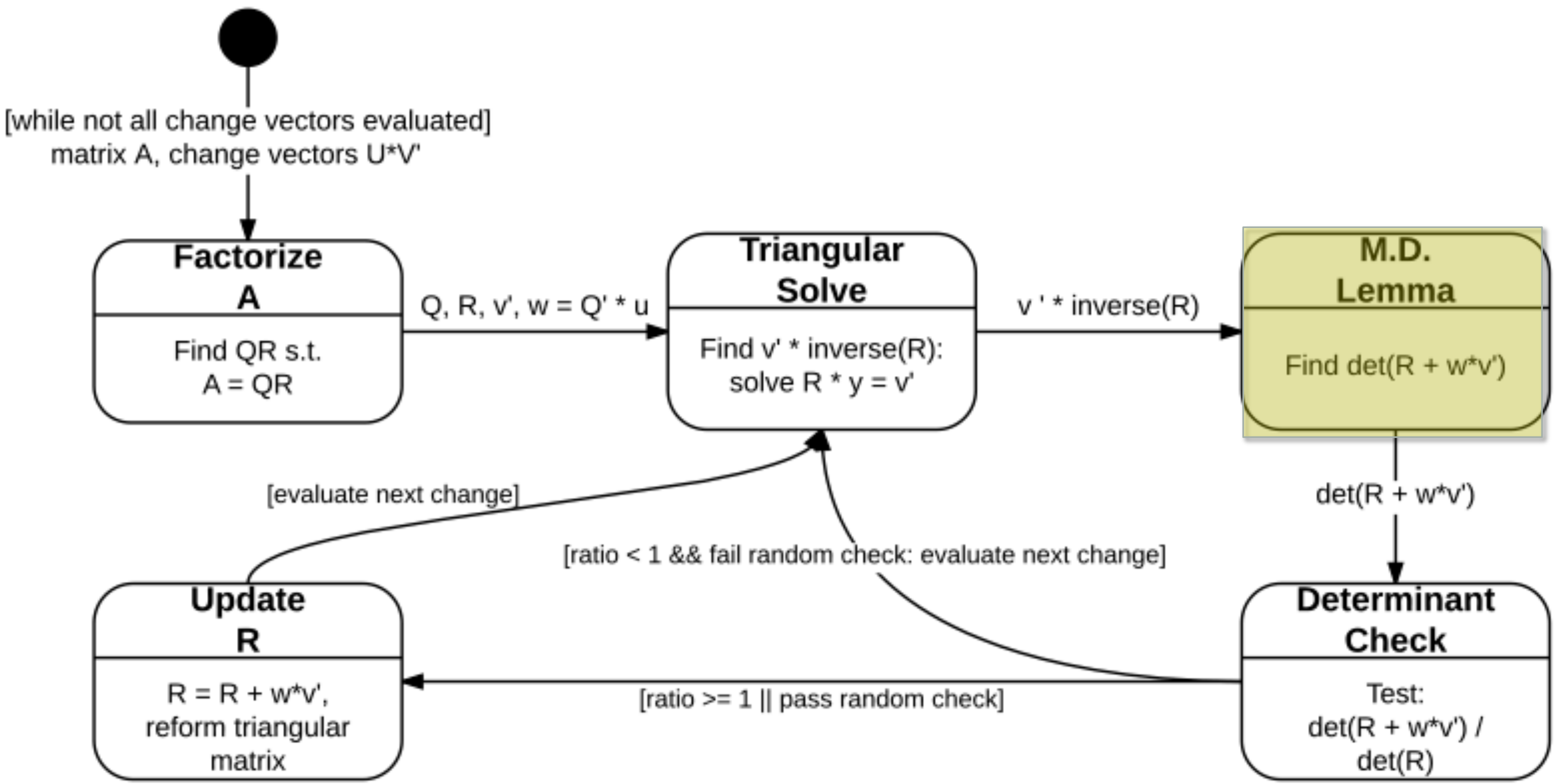
$$\begin{pmatrix} q_1 & \cdots & q_n \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & 0 & r_{nn} \end{pmatrix}$$

Note that Q is orthonormal and R is upper triangular.

Proposed Implementation



Proposed Implementation



Matrix Determinant Lemma

Rank-1 case (note that A is R in our scenario)

Suppose A is an invertible square matrix and u, v are column vectors.

$$\det(\mathbf{A} + \mathbf{u}\mathbf{v}^{\mathrm{T}}) = (1 + \mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{u})\,\det(\mathbf{A}).$$

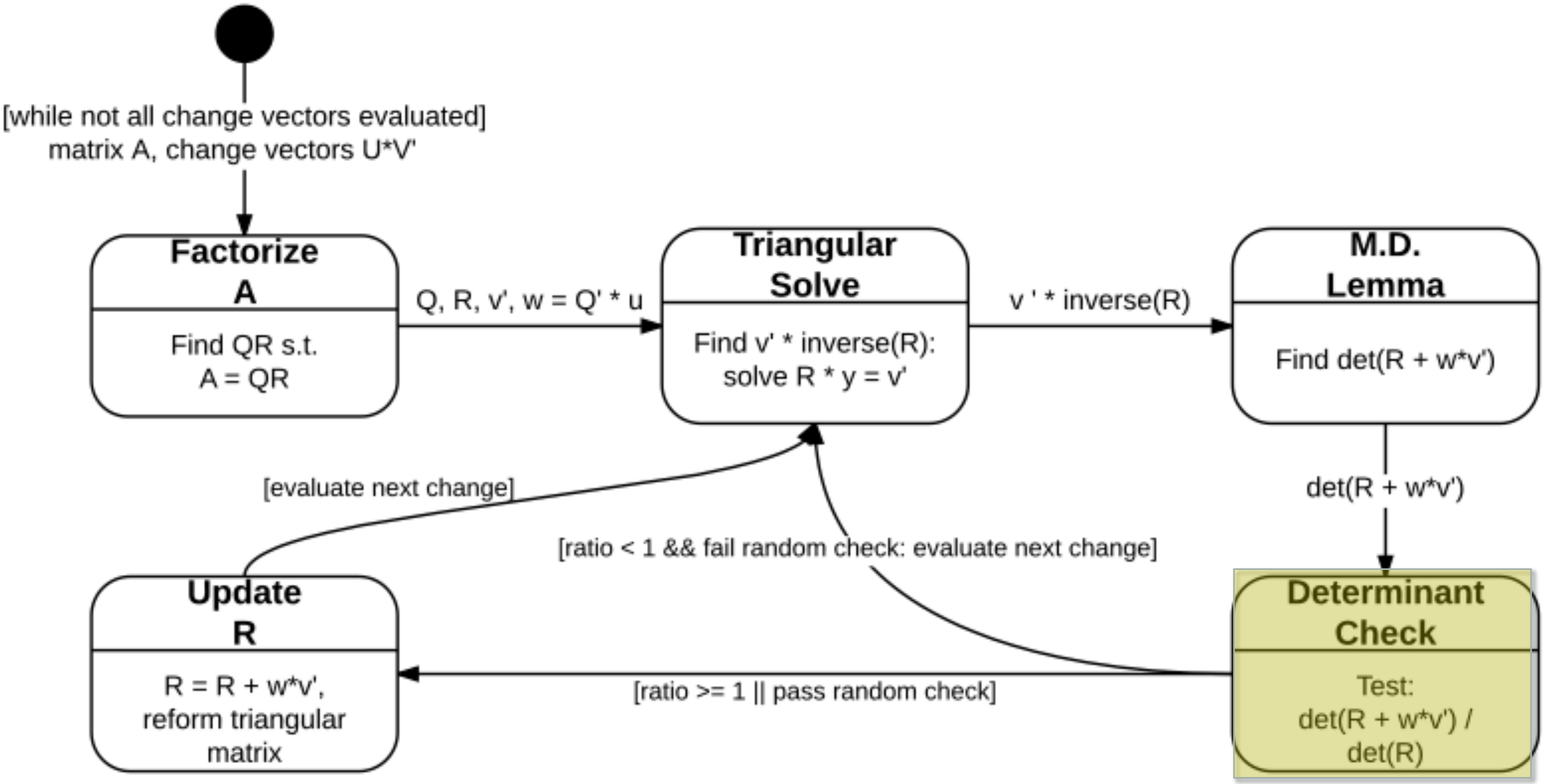
Rank-k case

Suppose A is an invertible n-by-n matrix and U, V are n-by-m matrices.

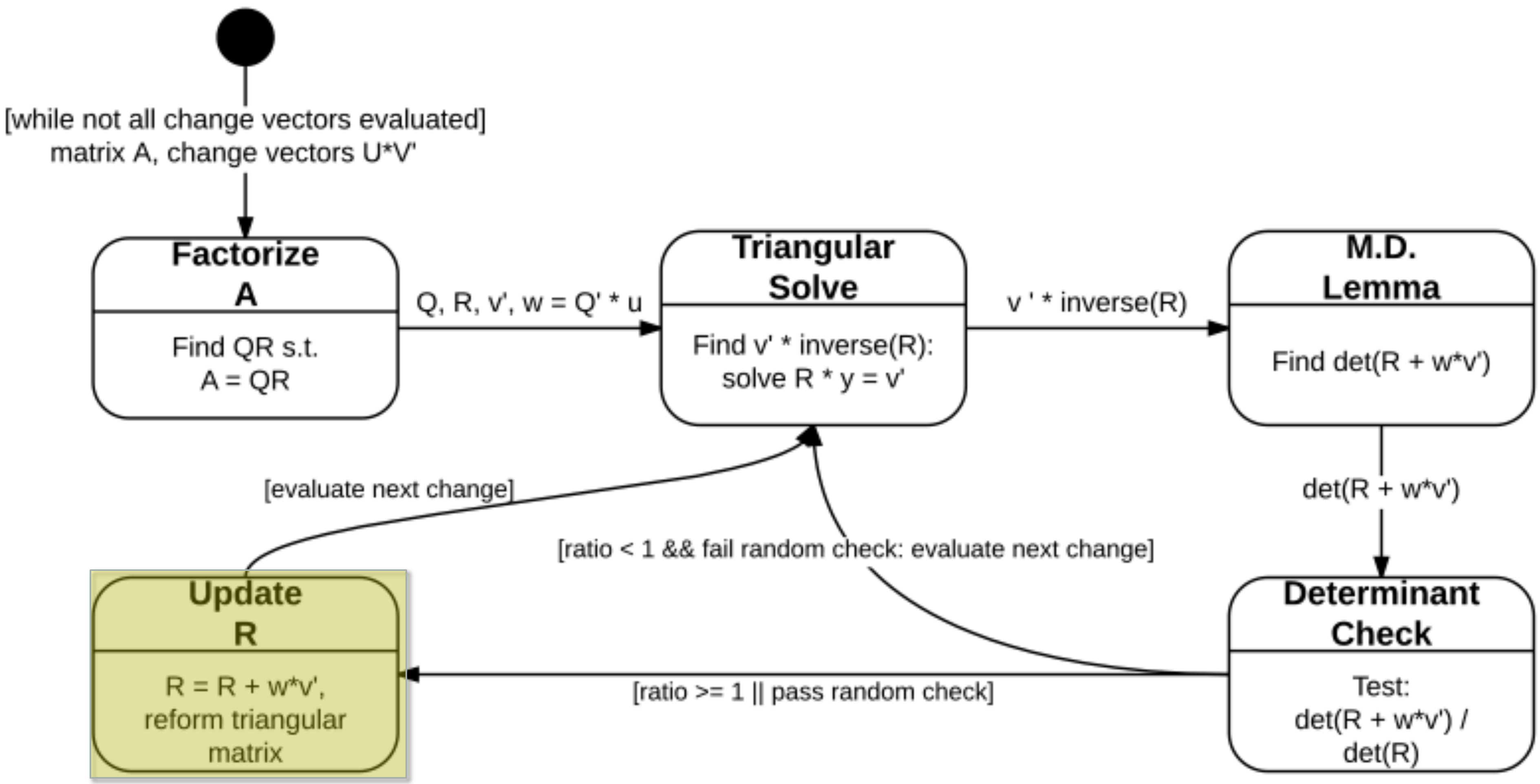
$$\det(\mathbf{A} + \mathbf{U}\mathbf{V}^{\mathrm{T}}) = \det(\mathbf{I_m} + \mathbf{V}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{U})\,\det(\mathbf{A}).$$

- det(R + u*v’)
  = det(R * (eye + (inv(R)*u) * v’))
  = det(R) * det(eye + w * v’)

Proposed Implementation



Proposed Implementation



Givens Rotation

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad r = \sqrt{a^2 + b^2}.$$

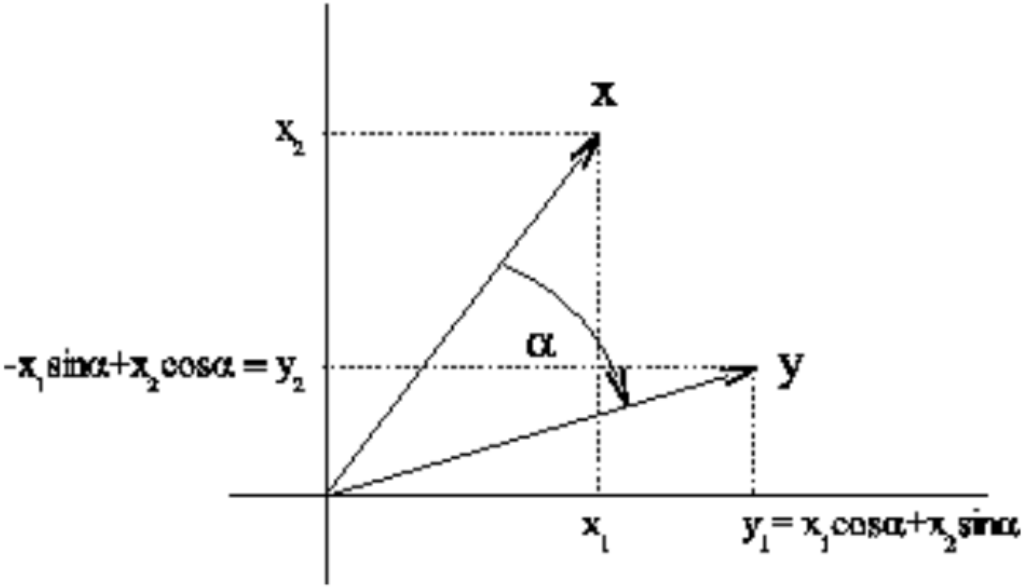
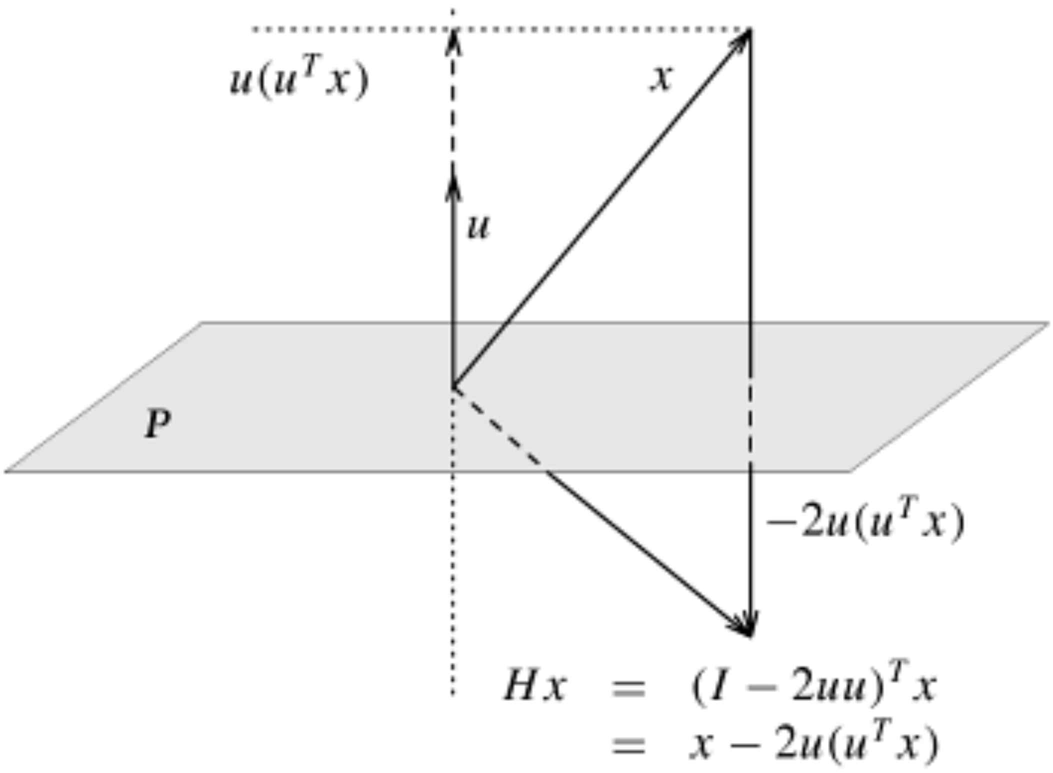


Figure 4.3: Rotation of x in a plane by an angle α

Householder Reflection



Returns R to upper triangular

$$
\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}
\xrightarrow{(3,4)}
\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}
\xrightarrow{(2,3)}
\begin{bmatrix} \times & \times & \times \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times \end{bmatrix}
\xrightarrow{(1,2)}
$$

$$
\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \mathbf{0} & \boldsymbol{\times} & \boldsymbol{\times} \\ & \times & \times \\ & \times & \times \end{bmatrix}
\xrightarrow{(3,4)}
\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \end{bmatrix}
\xrightarrow{(2,3)}
\begin{bmatrix} \times & \times & \times \\ & \boldsymbol{\times} & \boldsymbol{\times} \\ & \mathbf{0} & \boldsymbol{\times} \\ & & \times \end{bmatrix}
\xrightarrow{(3,4)} R
$$

- The updated column of R will be “shifted” at the nth column, where n is the size of the square matrix A.
- Utilizing the above techniques, zeros can be introduced below the diagonal in columns of R.
- Appropriate operations are performed on Q to maintain A = QR

Implementation Timeframe: 10 weeks

1) MATLAB
   - Establish basic algorithm execution flow in MATLAB
   - ~ 2 weeks
2) C (MKL/LAPACKE)
   - Translate into BLAS/LAPACKE
   - ~ 2 weeks
3) C, accelerated (cuBLAS)
   - Practice GPU mem. management, invoke cuBLAS from C
   - ~ 1 week
4) CUDA
   - Move execution to GPU
   - ~ 2 weeks

Results

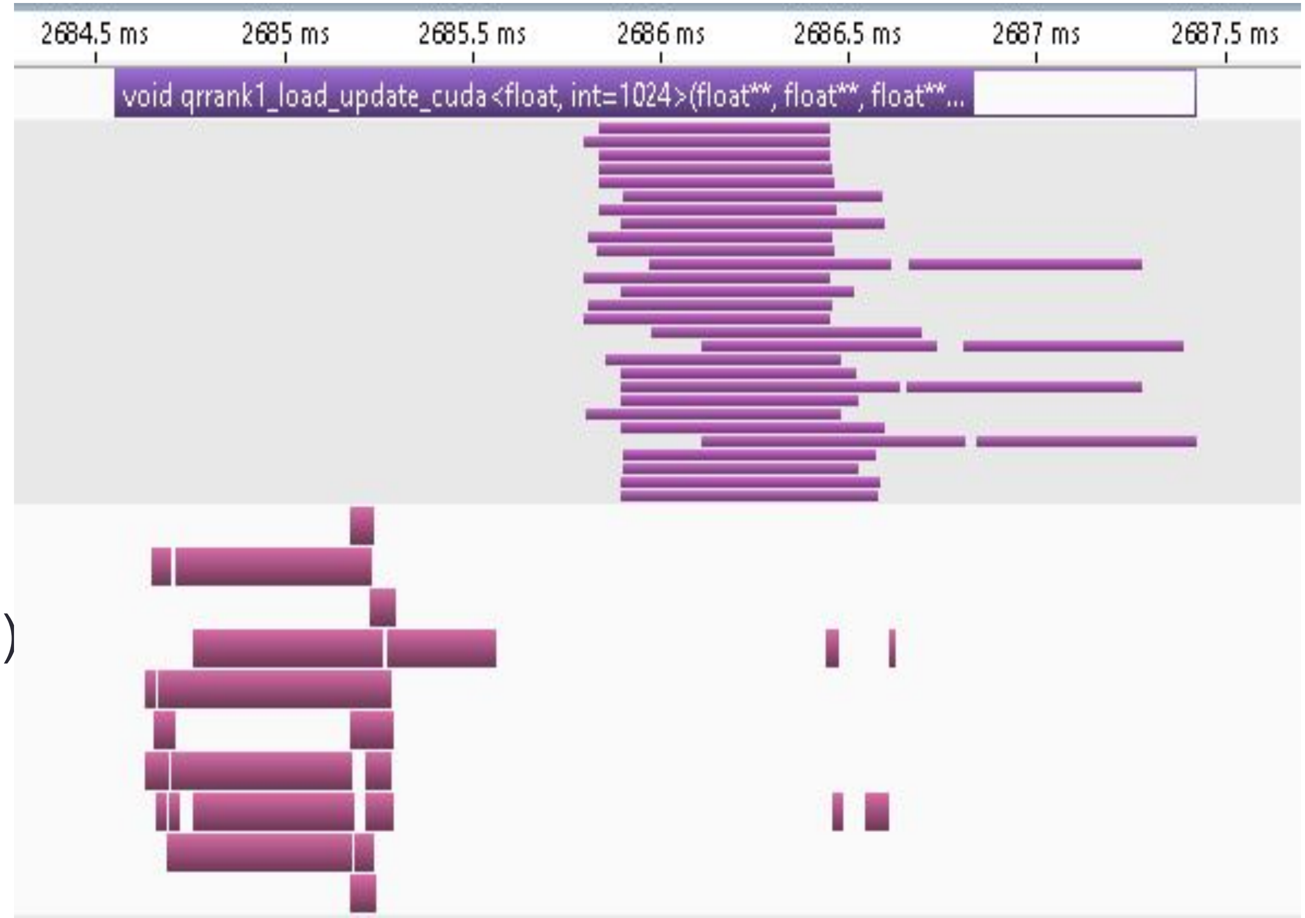
- ➢Implemented the algorithm in CUDA
- ➢Used dynamic parallelism and cuBLAS

Kernel 1:
Estimate
determinant delta

Operation 1:
GEMV
w = Qt * u

Operation 2:
TRSV (Child kernel)
y = R * w

Result:
delta = y[k] + 1



Results

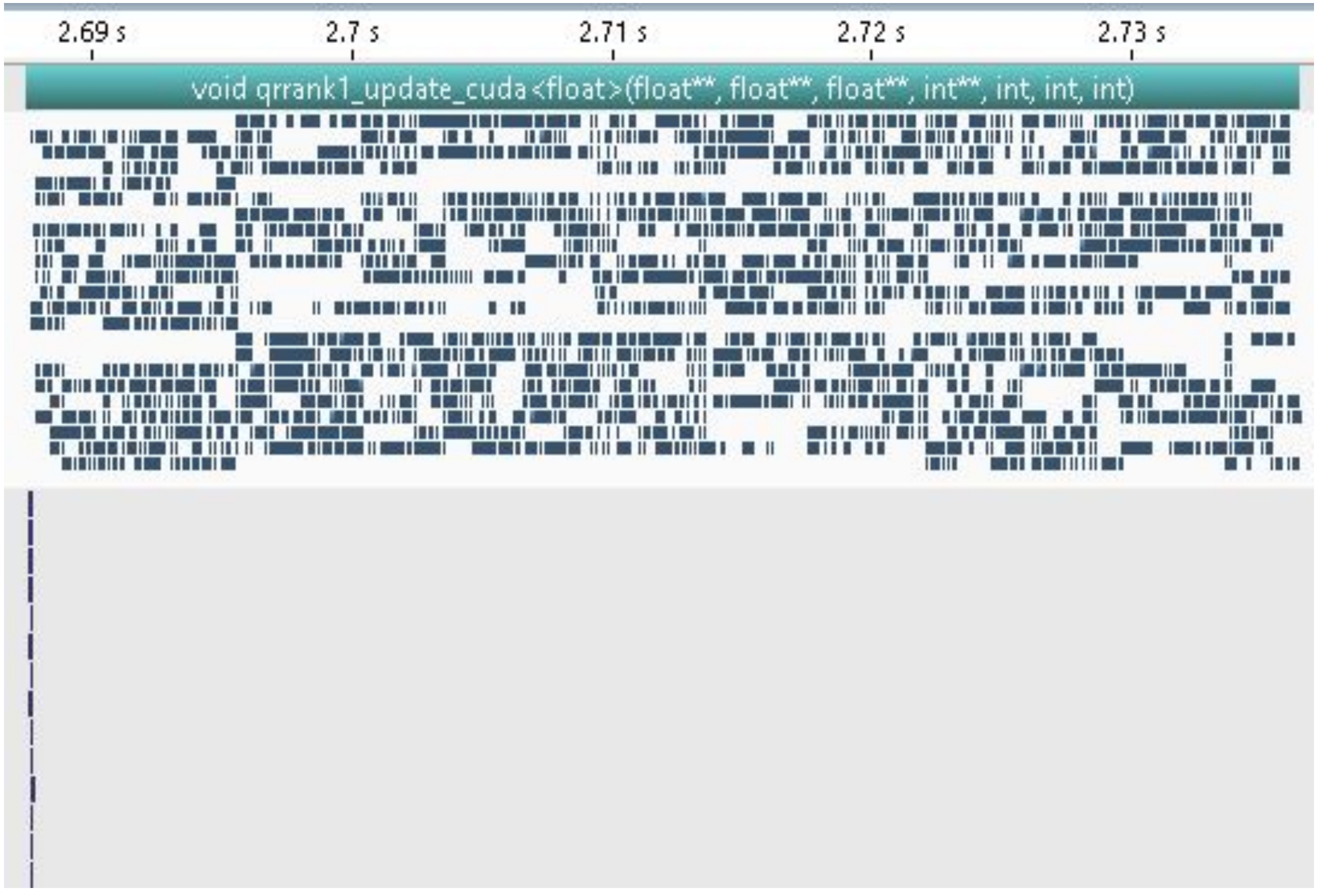
- ➢Implemented the algorithm in CUDA
- ➢Used dynamic parallelism and cuBLAS

Kernel 2:
Update R

Operation 1:
AXPY
R = R + w * transpose(v)

Operation 2:
ROTG/ROT
(Iterative)

Result:
Updated R



Results

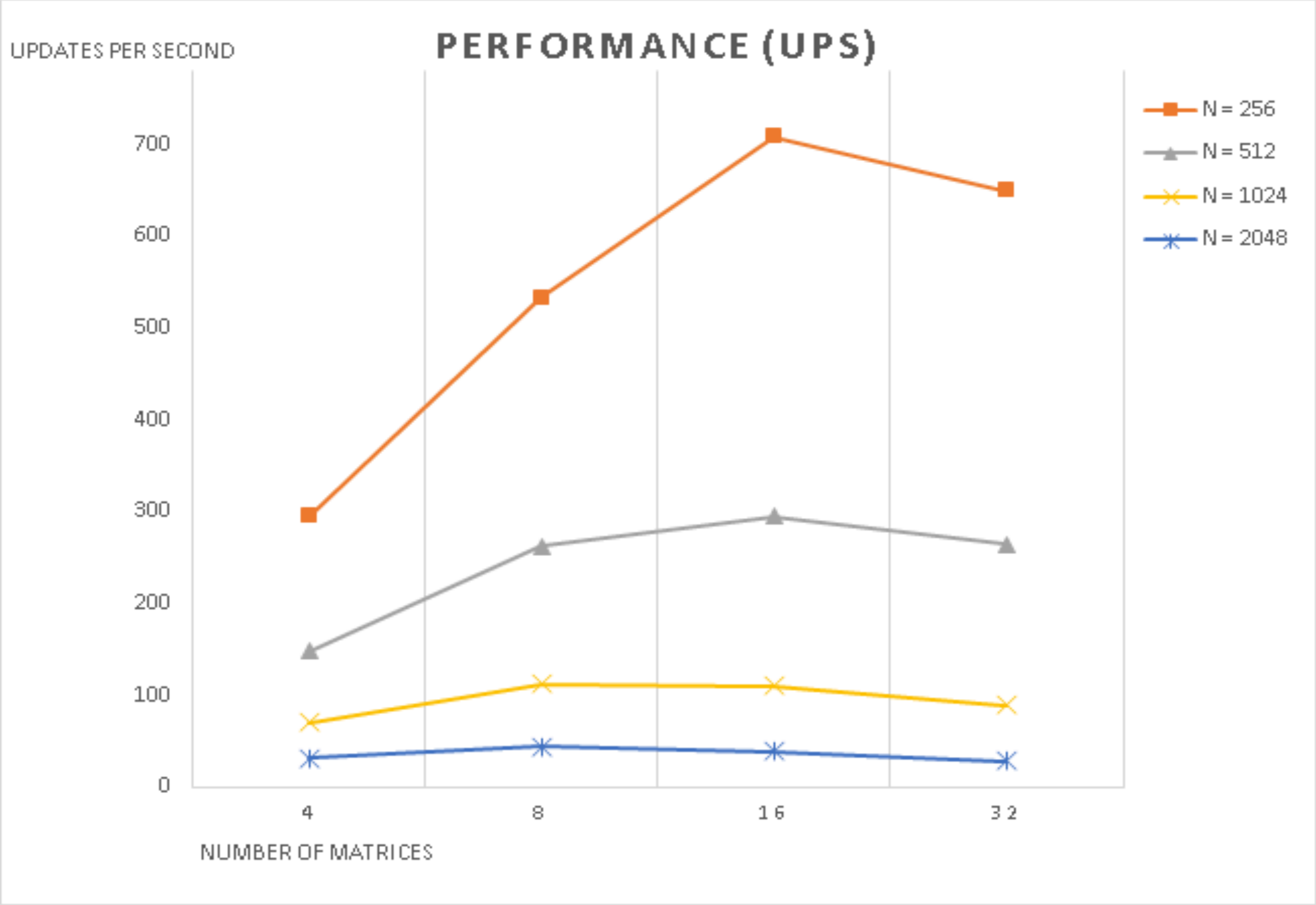
- Test platform: Beacon GPU node
  - equipped with 4x Tesla K20Xm GPUs; used 1 GPU

[0] Tesla K20Xm

Compute Capability	3.5
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[2147483647, 65535, 65535]
Max. Block Dimensions	[1024, 1024, 64]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	16
Number of Multiprocessors	14
Multiprocessor Clock Rate	732 MHz
Concurrent Kernel	true
Max IPC	7
Threads per Warp	32
Global Memory Bandwidth	249.6 GB/s
Global Memory Size	5.625 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1.5 MiB
Memcpy Engines	2
PCIe Generation	2
PCIe Link Rate	5 Gbit/s

Results

- GPU RAM: ~sizeof(float /double) * num_mats * $(2n^2+2n)$
- Flops per update (combined) $15n^2$



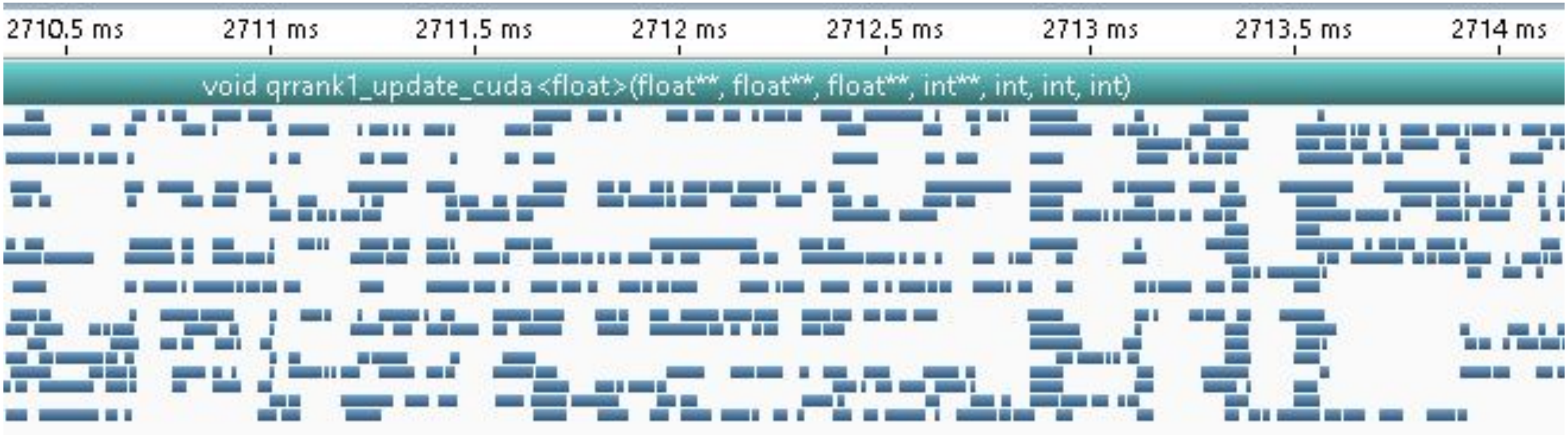
Greatest performance at N < 256:

5,000+ updates per second

However, small matrices not relevant to our use case

Discussion (Parallelism)

- Sequential Givens rotations limit scalability
  - Level 1 BLAS calls account for majority of kernel runtime
  - Control flow cost greater than compute cost



Discussion (Parallelism)

- Strategy: Adapt existing parallel implementations of Givens QR (e.g., those based on Sameh and Kuck, 1978) or Householder QR

- Some implementations require just ~5/8 of computational steps vs. sequential algorithms (Kontoghiorghes, 2002 p. 1266)
  - Effect: Decrease time cost of reforming triangular R, decrease execution gaps
  - Cost: Far more complex to implement

Discussion (Parallelism)

$$R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \qquad w = \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

Column permutations (used to reduce transformations required)

$$R = J_3^T R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \qquad w = J_3^T w = \begin{bmatrix} \times \\ \times \\ \times \\ 0 \end{bmatrix}$$

$$R = J_2^T R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \qquad w = J_2^T w = \begin{bmatrix} \times \\ \times \\ 0 \\ 0 \end{bmatrix}$$

$$H = J_1^T R = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \qquad w = J_1^T w = \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Discussion (Parallelism)

- Strategy: Replace column permutation with norm-preserving change vector rotations

  - Patterned on Golub and Van Loan, 1996 p. 606-607

  - Effect: Reduced complexity
    - R is always upper triangular (in memory)
    - Runtime variability is reduced

  - Cost: Increased flops

Discussion (Parallelism)

- Rank-1 change is evaluated
- Applied immediately if accepted
- Contiguous accepted changes not grouped

Discussion (Parallelism)

- Strategy: Generalize implementation for rank-k column update;
  - Evaluate change submatrix
  - Apply changes to R only after contiguous acceptance pattern is broken
- Effect: Leverage likely acceptance pattern
  - Perform block operations
- Cost: More complex to implement
  - May require extensive modification to QMCPACK

Discussion (CUDA)

- Improved cuBLAS Management:
  - Share cuBLAS handles between synchronized kernels to minimize overhead
  - "*...the recommended programming model is to create one CUBLAS handle per thread and use that CUBLAS handle for the entire life of the thread.*"

    ~CUDA Toolkit 6.5 Documentation: cuBLAS
  - Use cuBLAS streams to increase occupancy
  - Up to 16 concurrent kernels are supported (hardware dependent)

Discussion (CUDA)

- Decrease Memory Latency
  - Currently, kernels are heavily latency-bound (limited by memory access, not computation)
  - Reduce level of pointer indirection

Works Cited

- Andrew, Robert, and Nicholas Dingle. "Implementing QR Factorization Updating Algorithms on GPUs." Parallel Computing 40.7 (2014): 161-72. Web. 4 Aug. 2015. <http://www.sciencedirect.com/science/article/pii/S0167819114000337>.
- "CuBLAS :: CUDA Toolkit Documentation." CuBLAS :: CUDA Toolkit Documentation. Web. 4 Aug. 2015.
- Golub, Gene H., and Charles F. Loan. Matrix Computations. 3rd ed. Baltimore: Johns Hopkins UP, 1996. Print.
- Kontoghiorghes, Erricos J. "Parallel Strategies for Rank- K Updating of the QR Decomposition." SIAM. J. Matrix Anal. & Appl. SIAM Journal on Matrix Analysis and Applications 23.3 (2000): 714-25. Web. 4 Aug. 2015. <http://epubs.siam.org/doi/pdf/10.1137/S0895479896308585>.
- Kontoghiorghes, Erricos John. "Greedy Givens Algorithms for Computing the Rank-k Updating of the QR Decomposition." Parallel Computing 28 (2002): 1257-273. Web. 4 Aug. 2015. <http://www.dcs.bbk.ac.uk/~matrix/Papers/ErricosRankk.pdf>.
- Padua, David A. Encyclopedia of Parallel Computing. Vol. 4. New York: Springer, 2011. Print.
- Sameh, A. H., and D. J. Kuck. "On Stable Parallel Linear System Solvers." Journal of the ACM JACM J. ACM 25.1 (1978): 81-91. Web. 4 Aug. 2015. <http://dl.acm.org/citation.cfm?id=322054>.
- Volkov, V., and J.w. Demmel. "Benchmarking GPUs to Tune Dense Linear Algebra." 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (2008). Web. 4 Aug. 2015. <http://mc.stanford.edu/cgi-bin/images/6/65/SC08_Volkov_GPU.pdf>.

Acknowledgements

- We greatly appreciate help from our mentors:
  - Dr. Ed D'Azevedo from ORNL
  - Dr. Ying Wai Li from ORNL
  - Dr. Kwai Wong from UTK

  - NSF
  - ORNL
  - UTK

Exploring QR Factorization on GPU for Quantum Monte Carlo Simulation

Tyler McDaniel
Ming Wong

Mentors: Ed D'Azevedo, Ying Wai Li, Kwai Wong