Sparse LU Decomposition using FPGA *

Jeremy Johnson[1], Timothy Chagnon[1], Petya Vachranukunkiet[2],
Prawat Nagvajara[2], and Chika Nwankpa[2]

CS[1] and ECE[2] Departments
Drexel University, Philadelphia, PA
jjohnson@cs.drexel.edu,tchagnon@drexel.edu,pv29@drexel.edu,
nagvajara@ece.drexel.edu,nwankpa@ece.drexel.edu

Abstract. This paper reports on an FPGA implementation of sparse LU decomposition. The resulting special purpose hardware is geared towards power system problems - load flow computation - which are typically solved iteratively using Newton Raphson. The key step in this process, which takes approximately 85% of the computation time, is the solution of sparse linear systems arising from the Jacobian matrices that occur in each iteration of Newton Raphson. Current state-of-the-art software packages, such as UMFPACK and SuperLU, running on general purpose processors perform suboptimally on these problems due to poor utilization of the floating point hardware (typically 1 to 4% efficiency). Our LU hardware, using a special purpose data path and cache, designed to keep the floating point hardware busy, achieves an efficiency of 60% and higher. This improved efficiency provides an order of magnitude speedup when compared to a software solution using UMFPACK running on general purpose processors.

Key words: FPGA, Sparse Linear Algebra, Power Systems

1 Introduction

The goal of this work is to investigate the implementation of application specific hardware on an FPGA designed to compute the lower/upper triangular (LU) decomposition of sparse matrices arising from the Load Flow calculation of power systems. Load flow is an essential part of power system market and contingency analysis where 85% of the compute time is devoted to solving sparse linear systems [1].

Sparse matrices are ubiquitous in scientific calculations when modeling systems with a large number of variables with limited coupling. In the electric power grid computations, direct methods for solving large sparse system equations such as Lower/Upper triangular (LU) decomposition are preferred over iterative methods which suffer from convergence issues. However, our benchmark studies

* This work was partially supported by Dept. of Energy grant CH11171 and by DARPA through the DOI grant NBCH1050009 and the ARO grant W911NF0710416 and Intel. The authors are also grateful to PJM Interconnection for data provided.

show that high performance sparse linear solver software can fail to effectively utilize the available floating point computational throughput on general purpose processors due to data structure overhead and poor data locality [1].

We propose the use of application specific hardware implemented on an FPGA as a method to address the specific computational needs of sparse direct LU decomposition. Our design is based on an empirical study of benchmark power system matrices and their behavior during sparse LU decomposition.

The use of field programmable gate arrays (FPGAs) for high performance floating point scientific computing competitive with high performance general purpose processors has been previously reported [2–7]. Previous work on sparse matrix computations implemented on an FPGA include [5, 6], where sparse matrix vector multiplication is explored, and [8] which utilizes parallel soft-core processing. In contrast, the implementation of an application specific sparse direct LU decomposition hardware design on FPGA has not been previously attempted.

In this paper we present the design and prototype implementation of sparse direct LU decomposition hardware on FPGA, comparing performance to a general purpose processor based platform. Our results show that despite more than an order of magnitude deficit in clock speed as compared to general purpose processors, specialized sparse LU hardware running on an FPGA is capable of an order of magnitude speedup in computation time relative to high performance sparse linear solver packages.

In the next section we review the electric power flow computation which provides us with practical data used to develop high-performance custom hardware. In Section 3 we will review the sparse LU algorithm used in this implementation We present the hardware design in Section 4 and the results of performance evaluation in Section 5.

2 Power Flow Computation

Power flow is the calculation of the complex voltages and powers for a power system network to model the steady state of the system. In this analysis the power system is modeled as a set of nodes, also called buses, interconnected by transmission links, also called branches. The bus admittance matrix, or Ybus, is a mathematical representation of the power system based on Kirchoff's current law. Since the interconnectivity of a typical power system is less than four branches per bus, the Ybus matrix is very sparse for larger systems.

The real (P) and imaginary power (Q) for the ith bus in the power system can be calculated as

$$\begin{aligned} P_i(x) &= \sum_{k=1}^{n} |V_i|\,|V_k|\,[G_{ik}\cos(\theta_i-\theta_k) + B_{ik}\sin(\theta_i-\theta_k)] \\ Q_i(x) &= \sum_{k=1}^{n} |V_i|\,|V_k|\,[G_{ik}\sin(\theta_i-\theta_k) - B_{ik}\cos(\theta_i-\theta_k)] \end{aligned} \tag{1}$$

Where G_{ik} represents the real part of the ik entry in the Ybus and B_{ik} represents the complex part. V_i and θ_i represent the complex voltage magnitude and phase angle at a particular bus in the system. x is the state vector whose elements are the voltage magnitudes and phase angles at the buses in the transmission power grid.

The solution of the equations (1) can be calculated by using the Newton-Raphson method. The Newton-Raphson method utilizes successive refinements to an initial guess and converges quadratically to a solution.

$$\mathbf{J^v} \cdot \mathbf{\Delta x^v} = -f(\mathbf{x^v}) \tag{2}$$

The Jacobian matrix, $\mathbf{J^v}$, and the power mismatch vectors, $\mathbf{\Delta x^v}$ in equation (2), are updated during iteration based on the current solution vector. The repeated solution of the sparse Jacobian matrix can be calculated by LU decomposition.

During the operation of an electric grid, the power flow computation is constantly performed in conjunction with contingency and economic analysis. For each contingency, the grid is assumed to have a fault and power flow is calculated to determine if the fault will cause instability or black out. The number of faults considered for practical operation of a grid demands significant computing resources. Previous study on power flow calculation found that roughly 85% of the computation time is spent on LU decomposition [1, 9]. A cost-effective sparse LU hardware accelerator can provide a high-performance computing solution.

3 Sparse LU Decomposition

Lower/Upper triangular decomposition is the factorization of a matrix A into the product of a lower triangular matrix L and an upper triangular matrix U. For direct methods, it may be necessary to perform pivoting, the swapping of a row/column with another, in order to maintain numerical stability as well as accuracy in the solution. In these cases one or more permutation matrices, P for row and Q for column pivoting, are also included as part of the solution.

$$PAQ(Q^T x) = LU\tilde{x} = PB \tag{3}$$

With the upper, lower, and permutation matrices for a given system of linear equations, the solution x can be obtained by forward and backward substitution.

The LU decomposition of a matrix can be computed by iterative methods such as conjugate-gradient, or by direct methods such as Gaussian elimination. Iterative sparse LU solvers suffer from convergence issues with power flow [10], which restricts our focus to direct algorithms.

Direct LU solvers for dense matrices, require $O(n^3)$ floating point operations for LU decomposition. Sparse matrices on the other hand can benefit from algorithms that reduce the number of operations required to calculate the solution. Unlike dense methods which follow a regular computation pattern though, sparse

methods suffer from irregular computation patterns that are dependent on the non-zero structure of the matrix.

Based on empirical study on the power grid data and power flow calculation we reported that sparse LU decomposition requires roughly $O(n^{1.4})$ floating point operations [1], given a suitable matrix ordering, for the type of sparse matrices encountered in power flow. Properties for some of the power system matrices studied in [1] are given in Table 1. Suitable pre-ordering of the matrix is required to prevent large amounts of fill-in which can degrade the performance of sparse LU solvers. For the power flow computation, the pre-ordering computation time can be amortized over multiple contingencies of the same power system matrix. Our results include the use of the approximate minimum degree (AMD) ordering algorithm [11] computed in software prior to LU decomposition in hardware.

Table 1. Sparse Matrix Properties

System	Matrix Size	Number of Non-Zeros	Sparsity	Post-LU Sparsity	Reported MFlops	Floating Point Efficiency
1648 Bus	2,982	21,196	0.24 %	0.51 %	41.20	1.29 %
7917 Bus	14,508	105,522	0.05 %	0.11 %	39.25	1.23 %
10278 Bus	19,285	134,621	0.036 %	0.079 %	27.57	0.86 %

Benchmark system results from 3.2 GHz Pentium 4
Running UMFPACK 5.2.0 with ATLAS 3.8.0

4 Sparse LU Hardware Design

Our FPGA based sparse LU hardware implements a row-wise, right-looking method of Gaussian elimination with row partial pivoting. To maximize performance, the design of the sparse LU hardware focuses on maintaining regular computation and memory access patterns that are parallel and fully pipelined wherever possible. Synchronous First-In-First-Out buffers implemented with embedded memory blocks are used for high speed buffering of data words throughout the pipelined design. A separate column-oriented mapping (colmap) of the non-zero structure of the matrix reduces pivot search from $O(n^2)$ to $O(n)$ time. The empirical study of power system matrices in [1] provides parameters for the hardware design such as cache line size, total cache size, and buffer depths, minimizing the need to handle more general cases and error conditions with extra logic.

A high level diagram of the sparse LU hardware implementation and basic data flow is depicted in Figure 1. The design of the hardware can be broken down into four main partitions. A central control, implemented as a state machine, tracks the progress of the functional units to ensure synchronized operation. The pivot logic and sub-matrix update logic implement the necessary computations

required for sparse LU decomposition. The last partition, cache, handles sparse matrix data retrieval and storage for the pivot search and sub-matrix update.

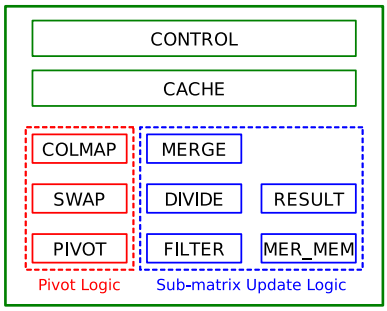


Fig. 1. Top Level Sparse LU Hardware Block Diagram

Not shown are the external memory interfaces to the Sparse LU Hardware, which depend on the FPGA prototype board used for implementation. Our design assumes independent memory banks for the units which require access to external memory such as SDRAM. The colmap utilizes one memory interface to store a column-wise representation of the sparse matrix structure for fast pivot search capability. The cache utilizes another memory interface to store a row-wise representation of the sparse matrix in compressed form. Having two separate memory banks and controllers allow concurrent operation for the colmap and cache units.

A detailed diagram of the pivot search logic and the submatrix update logic is depicted in Figure 2. The logic to perform the pivot search consists of three units, referred to as **colmap**, **swap**, and **pivot**. The pivot selection algorithm used in the hardware design is row partial pivoting based solely on numerical criteria and does not perform any analysis for potential fill-in reduction. The **pivot** logic performs a search, element by element, of the current column for the LU decomposition. The highest magnitude element is selected as the pivot element.

The **colmap** unit first performs a burst read of the column-wise matrix representation to form the pivot column. The **swap** unit maintains a mapping of the pivoting operations that have occurred. This is used to reject candidate rows from the colmap which have already been eliminated. Rows which are not rejected are sent to the cache read queue as single word read requests. The **pivot** unit compares pivot column values returned from cache, selecting the element with the highest floating point magnitude as the pivot element. Once the exhaustive search of the pivot column is complete the swap unit updates the row mappings and the sub-matrix update can begin.

The sub-matrix update logic has two main computations, the normalization of the pivot column by the pivot element, followed by a reduction of the remaining sub-matrix by the product of the pivot row and the pivot column. The

filter unit feeds the pivot column elements to the divide unit to be normalized. The **mer_mem** unit handles cache requests and schedules computation for row updates by the merge unit(s). The **result** unit records the pivot element, pivot row, and normalized pivot column as parts of the final L and U matrices.

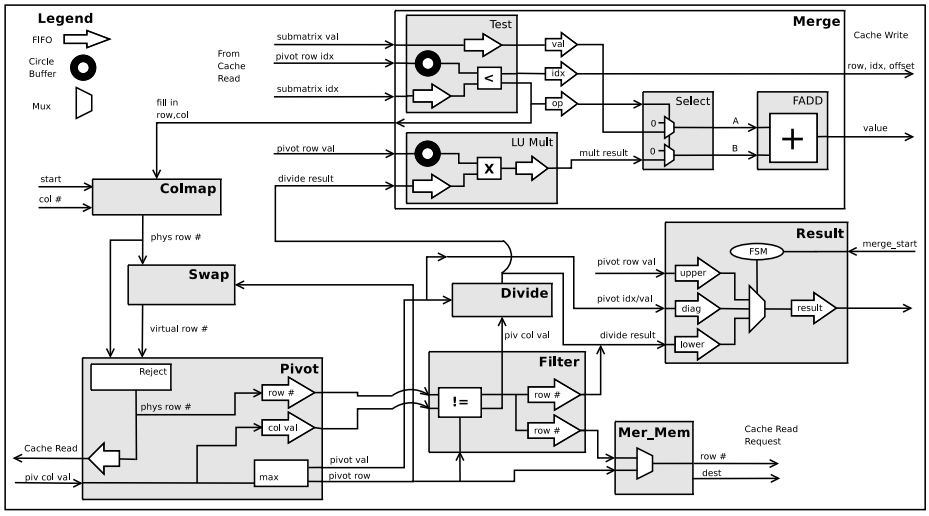


Fig. 2. Pivot and Sub-matrix Update Logic

The **merge** unit performs three tasks in a streaming parallel fashion which make up the bulk of computation. The first is calculating the product of the pivot row and an element of the normalized pivot column. The second is a comparison of the pivot row indices to the sub-matrix row indices to determine the non-zero structure of the reduced row. Finally, the scaled pivot row and sub-matrix row are merged into the new non-zero structure as operands to the floating point addition unit. Additional parallelism is possible by increasing the bandwidth to the cache and instantiating multiple merge units to allow row reductions in parallel.

The use of a memory hierarchy consisting of one or more levels of cache has been used for quite some time in order to address the growing disparity between memory performance and the performance of high speed logic. The use of a cache for our FPGA based Sparse LU Hardware is two fold. The first is to reduce the latency of memory read operations and therefor idle cycles where computations could occur. The second reason, and perhaps most important, is to supply the merge unit with enough scalable read/write bandwidth for high performance.

A detailed diagram of the special purpose cache is depicted in Figure 3. The cache design is single level and utilizes the embedded FPGA memory blocks for cache data storage and tag data arrays. The cache policy is write-back with read miss allocation and a modified First-In-First-Out (FIFO) replacement policy.

The cache is fully associative and stores entire compressed matrix rows to allow high speed constant burst read/write operations. The tag array logic uses content addressable memory (CAM) functionality based on [12] to look up a cache line from a matrix row number. Additional logic guarantees that no rows in-process will be replaced and all writes will be a cache hit.

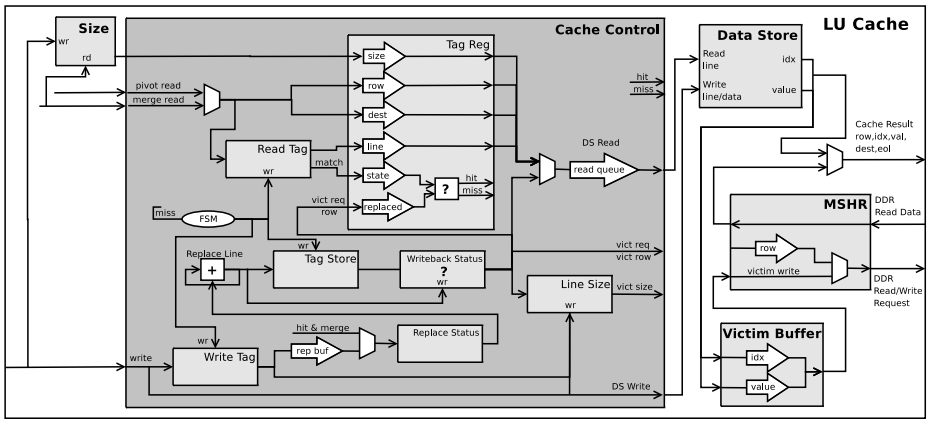


Fig. 3. Special Purpose Cache

This cache is an example of where the reconfigurable nature of the FPGA can allow application specific design for performance enhancements tailored to a specific algorithms data access requirements. Simulation results show that our cache design results in a row read hit rate of ∼85% (word read hit rate over 90%) including compulsory misses; all row writes are hits as previously mentioned.

5 Sparse LU Hardware Performance

In order to evaluate the performance of our hardware design, a prototype was implemented, a scalable performance model written, and performance compared to state-of-the-art sparse linear solver packages on matrices used in power system analysis [13].

To provide a valid reference for the performance of our hardware design, several state of the art linear solver packages were tested for their performance using a system based on a general purpose microprocessor [1]. Data for 1648, 7917, and 10279 bus systems (see Table 1 and [1]) was used. Three state of the art solvers were tested as packaged, UMFPACK [14], SuperLU [15], and WSMP [16]. For the LU decomposition of our benchmark matrices, UMFPACK provided the best solve times so it was chosen to be the reference for comparisons to the performance of the Sparse LU Hardware. Due to the size and structure of the power system matrices, parallel solvers did not perform as well as sequential the sequential packages. In our performance calculations we used the CPU

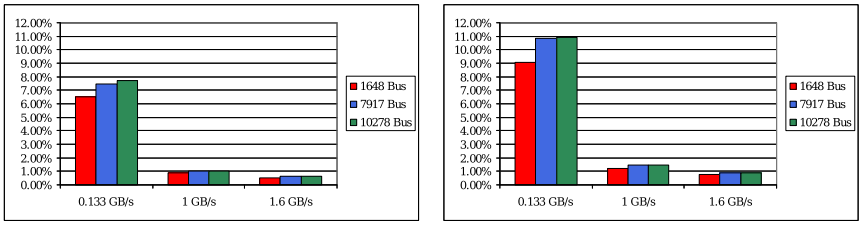
time reported by UMFPACK 5.2.0 for numerical solve running on an Intel Pentium 4 3.2 GHz system. It should also be noted that the pre-ordering used by UMFPACK is the same ordering used for our Sparse LU Hardware.

The Sparse LU hardware was first implemented using SBS Technologies Tsunami PCI board containing of an Altera 1S25 FPGA, 1MB ZBT SRAM, 256MB SDRAM, and running at 50 MHz capable of operating on the 1648 Bus and smaller systems. The number of clock cycles required to perform the LU decomposition for the FPGA based hardware was measured using a hardware counter that increments every clock cycle during LU decomposition. This hardware cycle count is used to verify the accuracy of the software performance model for the Sparse LU architecture. Table 2 details the FPGA resource usage for the Sparse LU Hardware in configurations of 1 and 2 merge units.

Table 2. Sparse LU Hardware Resource Utilization

	Logic Elements	Memory (Kb)	DSP Blocks
Single Merge	15,274	1,503	42
Dual Merge	22,165	1,606	50

To gauge the practicality of a FPGA based accelerator for sparse LU decomposition of power system matrices, analysis of the data transfer time was performed. Figure 4 illustrates the time required to transfer all necessary data to the FPGA relative to the numerical solve time on Pentium 4 running UMFPACK. Three typical interconnections are compared by using the bandwidth rates of PCI, PCI-X, and HyperTransport. These results show that given a high speed interconnect, the data transfer cost would not be a significant bottleneck relative to the calculation time.



(a) Matrix to FPGA (b) Result to CPU

Fig. 4. Data Transfer Relative to Benchmark System Solve Time

The performance model of the LU hardware was written to project performance on larger systems and to determine the impact of design changes. Based on hardware parameters such as latency and cache size, the model reports the

expected number of clock cycles for LU decomposition. The model successfully projects the performance to within 95% of the actual results of the hardware prototype.

Figure 5 compares the projected performance, with varying clock speeds and number of merge units, of the LU hardware versus the run time of UMFPACK [14] running on a Pentium 4 (3.2 GHz) benchmark system. To confirm the pro-

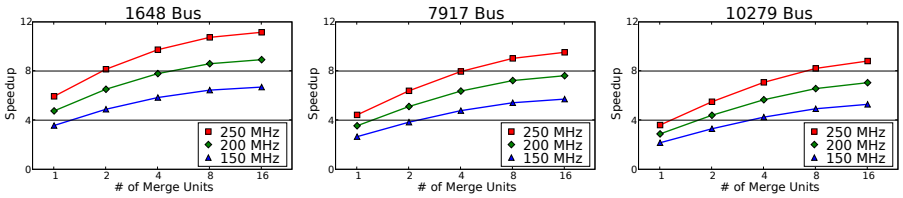


Fig. 5. Performance of Sparse LU Hardware Relative to Benchmark System

jected speedup, we are porting the LU hardware design to a Xilinx Virtex 4 LX200 FPGA which is capable of holding the larger systems, multiple merge units, and running at higher frequencies.

6 Conclusion

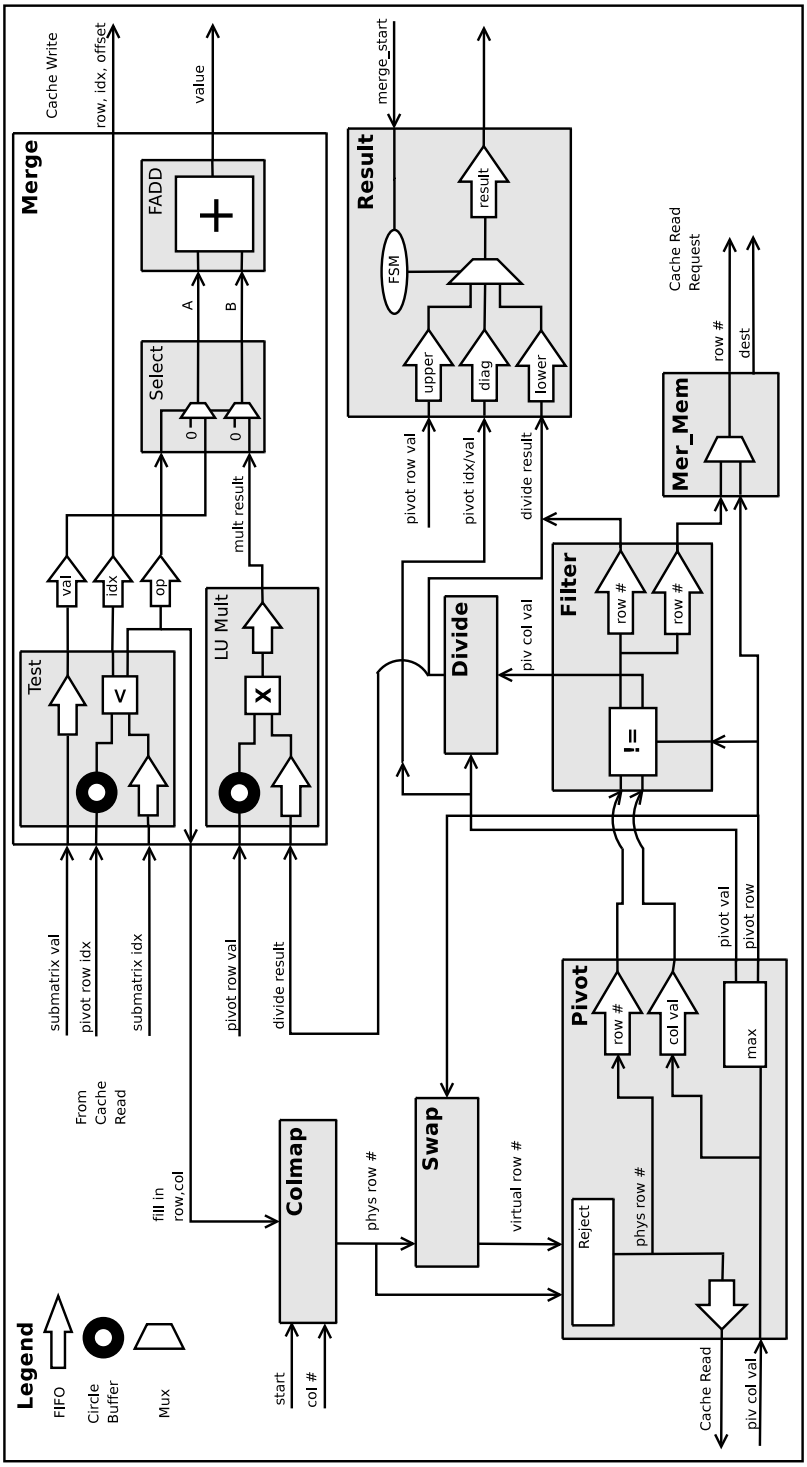
Our results show that the sparse LU decomposition hardware design implemented on an FPGA is capable of an order of magnitude speedup relative to the Pentium 4 based benchmark system. This result highlights the practical use of FPGAs for high performance sparse direct LU decomposition.

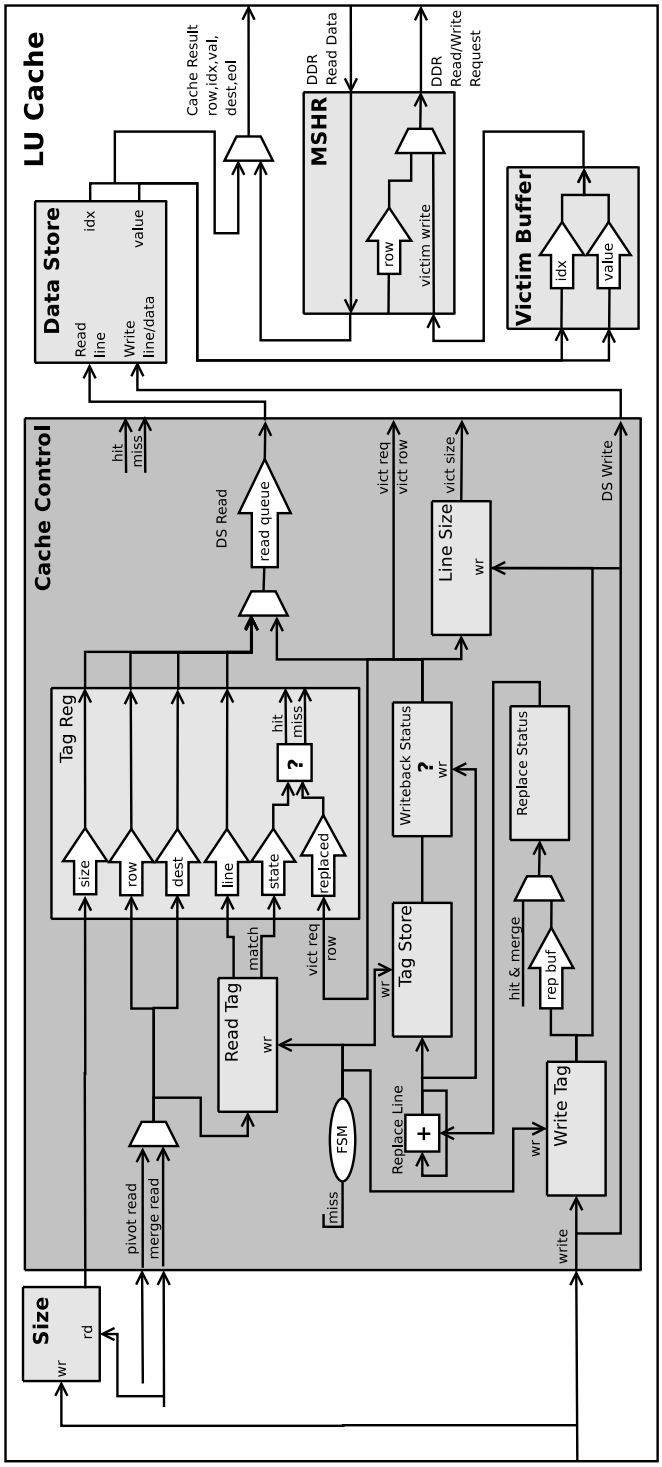
References

1. Vachranukunkiet, P., Johnson, J., Nagvajara, P., Tiwari, S., Nwankpa, C.: Performance analysis of load flow on fpga. In: 15th Power Systems Computational Conference. (August 2005)
2. Underwood, K.: Fpgas vs. cpus: trends in peak floating-point performance. In: FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, New York, NY, USA, ACM (2004) 171–180
3. Underwood, K.D., Hemmert, K.S.: Closing the gap: Cpu and fpga trends in sustainable floating-point blas performance. In: IEEE Symposium on FPGAs for Custom Computing Machines, Los Alamitos, CA, USA, IEEE Computer Society (2004) 219–228
4. Zhou, L., Prasanna, V.K.: Scalable and modular algorithms for floating-point matrix multiplication on fpgas. In: 18th International Parallel & Distributed Processing Symposium (IPDPS04), IEEE (2004)
5. Zhuo, L., Prasanna, V.K.: Sparse matrix-vector multiplication on fpgas. In: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays (FPGA05), New York, NY, USA, ACM (2005) 63–74

6. deLorimier, M., DeHon, A.: Floating-point sparse matrix-vector multiply for fpgas. In: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays (FPGA05), New York, NY, USA, ACM (2005) 75–85
7. Govindu, G., Choi, S., Prasanna, V., Daga, V., Gangadharpalli, S., Sridhar, V.: A high-performance and energy-efficient architecture for floating-point based lu decomposition on fpgas. (April 2004)
8. Wang, X., Ziavras, S.G.: Parallel lu factorization of sparse matrices on fpga-based configurable computing engines. Concurrency and Computation: Practice and Experience **16**(4) (2004) 319–343
9. Tu, F., Flueck, A.J.: A message-passing distributed-memory parallel power flow algorithm. In: Power Engineering Society Winter Meeting, IEEE (2002) 211–216
10. Pai, M., Dag, H.: Iterative solver techniques in large scale power system computation. In: Proceedings of the 36th IEEE Conference on Decision and Control. Volume 4. (Dec 1997) 3861–3866 vol.4
11. Amestoy, P.R., Davis, T.A., Duff, I.S.: An approximate minimum degree ordering algorithm. SIAM Journal on Matrix Analysis and Applications **17**(4) (1996) 886–905
12. Xilinx, Inc.: Designing Flexible, Fast CAMs with Virtex Family FPGAs. (September 1999) xapp203.
13. Vachranukunkiet, P.: Power flow computation using field programmable gate arrays. PhD thesis, Drexel University (2007)
14. Davis, T.A., Duff, I.S.: An unsymmetric-pattern multifrontal method for sparse *LU* factorization. SIAM Journal on Matrix Analysis and Applications **18**(1) (1997) 140–158
15. Demmel, J.W., Eisenstat, S.C., Gilbert, J.R., Li, X.S., Liu, J.W.H.: A supernodal approach to sparse partial pivoting. SIAM Journal on Matrix Analysis and Applications **20**(3) (1999) 720–755
16. Gupta, A., Joshi, M.: Wsmp: A high-performance shared- and distributed-memory parallel sparse linear equation solver. IBM Research Report RC 22038 (98932) (April 2001)

Appendix: Larger Diagrams for Review


Legend
FIFO
Circle Buffer
Mux
Merge
Cache Write
row, idx, offset
value
FADD
+
A
B
Select
0
0
mult result
Test
val
idx
op
<
LU Mult
X
submatrix val
pivot row idx
submatrix idx
pivot row val
divide result
From Cache Read
fill in row,col
Colmap
start
col #
phys row #
Swap
virtual row #
Result
merge_start
result
FSM
upper
diag
lower
pivot row val
pivot idx/val
divide result
Divide
piv col val
Filter
row #
row #
!=
Mer_Mem
Cache Read Request
row #
dest
Pivot
row #
col val
max
pivot val
pivot row
Reject
phys row #
Cache Read
piv col val


LU Cache
Cache Result
row,idx,val,
dest,eol
DDR
Read Data
DDR
Read/Write
Request
MSHR
row
victim write
Victim Buffer
idx
value
Data Store
idx
value
Read
line
Write
line/data
Cache Control
hit
miss
DS Read
read queue
vict req
vict row
vict size
Line Size
wr
DS Write
Tag Reg
size
row
dest
line
state
replaced
hit
miss
?
Writeback Status
?
wr
Replace Status
match
vict req
row
wr
Tag Store
hit & merge
rep buf
Read Tag
wr
FSM
Replace Line
+
Write Tag
wr
pivot read
merge read
miss
write
Size
wr
rd