
Hardware/Software Integration for FPGA-based All-Pairs Shortest-Paths

Uday Bondhugula

bondhugu@cse.ohio-state.edu

The Ohio State University

Introduction

- Field-Programmable Gate Arrays (FPGAs) are reconfigurable chips
- Modern FPGAs have a large amount of resources – configurable logic blocks, Block RAM, dedicated multipliers, IO blocks, IO pins
 - ◆ Enables extraction of a large amount of parallelism and effective reuse of data

Introduction

- Field-Programmable Gate Arrays (FPGAs) are reconfigurable chips
- Modern FPGAs have a large amount of resources – configurable logic blocks, Block RAM, dedicated multipliers, IO blocks, IO pins
 - ◆ Enables extraction of a large amount of parallelism and effective reuse of data
- FPGAs offer a good trade-off between reconfigurability and performance
- Performance of current day FPGAs comparable to that of general-purpose processors for a wide variety of routines

FPGAs in HPC systems

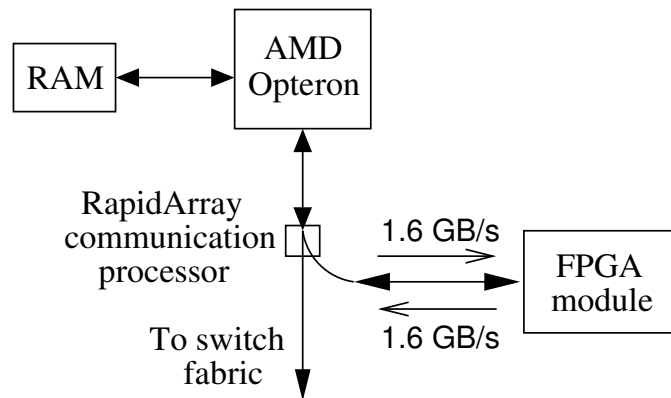


Figure 1: A single node of the Cray XD1

- 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

FPGAs in HPC systems

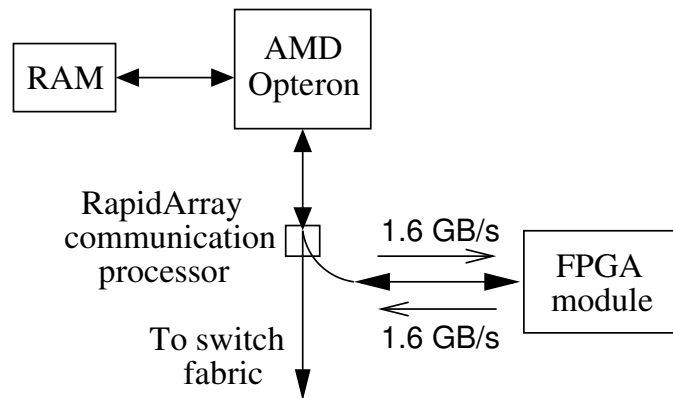


Figure 1: A single node of the Cray XD1

- 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
- The Cray XD1 application acceleration module comprises an FPGA, parallel access to off-chip QDR SRAM
 - Programming the FPGA, exposing FPGA resources (block RAM, registers) to application's address space, provided by vendor API

Motivation

- The All-Pairs Shortest-Paths problem is to find the shortest path between each pair of vertices in a directed graph

Motivation

- The All-Pairs Shortest-Paths problem is to find the shortest path between each pair of vertices in a directed graph
- We were particularly motivated with accelerating a bio-informatics application *Galaxy*
 - ◆ Uses multiple all-pairs shortest-paths evaluations on tens of 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

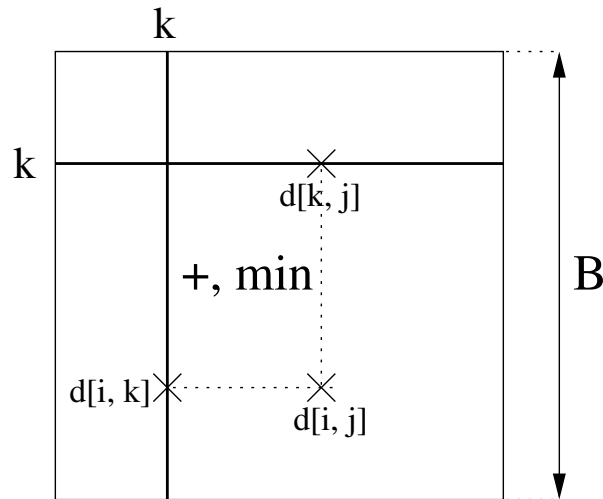
Overview – The Floyd-Warshall algorithm

```
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

Overview of parallel design

- Design should be simple and modular – low routing complexity and easy to build
- Extracting parallelism in the presence of data dependences
- Design should be scalable when higher I/O bandwidth or larger FPGAs are available

Overview of parallel design

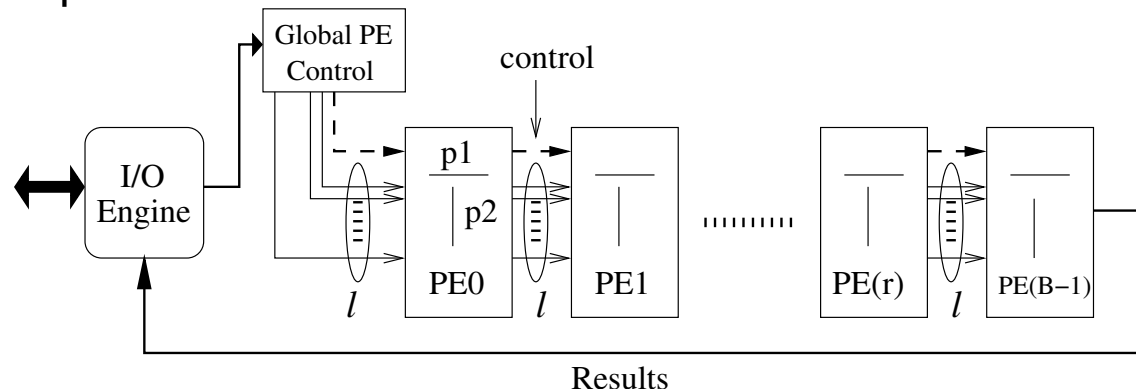
- Design should be simple and modular – low routing complexity and easy to build
- Extracting parallelism in the presence of data dependences
- Design should be scalable when higher I/O bandwidth or larger FPGAs are available

Overview of parallel design (Contd.)

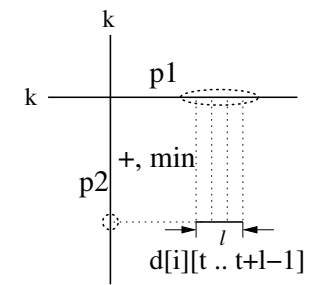
- Extract parallelism from the outermost k loop
- Organize the computation in two phases
 - ◆ Compute a set of *pivot* rows and column in advance
 - ◆ Use the stored pivot rows and columns to update the matrix elements in a streamed fashion

Overview of parallel design (Contd.)

- Extract parallelism from the outermost k loop
- Organize the computation in two phases
 - ◆ Compute a set of *pivot* rows and column in advance
 - ◆ Use the stored pivot rows and columns to update the matrix elements in a streamed fashion
- A linear array of B Processing Elements (PEs) each with l operators

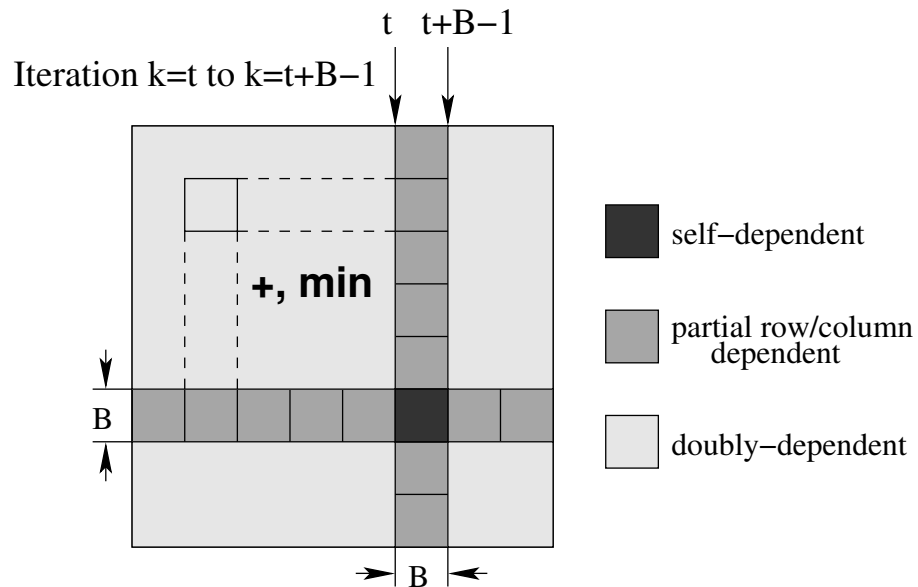


(c) Parallel FPGA-based FW architecture



(d) Update at $PE\langle k \rangle$

Extending for a blocked algorithm



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

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

Extending for a blocked algorithm

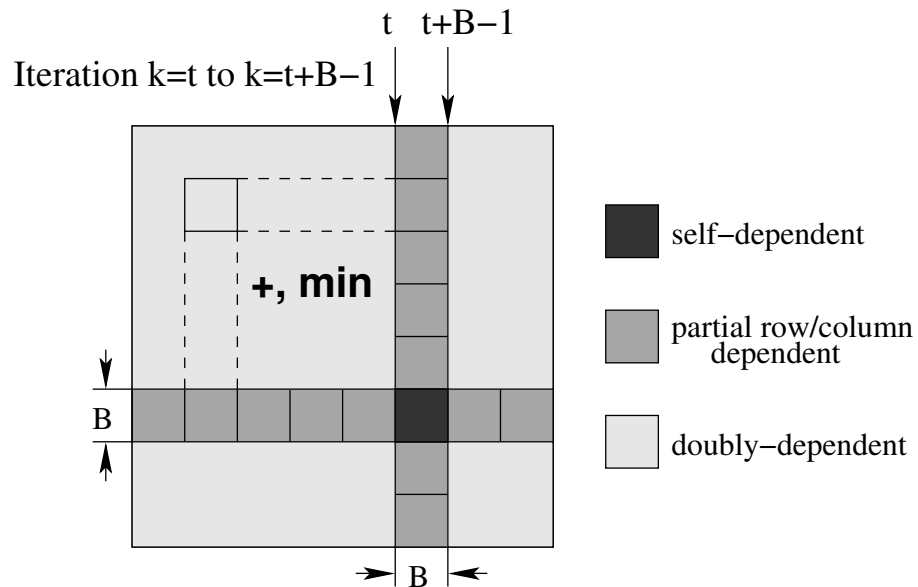


Figure 2: One of the N/B rounds of the blocked algorithm

- Blocking Floyd-Warshall has been addressed by Sahni et al.
- 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

Implementation

Table 1: Resource utilization on the Xilinx XC2VP50 FPGA

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%

Implementation

Table 1: Resource utilization on the Xilinx XC2VP50 FPGA

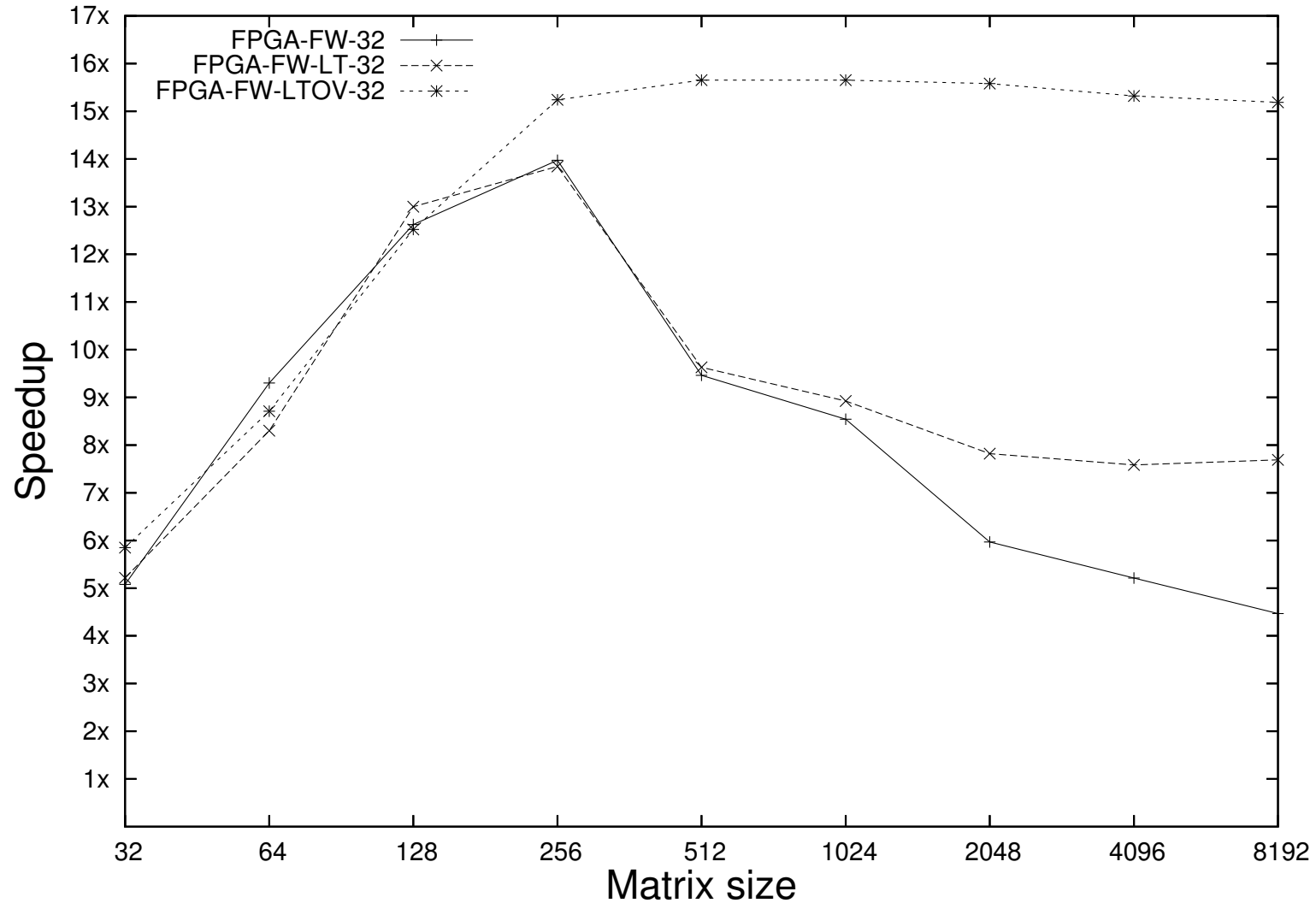
Area group	Number of Slices		
	8x8	16x16	32x32
Operator (s_o)	25	25	25
PE	584	550	553
Global control (c_g)	73	73	73
FW	3,983	8,256	17,223
I/O subsystem	3,193		
Total utilization	8,017	12,293	21,229
Available (A)	23,616		
Used	34%	50%	90%

- A set of registers for status and control
- I/O engine is oblivious to computation
- Data in communication buffer interleaved in the fashion required for the particular kind of tile
- Nice balance between computation and communication – idle instruction when data is not available

Experimental setup

- CPU Implementation (CPU-FW)
 - ◆ AMD Opteron 2.2 GHz with 64KB L1 cache
 - ◆ Compiled with GCC with -O3
 - ◆ Blocking with block size empirically optimized
 - ◆ Copying to avoid conflict misses
- FPGA-based implementation (FPGA-FW)
 - ◆ Cray XD1's FPGA – Xilinx Virtex-II Pro XC2VP50
 - ◆ 32x32 kernel is the best case
 - ◆ Clocked at 200 MHz
 - ◆ Cray User FPGA version 1.2
- 16-bit precision for edge weights

Measurements: application-level speedup



Model

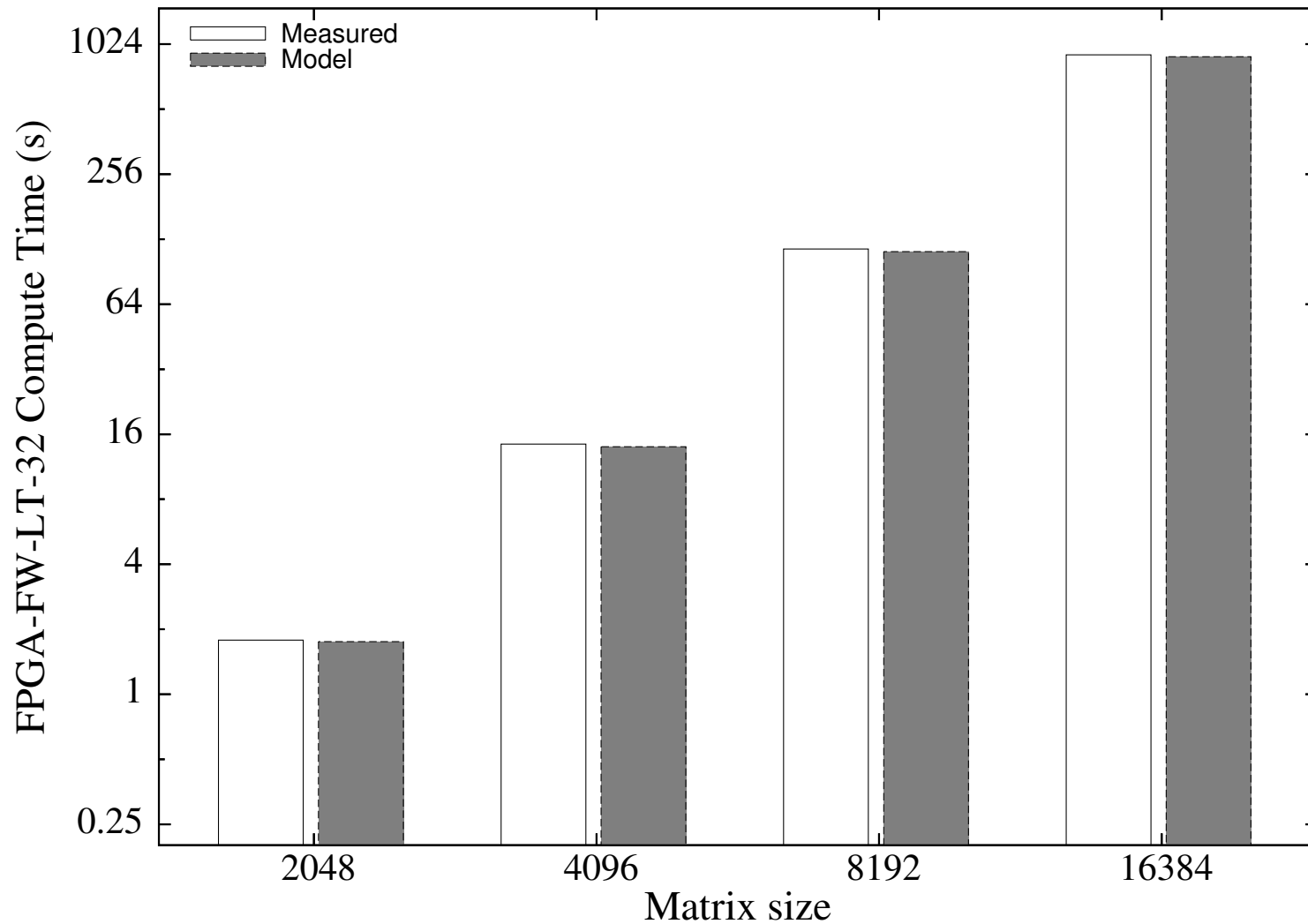


Figure 3: Measured compute time vs. Model for FPGA-FW-LT-32

April 22, 2006

FGCM 2006

Related and future work

- Tripp et al. SC 2005

Related and future work

- Tripp et al. SC 2005
- IPDPS 2006

Conclusion

- Speedup increased from 4 to 15 due to optimizations

Conclusion

- Speedup increased from 4 to 15 due to optimizations
- Compute/Copy overlap and layout transformation

Conclusion

- Speedup increased from 4 to 15 due to optimizations
- Compute/Copy overlap and layout transformation
- Model to give insight into effect of hardware parameters

Questions?
