Breadth-First Search on Xilinx Versal

Guilherme Prado Alves*[1], Antonino Tumeo[2], Marco Minutoli[2], Mehmet E. Belviranli[1]

[1] Colorado School of Mines, [2] Pacific Northwest National Laboratory

Versal Architecture

Scalar Engines (CPU)

- The Versal device contains a set of A-series and R-series ARM processors
- More compatible with external libraries, handles complex single-threaded algorithms well

Adaptable Engines (FPGA)

- FPGAs are best suited to handle time-sensitive computation
- Contains a High-Speed AXI interface between FPGA and AI Engines

Intelligent Engines (AIE)

- The AI Engines handle SIMD well due to wide vector registers and compute units
- Complex logic can be written and pipelined by joining many AI Engines together

Network on Chip

- The Versal board has a high-speed interconnect to allow communication between all Engines, as well as external interfaces

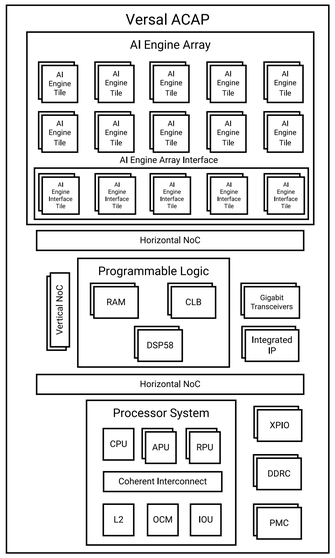
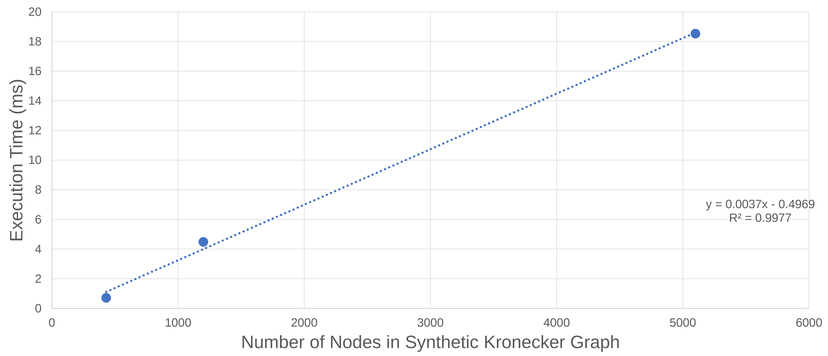


Image courtesy of Xilinx Inc

Graph Algorithms as Linear Algebra

- Graph Algorithms can be represented by Linear Algebra operations via Semirings [2].
- This approach eliminates the need for programs that specifically implement one algorithm. Operations on Graphs are easily modified as necessary, making the program more versatile. Further, Linear Algebra is more easily scalable than other Graph algorithms.
- Breadth-First Search (BFS) is one algorithm that may be represented through Matrix Multiplication in the Boolean Semiring
  - In this representation, real numbers are replaced by Boolean Values, scalar multiplication is replaced by the logical AND, and scalar addition is replaced by the logical OR operation.
  - The adjacency matrix ($\boldsymbol{A}$) is filled with either 0's or 1's, where a 1 indicates an edge.
  - The current frontier is represented by a vector ($\vec{v}$) of size equal to the number of nodes
    - A 1 indicates that a given index is present in the current frontier, a 0 indicates otherwise
  - Given both elements, the expression $\boldsymbol{A}^T\vec{v}$ computes the next frontier

Preliminary Results



- Execution Time data was gathered using synthetic Kronecker graphs obtained from MIT Graph challenge, and converted to a format that could be used in this program
- Given some challenges using the Versal platform, this data was gathered by simulating an Alveo U200 platform
  - This performance should be approximately the same as the Versal, as memory accesses bottleneck performance given the very low computational density of these kernels
  - Versal should present lower power consumption than the Alveo U200 platform, as that was the designed purpose of the AI Engines

Program layout

Pull Data from Memory

- Check if each row has been visited
- If not, pull each window of the row's adjacencies and the vector to send to AI Engines

Perform SpMSpV

- Given a pair of windows, check if one of the node's parents belongs to the current frontier
- If yes, write the node index to next kernel

Write Back to Memory

- For each node index received, update the current record on the Columns Index Array
- Once all stop signals have been received, signal next kernel

Gather Next Frontier

- Go through all items in Columns Index Array. If the step recorded is the current step, add that node to the next frontier.
- If the frontier is empty, signal all to stop

Optimizations

Window Splitting

- AI Engines begin execution when all inputs are available
- Both the graph and the frontier vector are partitioned into small chunks called windows that can be read in and passed to the AI Engines directly by the Programmable Logic (PL).
- Accessing the whole window coalesces the memory accesses and makes it possible to fully utilize wide memory interfaces

Input Sparsity

- When performing the multiplication $\boldsymbol{A}^T\vec{v}$, each row is checked one at a time
- However, only the rows that have not yet been visited need to be pulled from memory and checked for inclusion in the next frontier
- In a standard CSC sparse matrix, 3 memory accesses would have to be made, one to check for the row's inclusion, one for the row's start index in the columns array, and one to the columns array
- Attia et al. [1] reduce this to 2 memory accesses by packing the inclusion check and the start index into a single 64-bit value.
  - The LSB carries a binary flag signaling inclusion, while the other 63 bits carry the start index and length, or visit depth as appropriate

Short Circuiting

- Matrix Multiplication on the Boolean Semiring is a Boolean expression for each row
- This expression is in disjunctive normal form (DNF). Therefore, only one of the conjunctions must be true, and the entire expression must be true
- Given this, the dot product must only be calculated until a single conjunction becomes true
  - As implemented, this would mean that a sufficient condition for stopping evaluation of a row is if the same index occurs in both a row and the current vector.
  - If that happens, the rest of the row and vector can be disregarded

References:

[1] Osama Attia, Tyler Johnson, Kevin Townsend, Philip Jones, and Joseph Zambreno. 2014. CyGraph: A Reconfigurable Architecture for Parallel Breadth-First Search. In 2014 IEEE International Parallel Distributed Processing Symposium Workshops. 228–235. https://doi.org/10.1109/IPDPSW.2014.30

[2] Jeremy Kepner and John Gilbert. 2011. Graph Algorithms in the Language of Linear Algebra. Society for Industrial and Applied Mathematics, USA.

U.S. DEPARTMENT OF ENERGY | BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

For additional information, contact:

Guilherme Prado Alves | (509) 375-1234 | galves@mines.edu

* This work was performed when the author was affiliated with PNNL

10/12/2021 | PNNL-SA-165946

COLORADO SCHOOL OF MINES

Pacific Northwest NATIONAL LABORATORY

www.pnnl.gov