Parallel FPGA-based All-Pairs Shortest-Paths in a Directed Graph

Uday Bondhugula

bondhugu@cse.ohio-state.edu

The Ohio State University

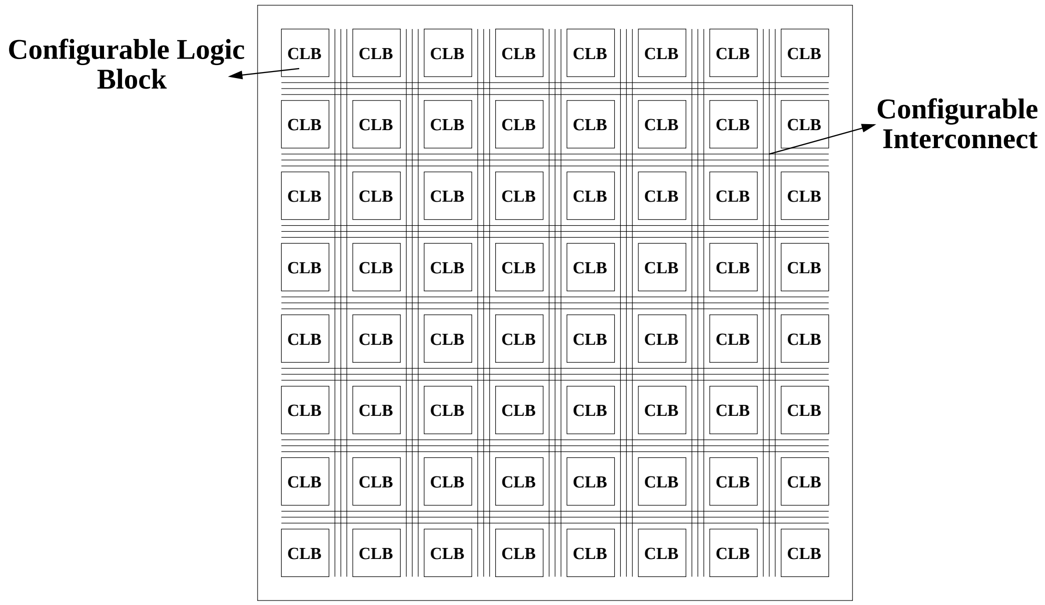
Uday Bondhugula, P. Sadayappan	Ananth Devulapalli, Joseph Fernando	Pete Wyckoff
Dept. of CSE, The Ohio State University	OSC, Springfield	OSC, Columbus

Introduction

- Field-Programmable Gate Arrays (FPGAs) are reconfigurable fabrics that can be programmed to implement desired logic
- Modern FPGAs have a large amount of configurable resources
  - Enables highly parallel designs and effective data reuse
  - Very efficient use of resources
  - Good trade-off between flexibility and performance
- Performance of modern FPGAs competing with that of microprocessors for a wide variety of routines

Overview – FPGAs

- An FPGA comprises a matrix of configurable logic blocks connected by a configurable interconnection matrix
- Each slice has two 4-input lookup tables (LUT), two flip-flops, arithmetic and carry logic
- A large number of dual-ported on-chip block RAMs
- Sufficient resources for routing and global clocking



FPGAs in HPC systems

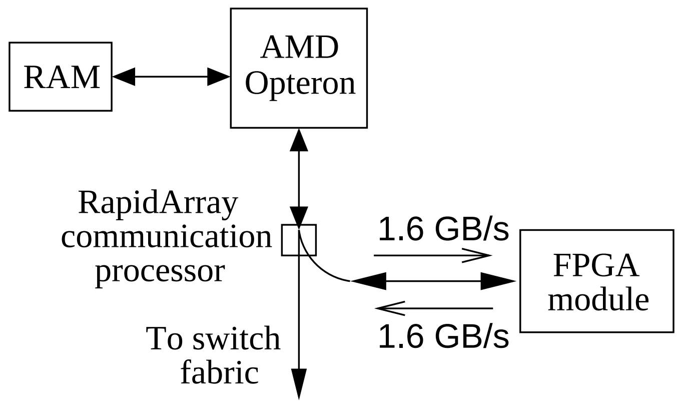


Figure 1: A single node of the Cray XD1

- FPGAs are becoming a viable option for high performance computing
- Several HPC systems – Cray XD1, SRC Mapstation, and SGI RASC employ FPGAs on their interconnects

- Vendor API allows hardware/software integration

Motivation

- The All-Pairs Shortest-Paths problem is to find the shortest path between each pair of vertices in a directed graph
- Applications in Internet topology, geography, interaction networks, VLSI, etc.

Motivation

- The All-Pairs Shortest-Paths problem is to find the shortest path between each pair of vertices in a directed graph
- Applications in Internet topology, geography, interaction networks, VLSI, etc.
- Accelerating a bio-informatics application – *Galaxy*
  - Dynamic Transitive Closure analysis
  - Multiple all-pairs shortest-paths evaluations on thousands of nodes
  - All edge weights between 0 and 1 with accuracy up to three places of decimal desired
  - Runs for several days on modern general-purpose processors

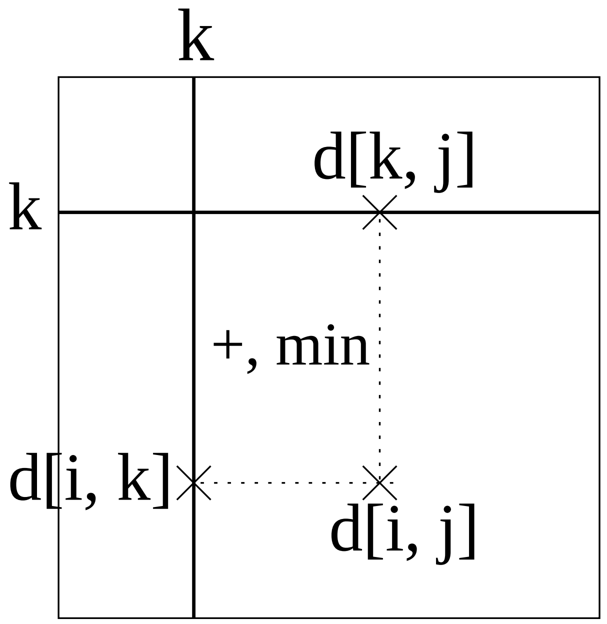
Overview – The Floyd-Warshall algorithm

```
for $k \leftarrow 1, N$ do
    for $i \leftarrow 1, N$ do
        for $j \leftarrow 1, N$ do
            $d[i,j] \leftarrow \min(d[i,j], d[i,k] + d[k,j])$
        end for
    end for
end for
```

- FW uses a dynamic programming approach to solve APSP
- Regular access pattern with significant data dependences
- $\theta(N^3)$ operations

Overview – The Floyd-Warshall algorithm

```
for $k \leftarrow 1, N$ do
    for $i \leftarrow 1, N$ do
        for $j \leftarrow 1, N$ do
            $d[i,j] \leftarrow \min(d[i,j],\, d[i,k] + d[k,j])$
        end for
    end for
end for
```



- FW uses a dynamic programming approach to solve APSP
- Regular access pattern with significant data dependences
- $\theta(N^3)$ operations

Challenges & Issues

- Design should be simple and modular
- Extracting parallelism in the presence of data dependences
- Scalability with resources and System-FPGA I/O bandwidth
- Naive approach: Extract parallelism from the two inner loops – infeasible to build
  - Very high complexity in distributing and addressing data across block RAMs
  - Does not scale with resources on an FPGA, fan-out problem

Challenges & Issues

- Design should be simple and modular
- Extracting parallelism in the presence of data dependences
- Scalability with resources and System-FPGA I/O bandwidth
- Naive approach: Extract parallelism from the two inner loops – infeasible to build
  - Very high complexity in distributing and addressing data across block RAMs
  - Does not scale with resources on an FPGA, fan-out problem
- How do we extract parallelism from the k loop?

New Parallel Design – Our approach

- Organize the computation into two phases

New Parallel Design – Our approach

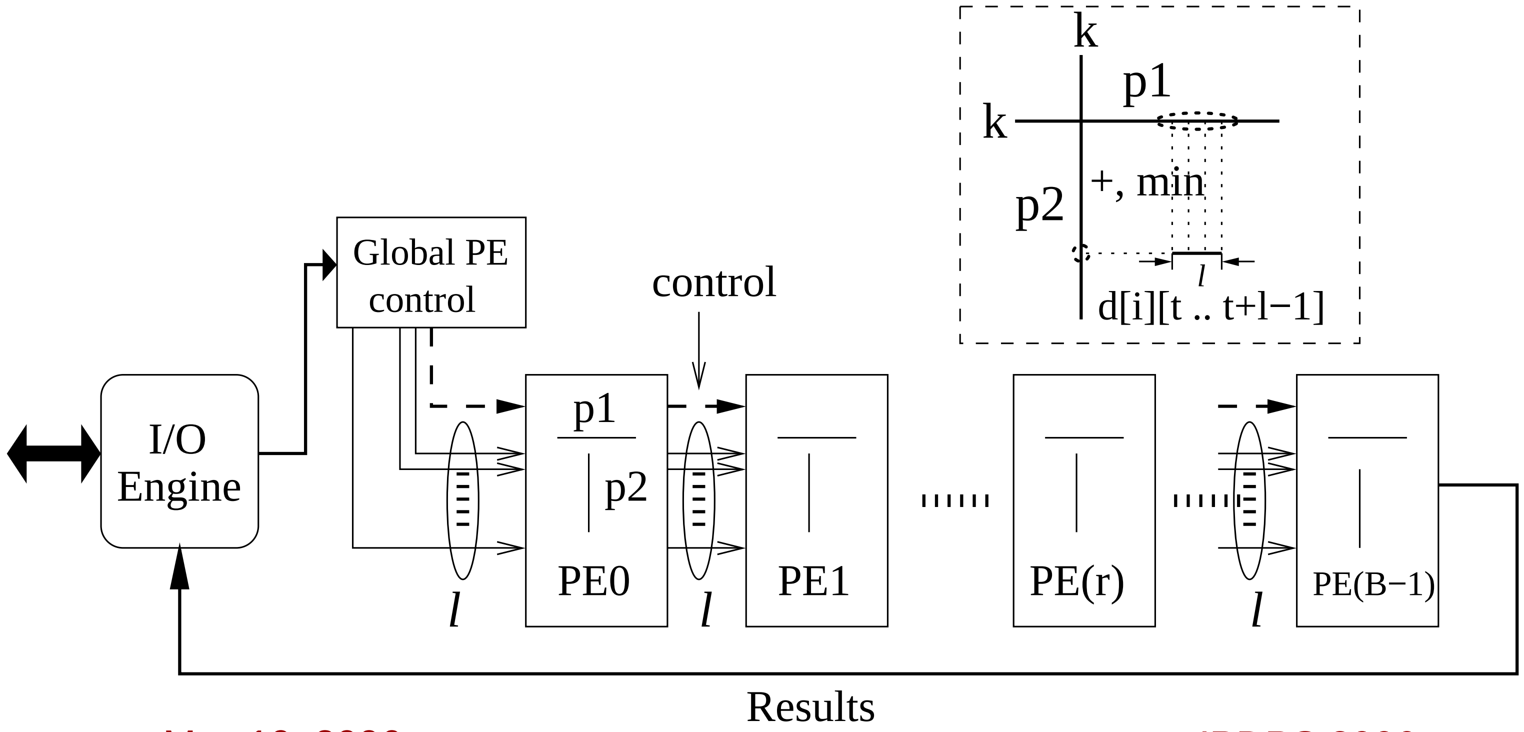
- Organize the computation into two phases
  - **Phase 1** Compute the set of *pivot* rows and columns in advance

New Parallel Design – Our approach

- Organize the computation into two phases
  - **Phase 1** Compute the set of *pivot* rows and columns in advance
  - **Phase 2** Use the stored pivot rows and columns to update the matrix elements in a streamed fashion

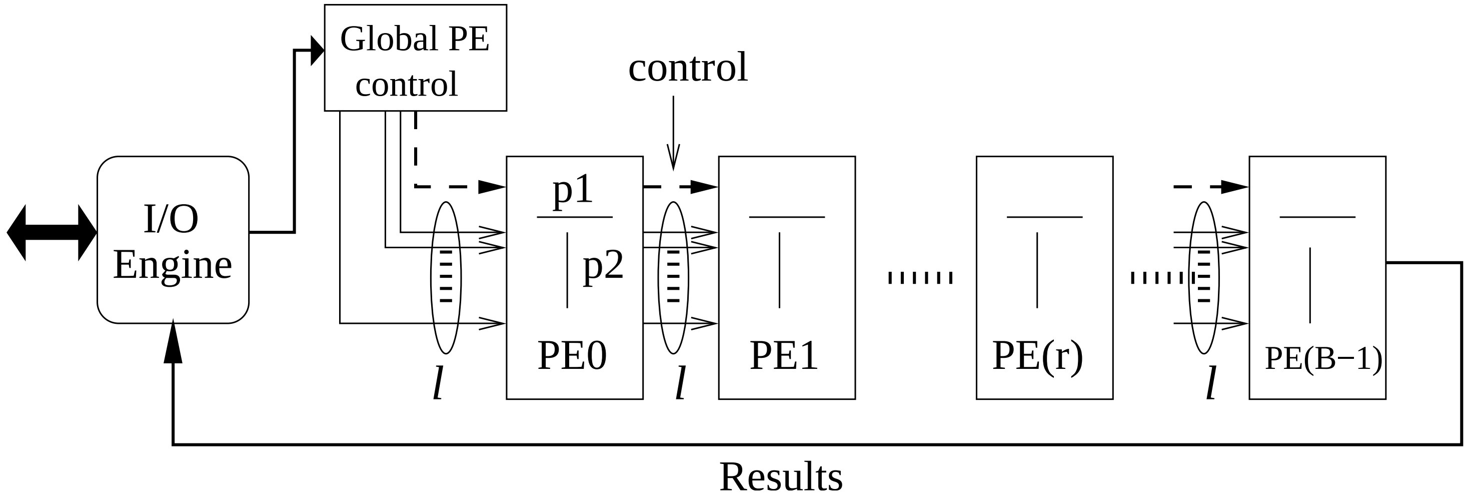
New Parallel Design – Our approach

- Organize the computation into two phases
  - **Phase 1** Compute the set of *pivot* rows and columns in advance
  - **Phase 2** Use the stored pivot rows and columns to update the matrix elements in a streamed fashion
- Linear array of B Processing Elements (PEs) each with l operators



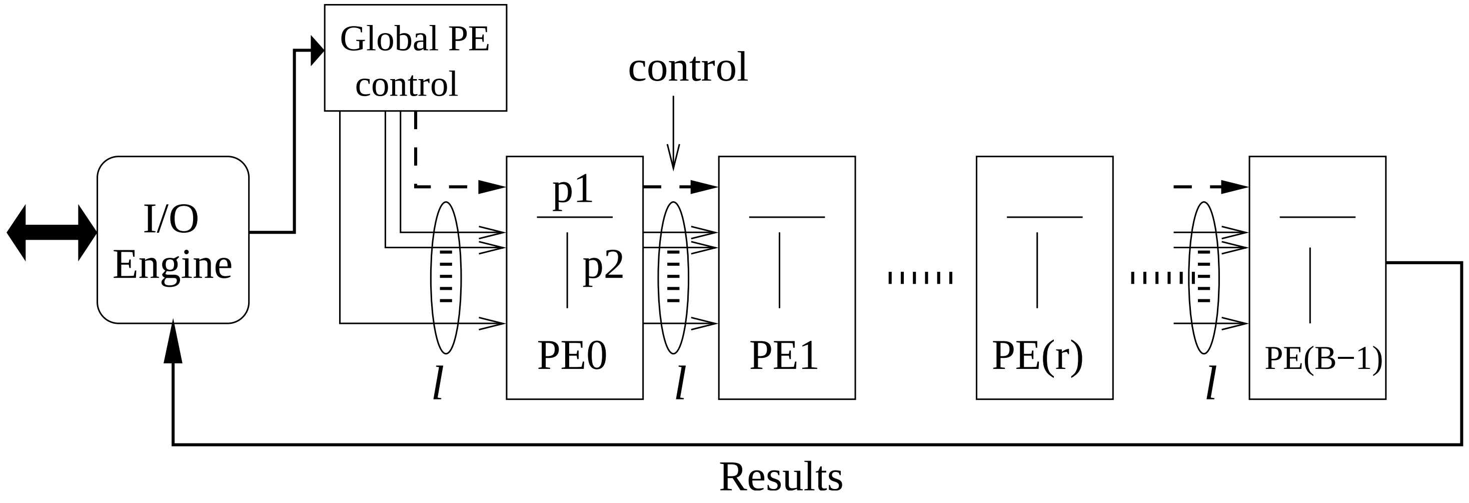
Parallel design: phase 1

- The r^{th} pivot row and column is updated $r-1$ times by upstreams PEs before storage at PE$\langle r \rangle$
- Each PE first stores its pivot row and column, then updates pivot rows and columns meant for downstream PEs
- Latency of phase 1: $T_1(r) = \frac{2B*(B-r)}{l}$ clock cycles



Parallel design: phase 2

- Each PE (from left to right) streams its pivot row downstream on its turn
- The row is updated by all downstream PEs
- The last PE outputs result matrix in row-major order
- Latency of phase 2: $T_2(r) = \frac{B*(r+1)}{l}$ cycles



Parallel design: performance

- Parallelism extracted in both phases
- Latency for processing a single tile:

$$\begin{aligned} L &= T_1(0) + B * p - 1 + T_2(B-1) \text{ cycles} \\ &= \left(\frac{3B^2}{l} + B * p - 1 \right) \text{ cycles} \end{aligned}$$

Extending for a blocked algorithm

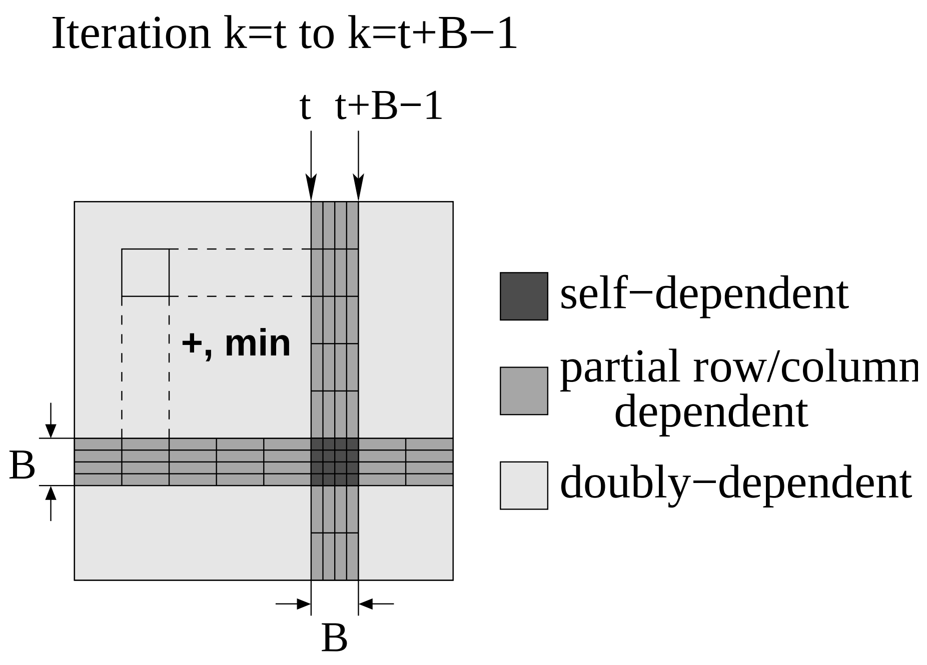


Figure 2: One of the N/B rounds of the blocked algorithm

- Blocking Floyd-Warshall has been addressed by Venkatraman and Sahni.
- Tiles to be processed in a particular order

Extending for a blocked algorithm

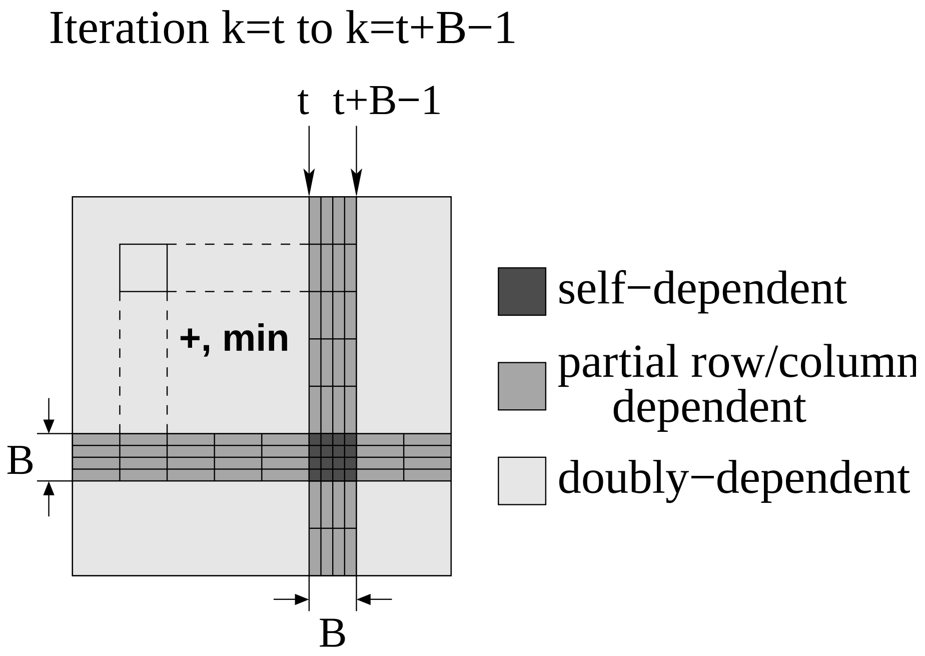
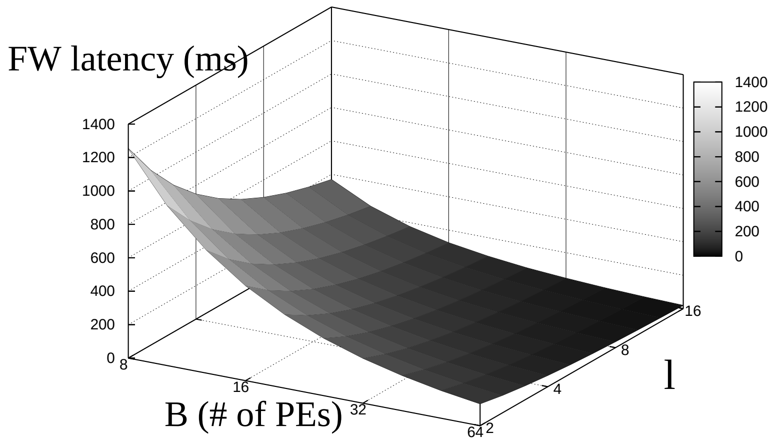


Figure 2: One of the N/B rounds of the blocked algorithm

- Blocking Floyd-Warshall has been addressed by Venkatraman and Sahni.
- Tiles to be processed in a particular order
- Minor changes in the design to incorporate the blocked algorithm
- Pivot rows and columns can come from different tiles

Performance model



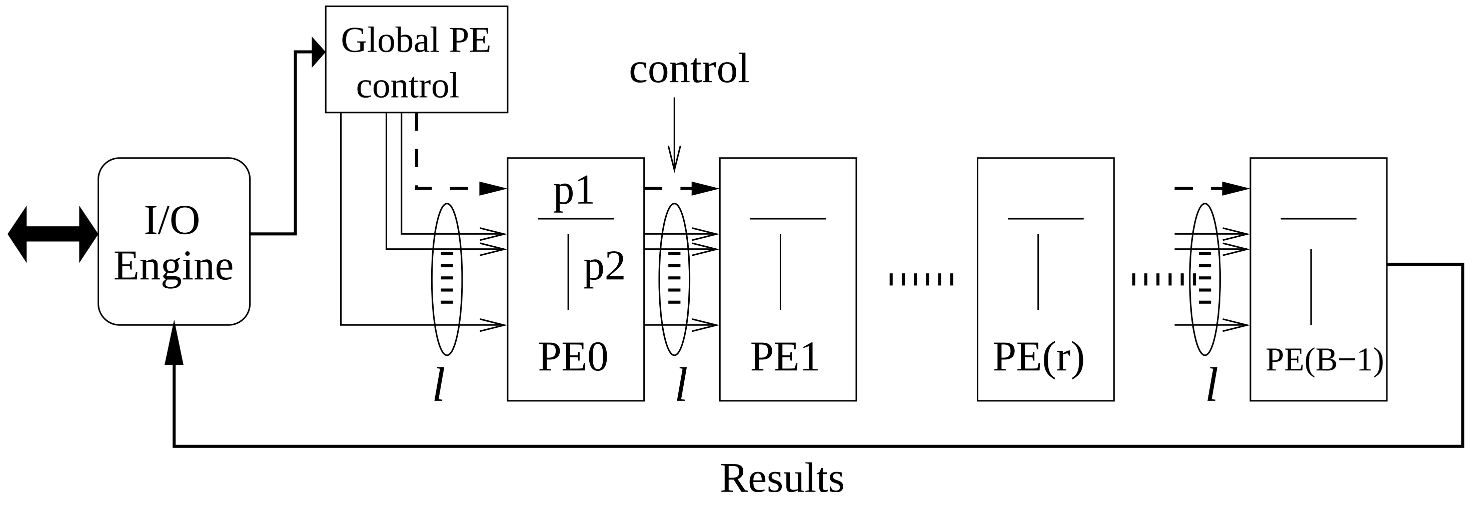
c_g: Global control, c_p: PE control, s_o: operator slices

- **Area constraint**: $s_oBl + c_pB + c_g \leq S$
- **I/O bandwidth constraint**

- The product of B and l is the degree of parallelism
- Scalable with FPGA slices and FPGA–system I/O bandwidth
- Pipelined parallelism (B) more expensive than *doAll* parallelism (l)
- Optimal values for B and l
  - First, determine l to make full use of I/O bandwidth
  - Then, determine B from area constraint

Optimizations

- Optimizing control logic – move as much state and control from the PEs to Global Control
- Instruction associated with each set of l elements
- Pipelining for a high frequency
  – PE pipeline split into three stages (decode, read from pivot memory, and compare/add)
- Overlapping computation of successive tiles



Implementation

Table 1: Resource utilization on the XD1 FPGA (XC2VP50)

Area group	Number of Slices		
	8x8	16x16	32x32
Operator (s_o)	25	25	25
PE	584	550	553
Global control (c_g)	73	73	73
FW	3,983	8,256	17,223
I/O subsystem	3,193		
Total utilization	8,017	12,293	21,229
Available (A)	23,616		
Used	34%	50%	90%

Experimental setup

- CPU Implementation (CPU-FW)
  - AMD Opteron 2.2 GHz with 64KB L1 cache
  - All cases fit in L1 cache
  - Compiled with GCC with -O3
- FPGA-based implementation (FPGA-FW)
  - Cray XD1’s FPGA – Xilinx Virtex-II Pro XC2VP50
  - 32x32 kernel is the best case
  - Clocked at 200 MHz
  - Xilinx ISE 7.1, Cray User FPGA version 1.2
- 16-bit precision for edge weights

Measurements

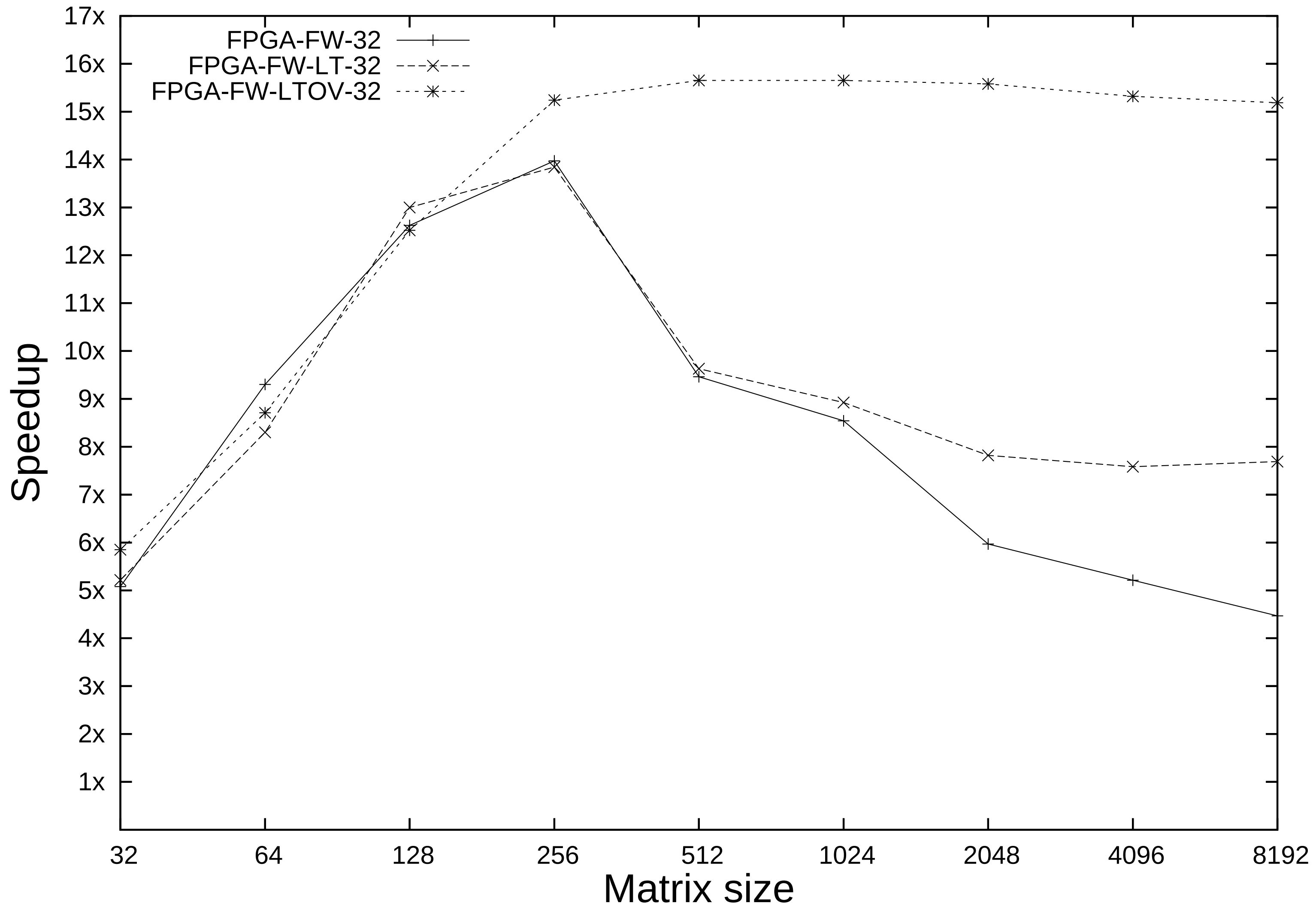
Table 2: Measured performance: FPGA-FW vs. CPU-FW

Tile size	FPGA-FW			CPU-FW
	Total	Overhead	Compute	
8x8	2.49 μs	2.07 μs	0.42 μs	1.6 μs
16x16	3.36 μs	2.07 μs	1.29 μs	14.1 μs
32x32	6.91 μs	2.07 μs	4.84 μs	106.5 μs

Table 3: Estimated vs. measured performance & Speedup

Tile size	FPGA-FW		CPU-FW	Measured Speedup
	Estimated	Measured		
8x8	0.36 μs	0.42 μs	1.6 μs	3.8x
16x16	1.20 μs	1.29 μs	14.1 μs	11x
32x32	4.14 μs	4.84 μs	106.5 μs	22x

Final speedup (FCCM 2006)



Related work

- FW first proposed by Floyd in 1962
- Venkatraman proposed a blocked version of FW to optimize for data locality [J. of Exp. Alg. 2003]
- Zhuo, Underwood have demonstrated competitiveness of FPGAs for double precision floating-point routines
- Significant systolic literature on the Algebraic Path Problem
- Effective hardware/software integration for FPGA-based All-Pairs Shortest-Paths [FCCM 2006]

Conclusions

- Proposed a highly parallel FPGA design for the Floyd-Warshall algorithm
- Simple and modular design that maximizes parallelism and makes maximal use of resources
- Model to determine the optimal values of parameters that govern the exploitable degree of parallelism under resource constraints
- Can be used for highly accelerated solution of large instances of APSP
- Experimental results on the Cray XD1 show a speedup of 22
- **Future work:** Automatic generation of parallel designs for arbitrarily nested loops

Acknowledgment

This work is supported in part by DOE's ASC program.

Questions?