Accelerating Linear Algebra Kernels on a Massively Parallel Reconfigurable Architecture

Anuraag Soorishetty*, Jian Zhou*, Subhankar Pal#, David Blaauw#, Hun Seok Kim#, Trevor Mudge#, Ronald Dreslinski#, Chaitali Chakrabarti*

*Arizona State University, Tempe, AZ
#University of Michigan, Ann Arbor, MI

Problem Definition

- Linear Algebra (LA) kernels form bottlenecks in many real-time applications including scientific computing, statistics and machine learning.

- This paper demonstrates acceleration of few key LA kernels onto a reconfigurable multi-core architecture, Transformer.

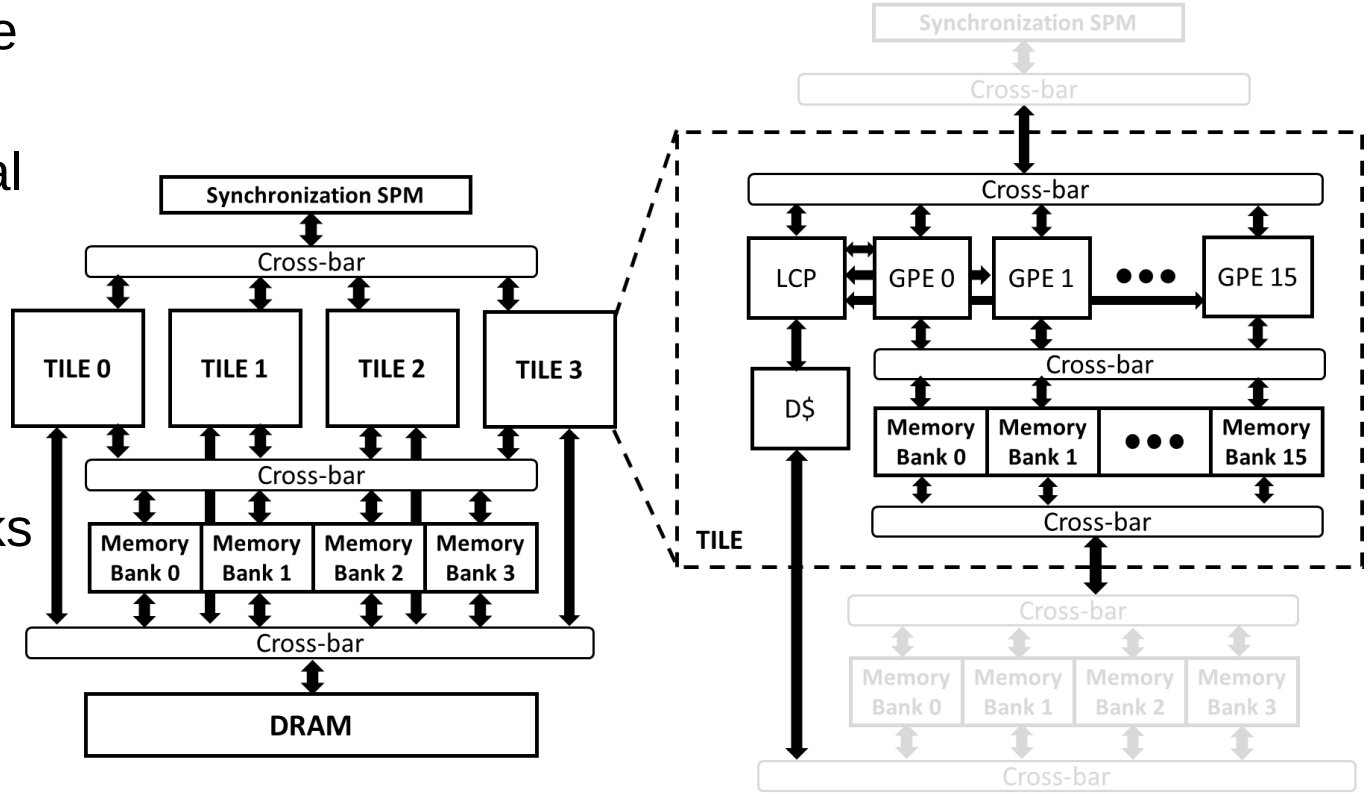
- LA Kernels Studied:
  - Triangular Matrix Solver (TRSM)
  - LU Decomposition (LUD)
  - QR Decomposition (QRD)
  - Matrix Inversion

Existing Hardware Solutions

- Many domain-specific architecture solutions have been designed to accelerate LA kernels. Some of them include,
  - ASIC: Lacore[1], QRD in MIMO receivers[2]
  - Systolic Arrays: Matrix Multiplication[3], Triangularization[4]
  - GPU: CULA[5], Alinea[6], Dense Linear Algebra Solvers[7]
  - FPGA: Linear Algebra in Adaptive Control Algorithms[8], Matrix-Multiplication on Virtex-7[9]
  - CGRA: REDEFINE[10], ADRES[11], DySER[12], LAC[13], PLASTICINE[14]

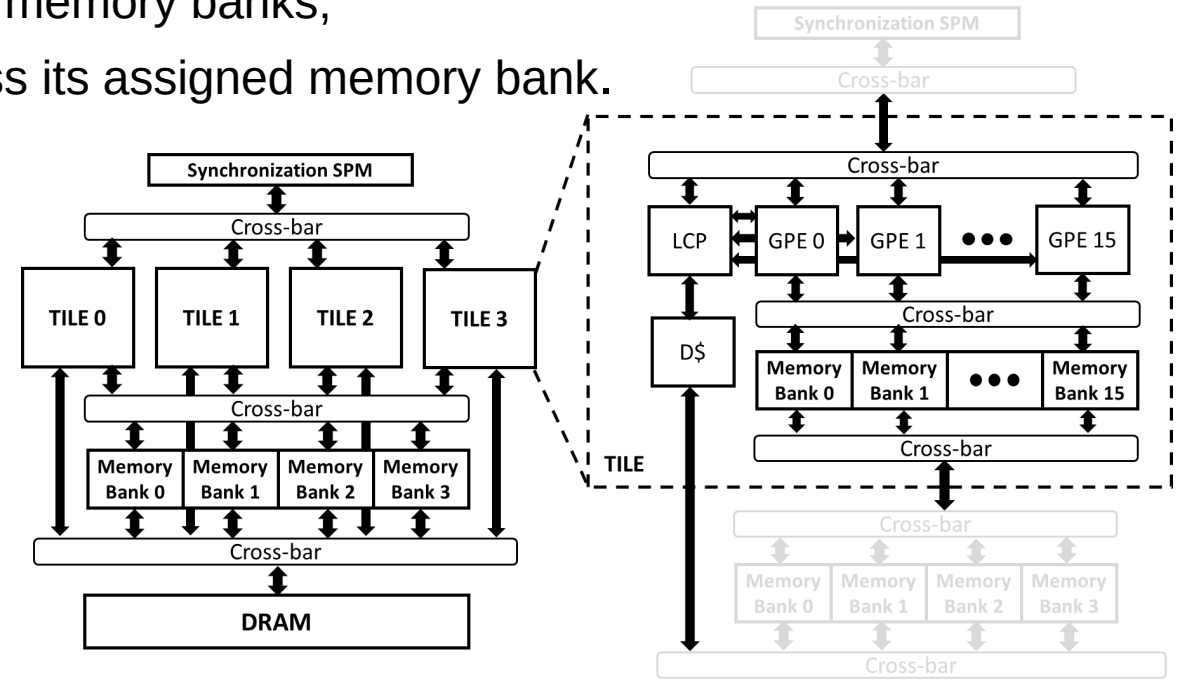
Transformer - I

- Transformer is a scalable, energy-efficient, reconfigurable multicore architecture with distributed on-chip memories, crossbars and a high-bandwidth DDR interface.
- m tiles with n GPEs (General-Purpose Processing Elements) per tile.
- GPEs are managed by the LCP (Local Control Processor).
- Two layer cache-crossbar hierarchy
  - L1: in-tile (within GPEs)
  - L2: out-of-tile (across LCPs)
- Crossbars are swizzle-switch networks which are scalable and energy-efficient.



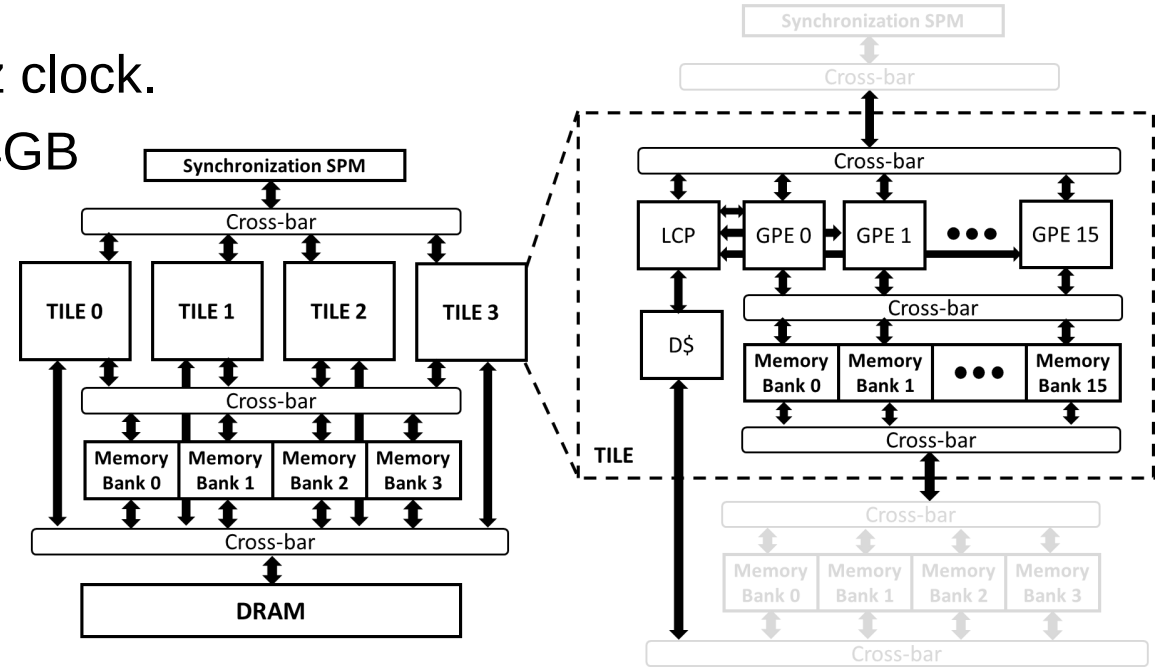
Transformer - II

- Transformer supports reconfiguration with different cache modes; reconfiguration costs only one cycle.
- Cross-bar connects GPE with memory banks in different modes:
  - **Shared mode (S)**: Each GPE can access all memory banks;
  - **Private mode (P)**: Each GPE can only access its assigned memory bank.
- A global Scratchpad Memory (SPM) can be accessed by all GPEs and LCPs.
- It is used for implementing software coherence and standard primitives such as locks, condition variables, barriers and semaphores.



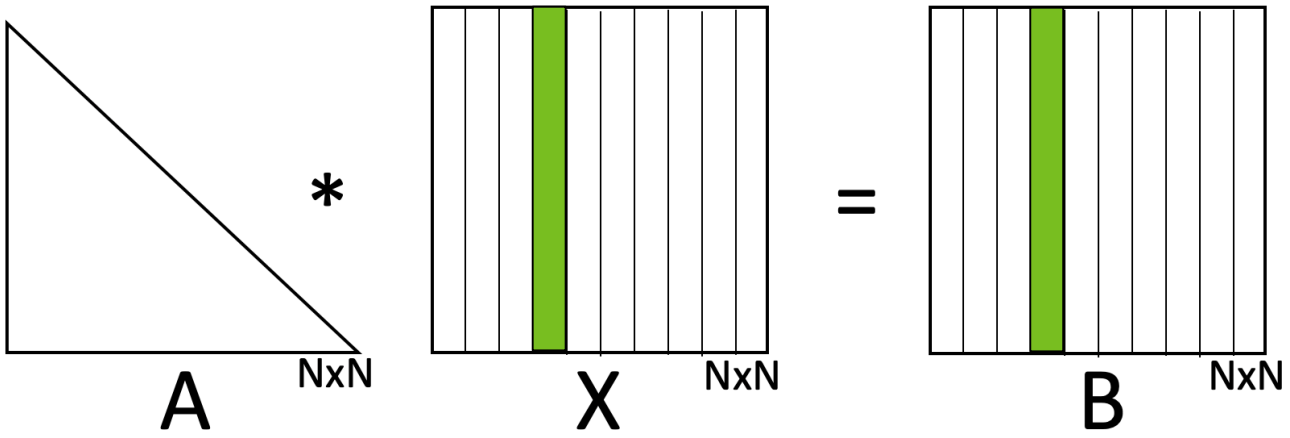
Transformer - Configuration

- Modeled using Gem5 architectural simulator.
- 4 tiles and 16 GPEs per tile running on 1GHz clock.
- L1: 4kB per GPE, L2: 64kB per tile, DRAM: 4GB
- Programmable using C/C++
- Cache configurations:
  - L1 Shared, L2 Shared (L1S, L2S)
  - L1 Shared, L2 Private (L1S, L2P)
  - L1 Private, L2 Shared (L1P, L2S)
  - L1 Private, L2 Private (L1P, L2P)
- Power model
  - ARM cores: Validated against a prototype chip (40nm)[15] and scaled down to 14nm.
  - Reconfigurable caches: Generated using CACTI model[16] for 14nm node and Gem5 stats file.
  - Crossbars: Obtained from Sewell et al.[17], scaled from 32nm to 14nm.



Triangular Matrix Solver (TRSM)

- Solves a system of linear equations of the form AX=B, where A is an upper or lower triangular matrix, and X & B are dense matrices.



Forward Substitution

Each GPE performs:
(k: column)

$X_{1k} = B_{1k}/A_{11}$
for i = 2:N
  $s = B_{ik}$
  for j = 1:(i-1)
    $s = s - A_{ij}X_{jk}$
  $X_{ik} = s/A_{ii}$

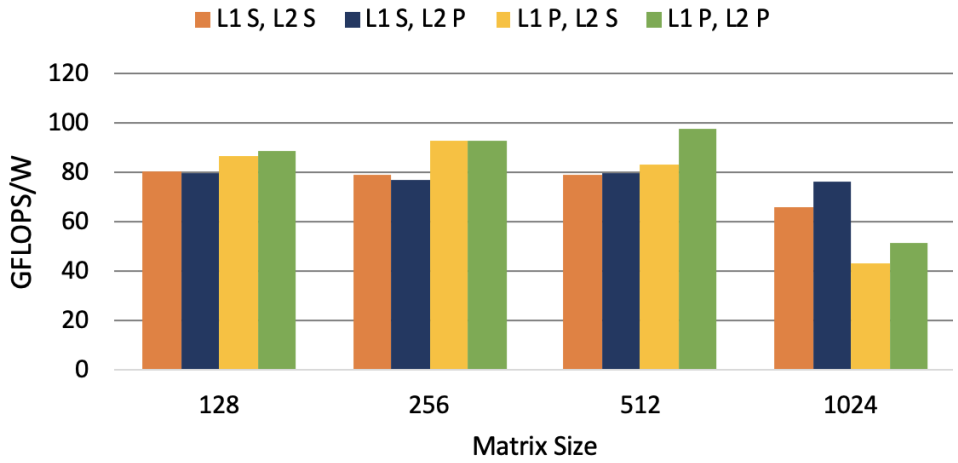
- Depending on whether A is an upper or lower triangular matrix, this algorithm employs backward or forward substitution.
- Columns of X can be solved independently using columns of B but each column has its own serial computational dependency.

TRSM – Peak 97.5 GFLOPS/W

- Each GPE is assigned the task of computing one or more columns of X.
- A column of X is solved one-by-one and stored in L1. After the entire column is solved, the values are flushed to DRAM through L2.

Execution Time (ms)

NxN	L1S, L2S	L1S, L2P	L1P, L2S	L1P, L2P
128	0.15	0.15	0.137	**0.132**
256	1.28	1.33	0.989	**0.981**
512	10.33	10.12	9.32	**7.22**
1024	104.55	**85.4**	178.61	143.7



- For small matrix sizes, L1P, L2P has the best performance; but does not perform well for large sizes due to insufficient L1 cache bank.
- For larger matrix sizes (1024x1024) L1S, L2P does better.

LU Decomposition (LUD)

- Factorizing a square matrix A into a product of lower triangular matrix, L and an upper triangular matrix, U, given by A=LU.

LUD v1

```
for k = 1:N
  for j = k:N
    L_jk = A_jk / A_kk
  for j = k+1:N
    for i = k+1:N
      A_ij = A_ij – L_ik * A_kj
```

LUD v2

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

LUD v1: $A_{11} = L_{11} U_{11}$

TRSM: $A_{21} = L_{21} U_{11}$

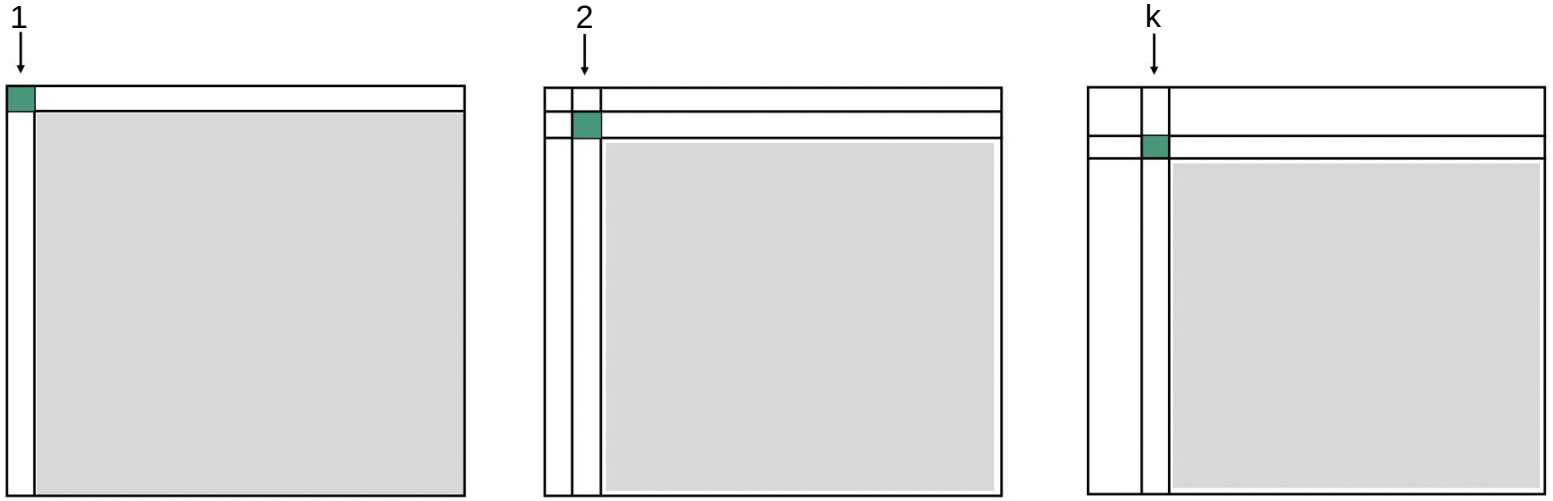
TRSM: $A_{12} = L_{11} U_{12}$

GEMM and LUD: $A_{22} - L_{21} U_{12} = L_{22} U_{22}$

- LUD has serial dependency within each column and across columns.
- LUD v1 is computed by Gaussian elimination where U overwrites A; L is stored separately.
- LUD v2 is computed by dividing the matrix into blocks and solving using a combination of LUD v1, GEMM and TRSM.

LUD v1 - Mapping

- LUD v1:
  - One or more rows assigned to each GPE per column-update.
  - The updated values stay in L1 and are flushed to DRAM after every column-update.
  - GPEs assigned to rows above the pivot row stay idle - very low utilization.
  - Tile 0 becomes inactive after N/4 column-updates.



LUD v1

```
for k = 1:N
  for j = k:N
    L_jk = A_jk / A_kk
  for j = k+1:N
    for i = k+1:N
      A_ij = A_ij – L_ik * A_kj
```

LUD v2 – Mapping

- LUD v2:
  - Blocked approach solved using LUD v1, TRSM and GEMM.
  - LUD v1 here works on a smaller block. So the imbalance in workload distribution is not much.
  - GEMM is performed by dividing matrix into blocks of 16 and assigning to GPEs.
  - Better utilization of GPEs compared to LUD v1.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

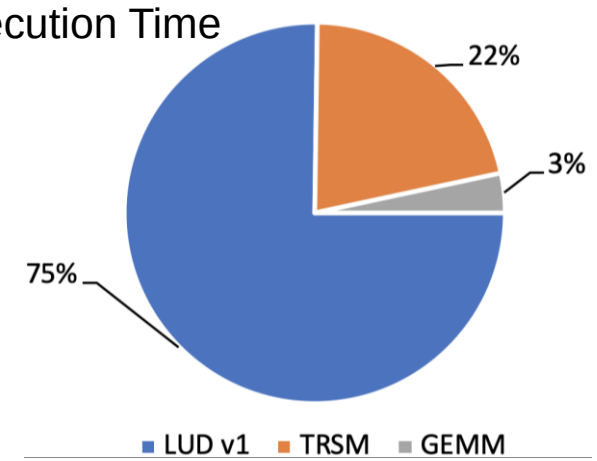
$$\begin{aligned} \text{LUD v1:} \quad & A_{11} = L_{11}\, U_{11} \\ \text{TRSM:} \quad & A_{21} = L_{21}\, U_{11} \\ \text{TRSM:} \quad & A_{12} = L_{11}\, U_{12} \\ \text{GEMM and LUD:} \quad & A_{22} - L_{21}\, U_{12} = L_{22}\, U_{22} \end{aligned}$$

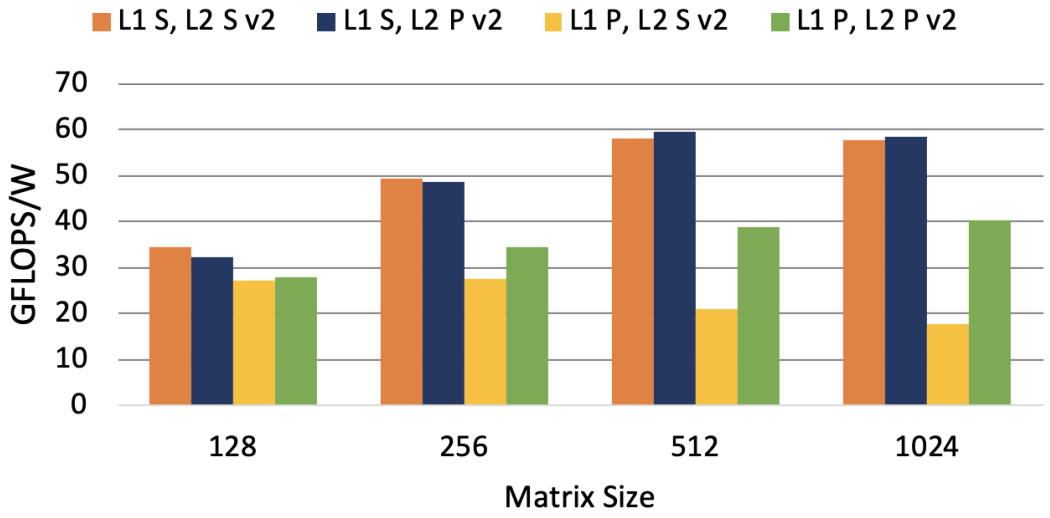
LUD v2 – Peak 59 GFLOPS/W

Execution Time (ms)

NxN	L1S, L2S		L1S, L2P		L1P, L2S		L1P, L2P	
	v1	v2	v1	v2	v1	v2	v1	v2
128	0.67	**0.46**	0.69	0.5	0.69	0.57	0.67	0.55
256	3.58	**2.06**	3.44	2.1	3.52	3.78	3.64	2.9
512	25.61	12.96	25.52	**12.62**	27.63	40.05	24.5	19.52
1024	168.66	99.6	169.62	**97.25**	371.25	382.6	157.17	143.77

Average Execution Time
For N=512





- LUD v2 outperforms LUD v1 for all matrix sizes and all cache modes except for L1P, L2S.
- For all matrix sizes, L1S, L2S/P performs well.

QR Decomposition (QRD)

- Factorizing a square or a non-square matrix A into a product of an orthogonal matrix, Q and an upper triangular matrix, R, given by A=QR using Givens rotation.

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad r = \sqrt{a^2 + b^2} \Rightarrow \begin{array}{l} c \leftarrow a/r \\ s \leftarrow -b/r \end{array}$$

```
for j = 1:N
   for i = m:-1:j+1
   [c,s] = Givens(A(i-1,j), A(i,j))
    A(i-1:i, j:N) = row.rot(A(i-1:i, j:N), c, s)
```

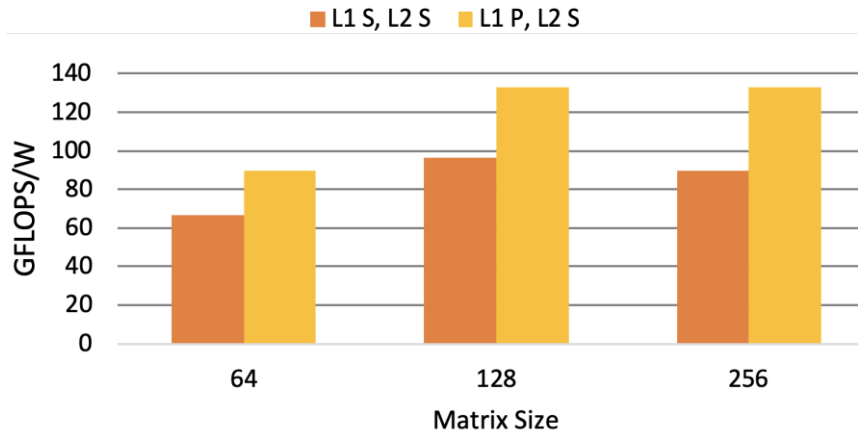
- Within a column, each element below the diagonal is annihilated from the last row and in reverse order.
- *row.rot* performs the Givens rotation of two adjacent rows.
- Multiplying *I* (Identity Matrix) with the Givens rotation matrices yields Q.

QRD – Peak 130 GFLOPS/W

- Annihilation of each column is assigned to a GPE.
- Every annihilation requires updating the entire row.
- The maximum parallelism is N/2 at cycle N-1.

Execution Time (ms)

NxN	L1S, L2S	L1P, L2S
64	0.5	0.35
128	2.44	1.54
256	22.88	13.3



- L1P, L2S works better for all matrix sizes as each GPE works independently on a column.

Matrix Inversion

- The inverse of a matrix is one which when multiplied by the original matrix A results in an identity matrix *I*, given by $AA^{-1} = A^{-1}A = I$, where A, A^{-1} and *I* are square matrices.

- Here, we use a combination of LUD and TRSM to compute A^{-1}. The steps are:

$$A = LU$$ LUD v2

$$LY = I$$ TRSM: Forward Substitution

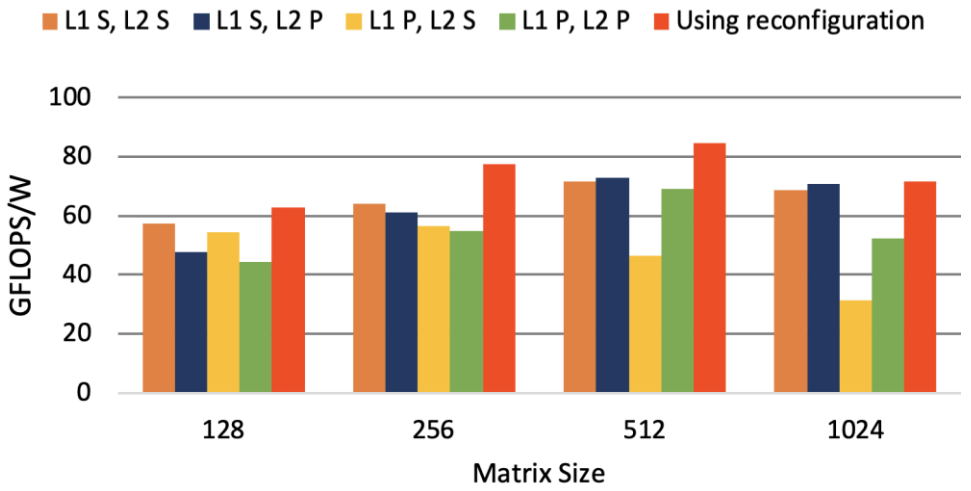
$$UX = Y$$ TRSM: Backward Substitution

Matrix Inversion – Peak 83.05 GFLOPS/W

- Matrix inversion using LUD v2, TRSM (forward sub) and TRSM (backward sub).
- LUD: 42.79%; forward sub: 29.96%; backward sub: 27.75% of total execution time.

Execution Time (ms)

NxN	L1S, L2S	L1S, L2P	L1P, L2S	L1P, L2P	Reconfig.
128	0.78	0.82	0.84	0.86	**0.72**
256	4.83	4.95	5.76	5.14	**3.9**
512	33.71	32.93	58.7	33.96	**27.32**
1024	279.66	**268.06**	739.82	385.22	**268.06**



- For example, for N=512
  - Reconfiguration helps increase GFLOPS/W from 72.67 (L1S, L2P) to 83.05.
  - Cache modes employed: LUD v2– L1S, L2P, TRSM – L1P, L2P

Conclusion

- Implemented several LA kernels on a reconfigurable multicore architecture, Transformer.
- Investigated performance for different kernels sizes and different L1 and L2 cache configurations (Shared and Private).
- Each kernel achieves high performance for a certain cache configuration and this cache configuration can change when the matrix size changes.
- Achieved a peak performance of 97.5, 59, 130.0 and 83.05 GFLOPS/W for TRSM, LUD, QRD and Matrix Inversion respectively.
- The reconfigurable cache features are utilized in the implementation of matrix inverse.

- Acknowledgement:

  The material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7864. The views and conclusions contained herein are those of the authors and do not represent the official policies or endorsements, either expressed or implied, of ARFL and DARPA or the U.S. Government.

References

[1] S. Steffl and S. Reda, “Lacore: A supercomputing-like linear algebra accelerator for soc-based designs,” in 2017 IEEE International Conference on Computer Design (ICCD), Nov 2017, pp. 137–144.

[2] D. Patel, M. Shabany, and P. G. Gulak, “A low complexity high-speed qr decomposition implementation for mimo receivers,” in 2009 IEEE International Symposium on Circuits and Systems, May 2009, pp. 33–36.

[3] Peddawad, S. Chaitanya and A. Goel. “Matrix-Matrix Multiplication Using Systolic Array Architecture in Bluespec Team”, 2015.

[4] H. T. Kung, and W. M. Gentleman, "Matrix triangularization by systolic arrays" (1982). Computer Science Department. Paper 1603.

[5] J. Humphrey, D. Price, K. Spagnoli, A. Paolini, and E. Kelmelis, “CULA: hybrid GPU accelerated linear algebra routines,” in Modeling and Simulation for Defense Systems and Applications V. International Society for Optics and Photonics, 2010, vol. 7705, pp. 9–15, SPIE.

[6] F. Magoules and A. Ahamed, “Alinea: An advanced linear algebra library for massively parallel computations on graphics processing units,” The International Journal of High Performance Computing Applications, vol. 29, no. 3, pp. 284–310, 2015.

[7] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, “Dense linear algebra solvers for multicore with gpu accelerators,” in 2010 IEEE International Symposium on Parallel Distributed Processing,Workshops and Phd Forum (IPDPSW), April 2010, pp. 1–8.

[8] F. A. Khan, R. A. Ashraf, Q. H. Abbasi, and A. A. Nasir, “Resource efficient parallel architectures for linear matrix algebra in real time adaptive control algorithms on reconfigurable logic,” in 2008 Second International Conference on Electrical Engineering, March 2008, pp. 1–9.

[9] W. Jos´e, A. R. Silva, H. Neto, and M. V´estias, “Analysis of matrix multiplication on high density virtex-7 fpga,” in 2013 23rd International Conference on Field programmable Logic and Applications, Sep. 2013, pp. 1–4.

[10] M. Alle, K. Varadarajan, A. Fell, N. Joseph, S. Das, P. Biswas, J. Chetia, A. Rao, SK. Nandy, R. Narayan, et al., “Redefine: Runtime reconfigurable polymorphic asic,” ACM Transactions on Embedded Computing Systems (TECS), vol. 9, no. 2, pp. 11, 2009.

References

[11] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, “Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix,” in International Conference on Field Programmable Logic and Applications, Springer, 2003, pp. 61–70.

[12] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, “Dyser: Unifying functionality and parallelism specialization for energy-efficient computing,” IEEE Micro, vol. 32, no. 5, pp. 38–51, 2012.

[13] A. Pedram, A. Gerstlauer, and R. A. Van De Geijn, “Floating point architecture extensions for optimized matrix factorization,” in 2013 IEEE 21st Symposium on Computer Arithmetic, 2013, pp. 49–58.

[14] A. Pedram, A. Gerstlauer, and R. A. Van De Geijn, “Algorithm, architecture, and floating-point unit codesign of a matrix factorization accelerator,” IEEE Transactions on Computers, vol. 63, no. 8, pp. 1854–1867, 2014.

[15] R. Balasubramonian, A. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, “Cacti 7: New tools for interconnect exploration in innovative off-chip memories,” ACM Trans. Archit. Code Optim., vol. 14, no. 2, pp. 14:1–14:25, June 2017.

[16] K. Sewell, R. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. Wenisch, D. Sylvester, et al., “Swizzle-switch networks for many-core systems,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 2, no. 2, pp. 278–294, 2012.

[17] S. Pal, D. Park, S. Feng, P. Gao, J. Tan, A. Rovinski, S. Xie, C. Zhao, A. Amarnath, T. Wesley, et al., “A 7.3 m output non-zeros/j sparse matrix-matrix multiplication accelerator using memory reconfiguration in 40 nm,” in 2019 Symposium on VLSI Technology. IEEE, 2019, pp. C150–C151.

Thank you!

Have a great day!