EXCURSIONS INTO DIVERSE AREAS OF COMPUTER ARCHITECTURE

Vima Gupta
vgupta345@gatech.edu

1 Overview

Over the course of the summer semester, I had the opportunity to explore a variety of topics being worked on in the TINKER lab.

- SpGEMM papers, a literature survey summary
- Program profiling for exascale proxy apps
- Synthesis tool chain setup for neuromorphic branch predictor

2 Project Summary

2.1 Sparse Matrix Multiplication

Sparse matrix multiplication was the primary focus of my study in the initial phase of the semester. Survey papers reveal latest advances in the algorithmic and architecture related advances for SpGEMM [1]. There is a noted advantage in using bipartite graph models over hypergraphs for 1-D slicing [2]. Using register-aware algorithms provides an interesting take on enhancing utilization of registers for sparse accumulators [3]. Analysis of inner product versus outer product shows that row-by-row technique (inner product) for matrix mulitplication is best for first two multiplications and outer-product is better for third multiplications and onwards [4]. Utilizing 3-D parallel formulation is a technique that is used to hide communication under computation latency [5].

Interesting advances have also been made on sorting techniques for intermediate results that are used for accumulation in SIMD and NUMA architectures. Radix-hash join has been claimed to be superior to the previously believed sort-based join for in-memory data processing techniques [6]. This paper outlines the techniques used in supercomputers like multi-level parallelism and optimized kernels such as COO, CSR, ELL and CSC [7]. Chunking-based algorithm make justified claim for kernels with emphasis on multi-level memories take advantage of cache reuse techniques [8]. A further take on chunking algorithms details dynamic chunking of matrix streams for efficient memory allocation [9].

SpGEMM on GPU's calls for improved load balancing achieved via uniform assignment of non-zeros to thread blocks such that bit-stable results can be ensured [10]. FPGA's have the additional constraint of limited bandwidth and can be addresses by following techniques. Performance bottlenecks are alleviated via decoupling the process of index matching and the multiplication to accomplish load balance [11]. [12] A unique performance enhancement for SpGEMM for FPGA has been identified, by having dedicated modules for index comparison and floating point computations [12].

In addition to the survey, I also did an in-depth study of SPAGHETTI (Streaming Accelerators for Highly Sparse GEMM on FPGA's) architecture [13]. Spaghetti takes advantage of our observation that in the outer product the rows in the input matrix lead to mutually independent rows in the final output. Thus, the scheduler can partition the input into tiles that maximize reuse and eliminate re-fetching of the partial matrices from the DRAM.

2.2 ECP proxy apps profiling

In the next phase of the project, I explored the realm of code profiling and exascale computing. To begin with, I compiled and ran open-source program, QuickSilver. Followed by a brief analysis of the most time consuming kernels, I moved on to application from the Exascale Computing Proxy Apps. For my analysis, we chose MiniFE, which is touted to be the best approximation to an unstructured implicit finite element or finite volume application under 8000 lines of code. Using Vtune, I performed some preliminary analysis for performance and hotspots.Performance bottleneck (as seen in Figure 1) was identified in a sparse matrix multiplication module where the search for an element in a given row stored in a compressed row format. This call to set based data structure leads to a time complexity of O(NlogN) which calls for experimenting with other compression techniques such as CSC and COO for improvement.

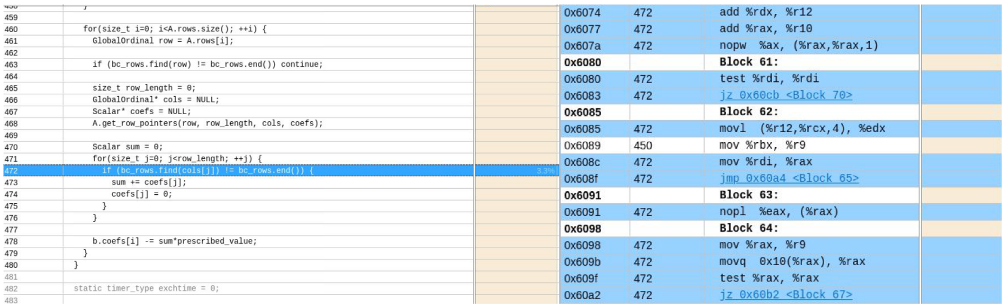

Figure 1: Performance bottleneck as identified during hotspot analysis in the sparse matrix multiplication module

2.3 Synthesis workflow for Neuromorphic predictor

Towards the end of the semester, I had the opportunity to work on developing and deploying the synthesis flow for Neuromorphic predictor. The designs were run for 2GHz and 4GHz, analysed for power and gate count.

```
read_libs {../Downloads/Nangate15FreePDK/libs/NanGate_15nm_OCL_typical_conditional_nldm.lib }
read_physical -lef {../Downloads/Nangate15FreePDK/lefs/NanGate_15nm_OCL.tech.lef ../Downloads/Nangate15FreePDK/lefs/NanGate_15nm_OCL.macro.lef }
read_hdl " src/forward_64x8.v "
elaborate
check_design
check_design -unresolved
read_sdc time.sdc
syn_generic
syn_map
syn_opt
report_timing
report_timing > timing.log
report_timing
report_timing -n 5
report_power
report_power > power.log
write_db dtmf_recvr_core -to_file design.db
write_db forward64x8 -to_file design.db
write_db forward_64x8 -to_file design.db
write_hdl > fwd64x8syn.v
exit
```

Figure 2: The synthesis flow ensures the design elaborates without any errors and reports timing, power and area.

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 18.11-s009_1
  Generated on:           Jul 28 2021  11:28:05 pm
  Module:                 forward_64x8
  Operating conditions:   typical
  Interconnect mode:      global
  Area mode:              physical library
============================================================

   Gate     Instances    Area        Library
--------------------------------------------------
AND2_X1          6210  1831.404    NanGate_15nm_OCL
AND3_X1            16     6.291    NanGate_15nm_OCL
AND3_X2             4     1.573    NanGate_15nm_OCL
AOI21_X1           37    10.912    NanGate_15nm_OCL
AOI21_X2            6     2.654    NanGate_15nm_OCL
AOI22_X1          114    39.223    NanGate_15nm_OCL
AOI22_X2            3     1.769    NanGate_15nm_OCL
BUF_X4              1     0.393    NanGate_15nm_OCL
FA_X1            5615  6623.724    NanGate_15nm_OCL
HA_X1             143    91.374    NanGate_15nm_OCL
INV_X1            335    49.398    NanGate_15nm_OCL
INV_X2             12     2.359    NanGate_15nm_OCL
INV_X4              4     1.180    NanGate_15nm_OCL
MUX2_X1            31    19.808    NanGate_15nm_OCL
NAND2_X1          170    33.423    NanGate_15nm_OCL
NAND3_X1            7     2.064    NanGate_15nm_OCL
NAND4_X1            5     1.720    NanGate_15nm_OCL
NOR2_X1           146    28.705    NanGate_15nm_OCL
NOR2_X2             2     0.590    NanGate_15nm_OCL
NOR3_X1             2     0.590    NanGate_15nm_OCL
```

```
Path 1: MET (0 ps) Setup Check with Pin outputsum_r_reg[11]/CLK->SE
          Group: clk
     Startpoint: (R) inputweights[756]
          Clock: (R) clk
       Endpoint: (R) outputsum_r_reg[11]/SE
          Clock: (R) clk

                    Capture       Launch
     Clock Edge:+     500           0
     Drv Adjust:+       0           0
    Src Latency:+       0           0
    Net Latency:+       0 (I)       0 (I)
        Arrival:=     500           0

          Setup:-      12
  Required Time:=     488
   Launch Clock:-       0
    Input Delay:-       0
      Data Path:-     488
          Slack:=       0
```

Figure 3: The gate analysis shows that the tool is primarily biased towards AND gates and full adders. The timing can be further optimized by increasing the flow effort

3 Future Work

In addition to finishing a more detailed analysis based on cache analysis for MiniFE application, I am interested in exploring some other areas such as quantum and adiabatic computing. Another area that interests me lies at the intersection of distributed systems and cryptography. Having worked on circuit and processor block level innovation, I wish to expand my scope vertically upwards in the computing stack.

References

[1] Jianhua Gao, Weixing Ji, Zhaonian Tan, and Yueyan Zhao. A systematic survey of general sparse matrix-matrix multiplication, 2020.

[2] Kadir Akbudak, Oguz Selvitopi, and Cevdet Aykanat. Partitioning models for scaling parallel sparse matrix-matrix multiplication. 4(3), January 2018.

[3] Junhong Liu, Xin He, Weifeng Liu, and Guangming Tan. Register-aware optimizations for parallel sparse matrix—matrix multiplication. *Int. J. Parallel Program.*, 47(3):403–417, June 2019.

[4] Grey Ballard, Christopher Siefert, and Jonathan Hu. Reducing communication costs for sparse matrix multiplication within algebraic multigrid. *SIAM Journal on Scientific Computing*, 38(3):C203–C231, 2016.

[5] Grey Ballard, C. Siefert, and Jonathan J. Hu. Reducing communication costs for sparse matrix multiplication within algebraic multigrid. *SIAM J. Sci. Comput.*, 38, 2016.

[6] Cagri Balkesen, Gustavo Alonso, Jens Teubner, and M. Tamer Özsu. Multi-core, main-memory joins: Sort <i>vs.</i> hash revisited. *Proc. VLDB Endow.*, 7(1):85–96, September 2013.

[7] Yuedan Chen, Kenli Li, Wangdong Yang, Guoqing Xiao, Xianghui Xie, and Tao Li. Performance-aware model for sparse matrix-matrix multiplication on the sunway taihulight supercomputer. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):923–938, 2019.

[8] Mehmet Deveci, Simon D. Hammond, Michael M. Wolf, and Sivasankaran Rajamanickam. Sparse matrix-matrix multiplication on multilevel memory architectures : Algorithms and experiments, 2018.

[9] Mehmet Deveci, Christian Trott, and Sivasankaran Rajamanickam. Multi-threaded sparse matrix-matrix multiplication for many-core and gpu architectures, 2018.

[10] Martin Winter, Daniel Mlakar, Rhaleb Zayer, Hans-Peter Seidel, and Markus Steinberger. Adaptive sparse matrix-matrix multiplication on the gpu. PPoPP '19, page 68–81, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Mohammadreza Soltaniyeh, Richard P. Martin, and Santosh Nagarakatte. Synergistic cpu-fpga acceleration of sparse linear algebra, 2020.

[12] E. Jamro, Tomasz Pabis, P. Russek, and K. Wiatr. The algorithms for fpga implementation of sparse matrices multiplication. *Comput. Informatics*, 33:667–684, 2014.

[13] Reza Hojabr, A. Sedaghati, A. Sharifian, A. Khonsari, and Arrvindh Shriraman. Spaghetti: Streaming accelerators for highly sparse gemm on fpgas. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 84–96, 2021.