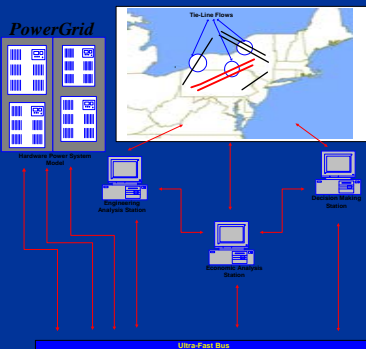


# Sparse Linear Solver for Power System Analysis using FPGA

Jeremy Johnson, Prawat  
Nagvajara, Chika Nwankpa

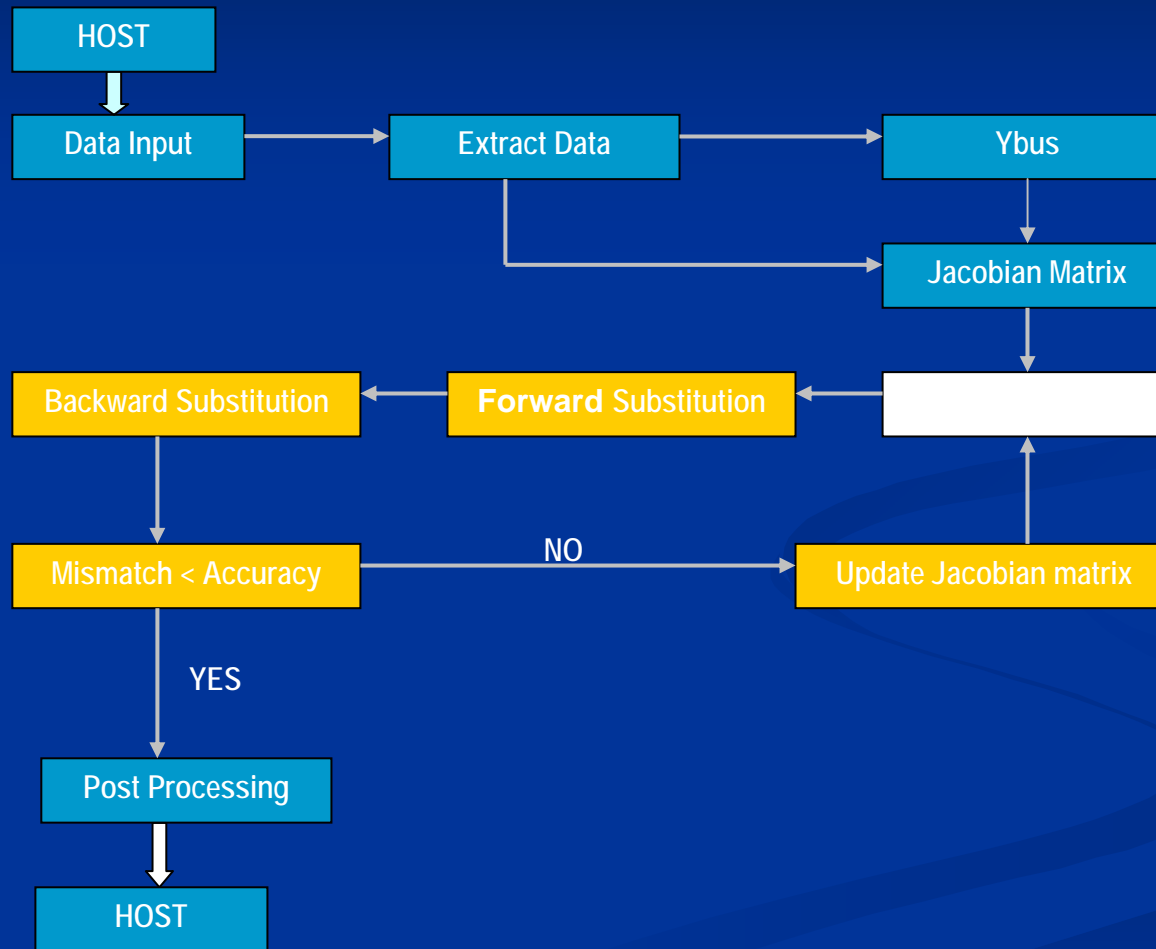
Drexel University



# Goal & Approach

- To design an embedded FPGA-based multiprocessor system to perform high speed Power Flow Analysis.
- To provide a single desktop environment to solve the entire package of Power Flow Problem (Multiprocessors on the Desktop).
- Solve Power Flow equations using Newton-Raphson, with hardware support for sparse LU.
- Tailor HW design to systems arising in Power Flow analysis.

# Algorithm and HW/SW Partition



# Results

- Software solutions (sparse LU needed for Power Flow) using high-end PCs/workstations do not achieve efficient floating point performance and leave substantial room for improvement.
- High-grained parallelism will not significantly improve performance due to granularity of the computation.
- FPGA, with a much slower clock, can outperform PCs/workstations by devoting space to hardwired control, additional FP units, and utilizing fine-grained parallelism.
- Benchmarking studies show that significant performance gain is possible.
- A 10x speedup is possible using existing FPGA technology

# Benchmark

- Obtain data from power systems of interest

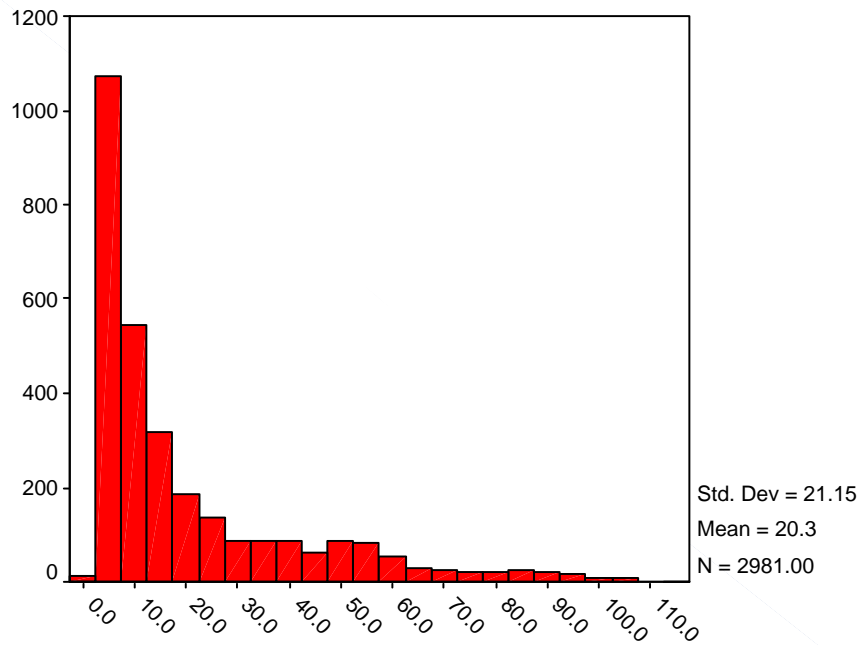
Source	# Bus	Branches/Bus	Order	NNZ
PSSE	1,648	1.58	2,982	21,682
PSSE	7,917	1.64	14,508	108,024
PJM	10,278	1.42	19,285	137,031
PJM	26,829	1.43	50,092	361,530

# System Profile

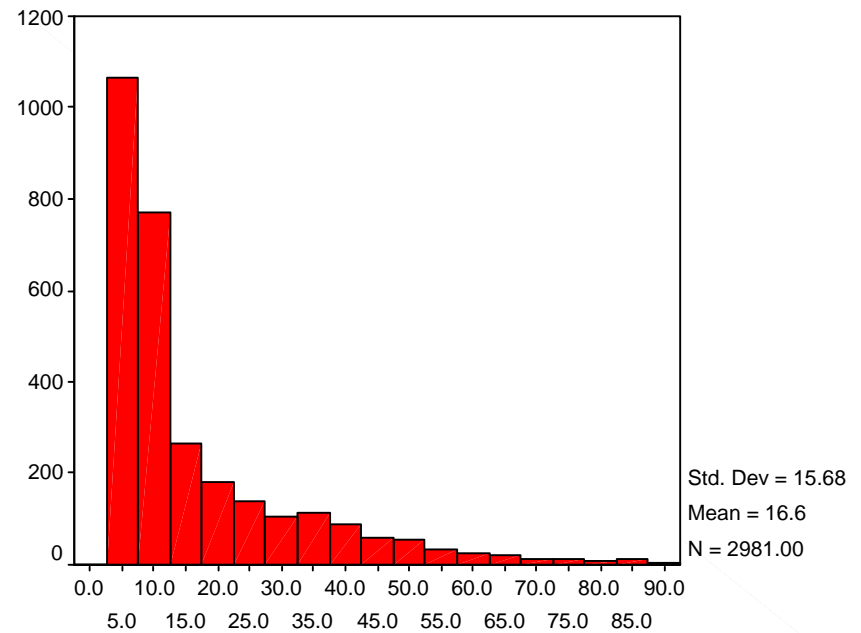
# Bus	# Iter	#DIV	#MUL	#ADD	NNZ L+U
1,648	6	43,876	1,908,082	1,824,380	108,210
7,917	9	259,388	18,839,382	18,324,787	571,378
10,279	12	238,343	14,057,766	13,604,494	576,007
26,829		770,514	90,556,643	89,003,926	1,746,673

# System Profile

- More than 80% of rows/cols have size < 30



ROW\_SIZE



COL\_SIZE

# Software Performance

- Software platform
  - UMFPACK
  - Pentium 4 (2.6GHz)
  - 8KB L1 Data Cache
  - Mandrake 9.2
  - gcc v3.3.1

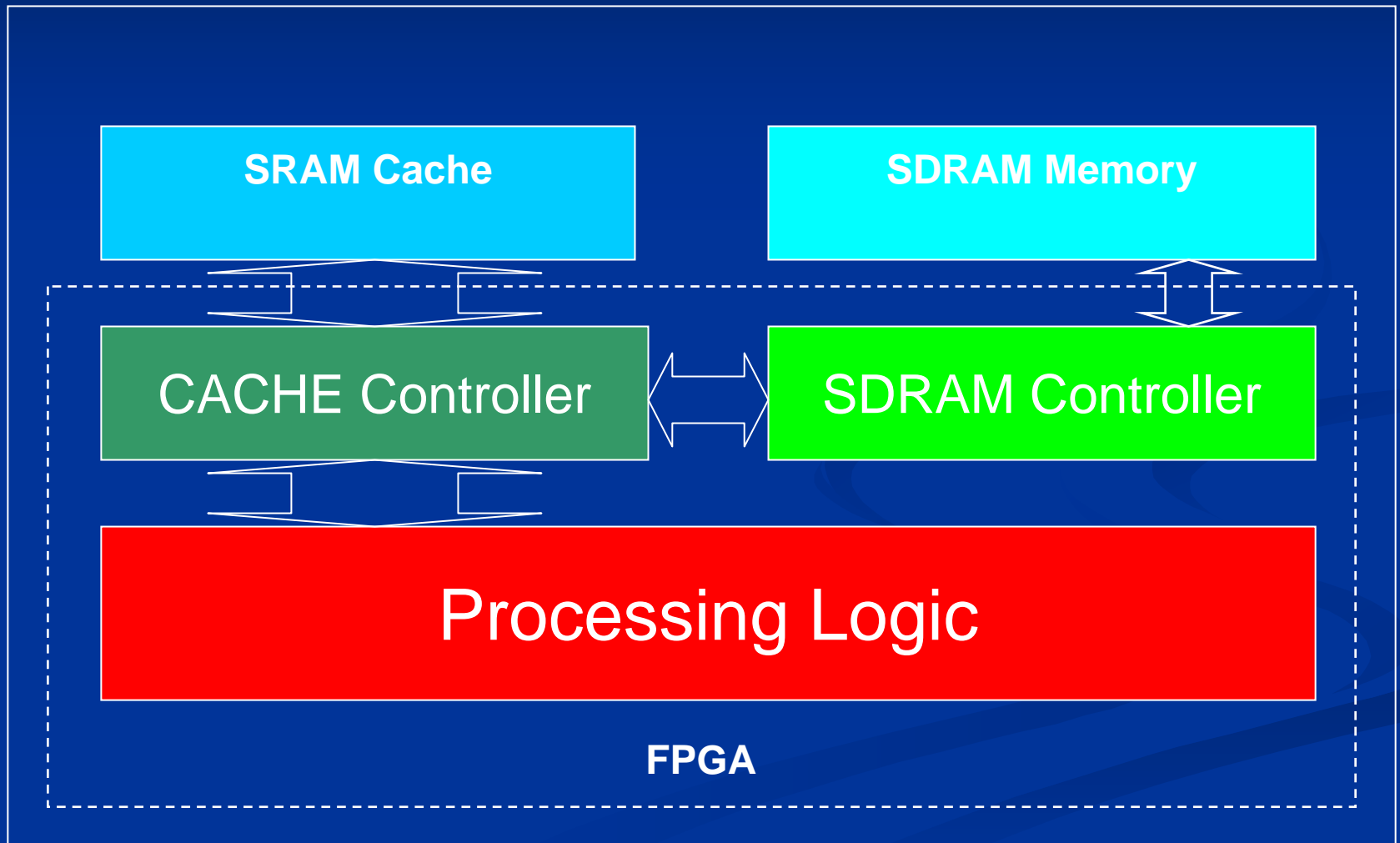
# Bus	Time	FP Eff
1,648	0.07 sec	1.05%
7,917	0.37 sec	1.33%
10,278	0.47 sec	0.96%
26,829	1.39 sec	3.45%



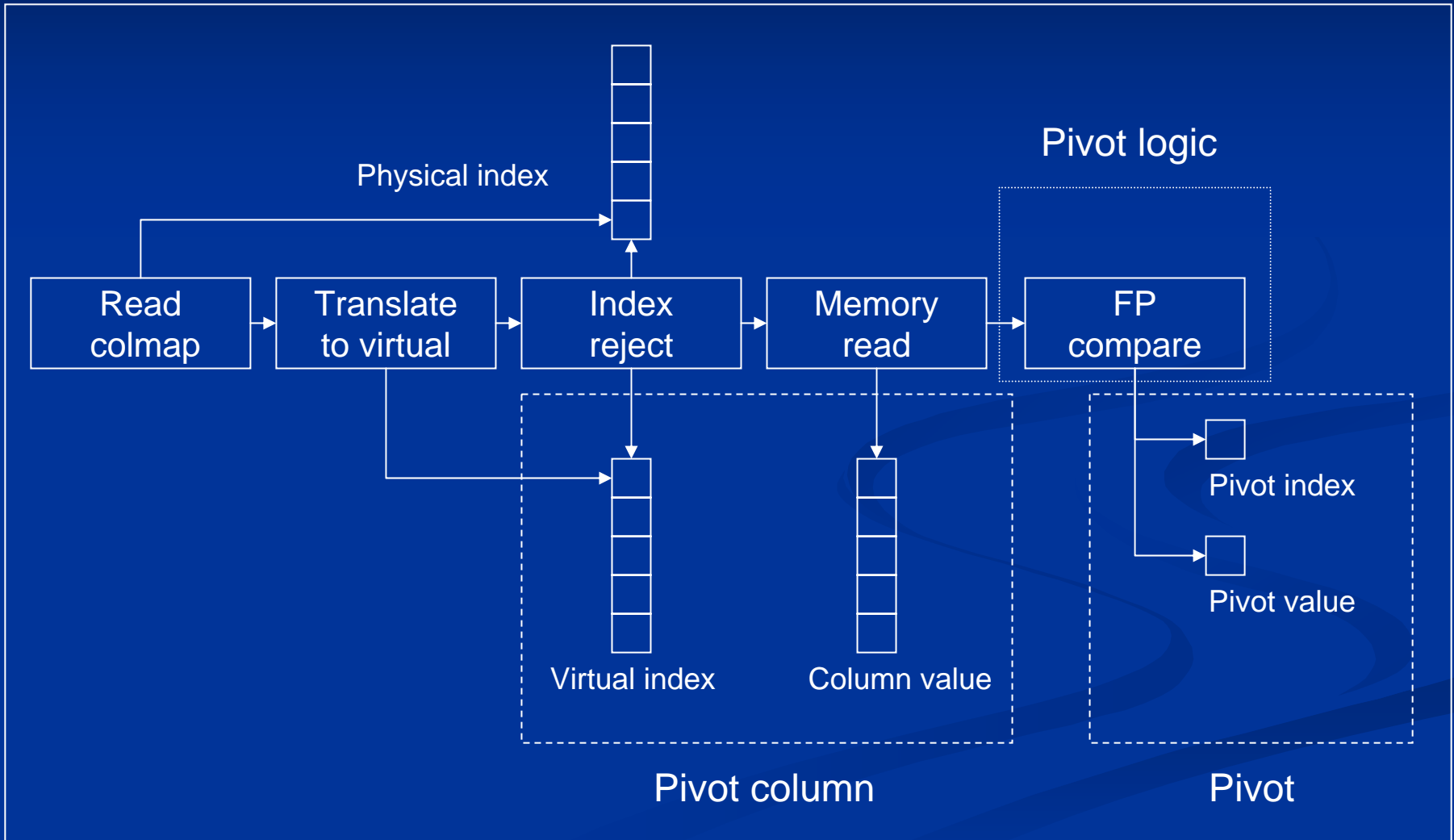
# Hardware Model & Requirements

- Store row & column indices for non-zero entries
- Use column indices to search for pivot. Overlap pivot search and division by pivot element with row reads.
- Use multiple FPUs to do simultaneous updates (enough parallelism for 8 – 32, avg. col. size)
- Use cache to store updated rows from iteration to iteration (70% overlap, memory  $\approx$  400KB - largest). Can be used for prefetching.
- Total memory required  $\approx$  22MB (largest system)

# Architecture

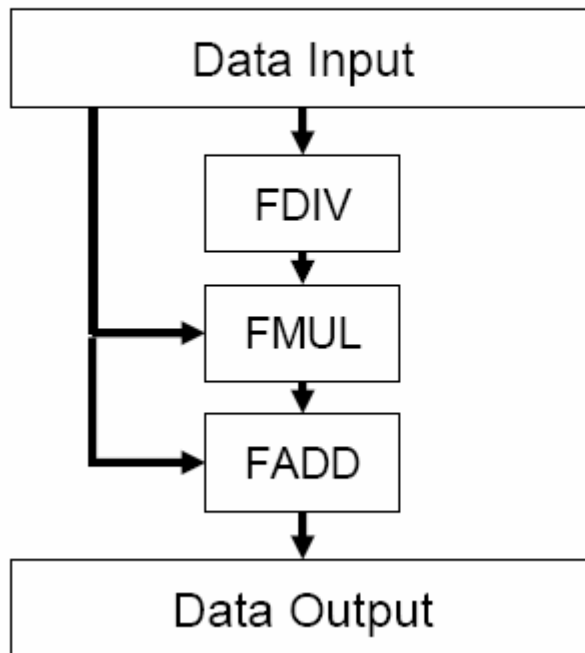


# Pivot Hardware

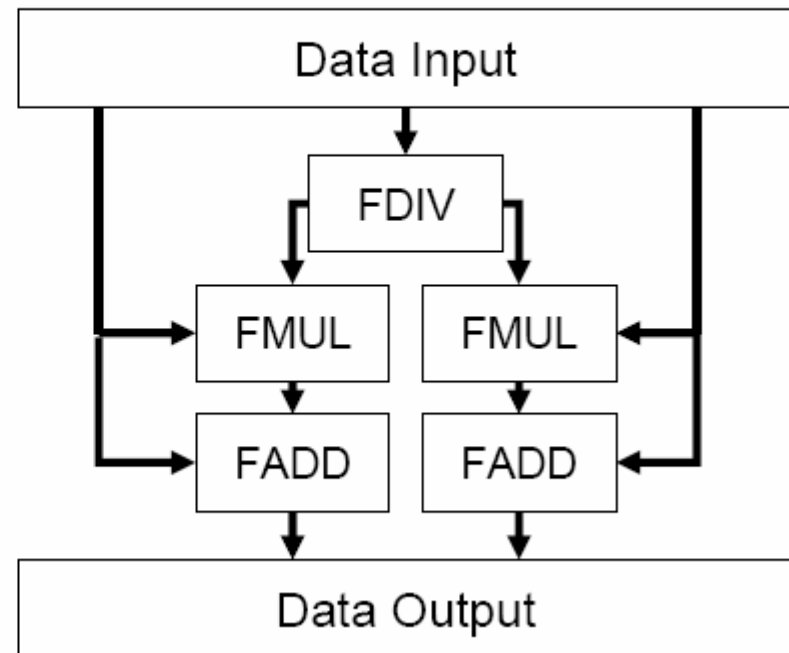


# Parallel FPUs

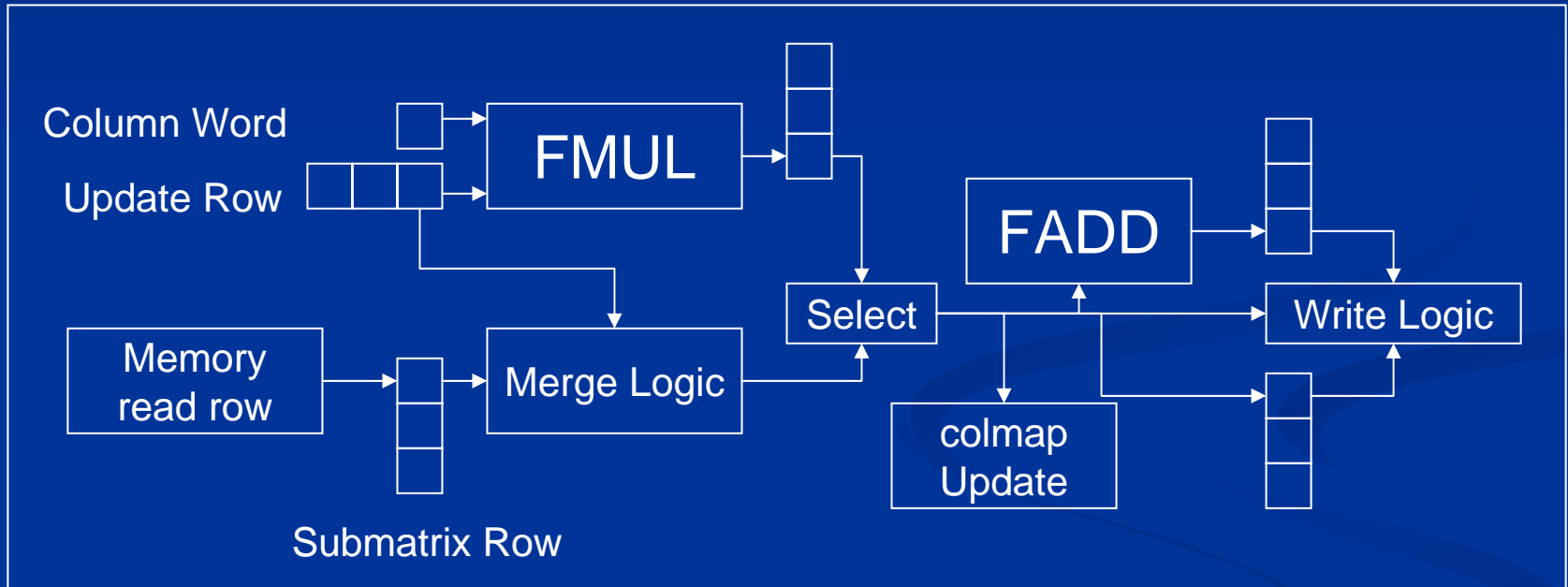
Single FPU



Multiple FPUs



# Update Hardware



# Performance Model

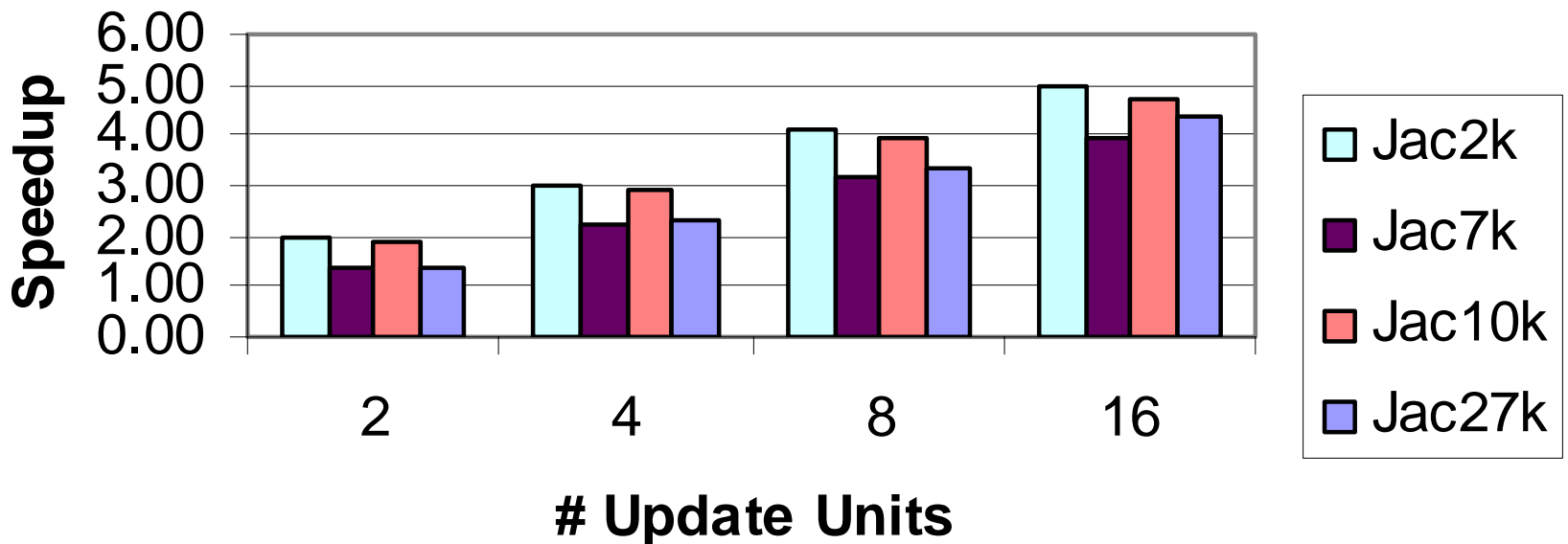
- C program which simulates the computation (data transfer and arithmetic operations) and estimates the architecture's performance (clock cycles and seconds).

## Model Assumptions

- Sufficient internal buffers
- Cache write hits 100%
- Simple static memory allocation
- No penalty on cache write-back to SDRAM

# Performance

## Speedup (P4) vs. # Update Units by Matrix Size

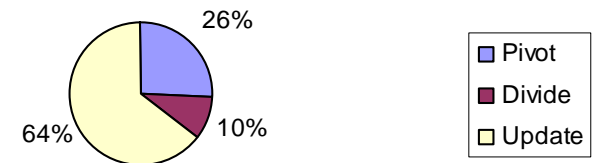


# GEPP Breakdown

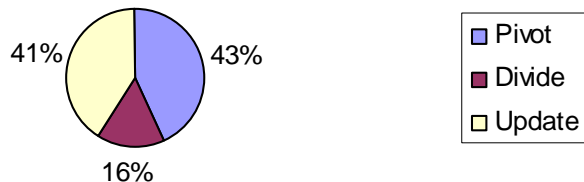
Relative portion of LU Solve Time  
Single Update Unit



Relative portion of LU Solve Time  
Quad Update Units



Relative portion of LU Solve Time  
16 Update Units



Cycle Count  
By # of Update Units

	<u>1</u>	<u>4</u>	<u>16</u>
Pivot	259401	259401	259401
Divide	96439	96439	96439
Update	2409312	642662	248295