RESEARCH REVIEW 2020

Spiral AI/ML: Co-optimization for High-Performance, Data-Intensive Computing in Resource Constrained Environments

Problems

- The need exists for increased computational power to process, exploit, and disseminate information for decision makers.
- Massive amounts of information, along with AI/ML algorithms, generate data and computational-intensive applications.
- Implementing these applications efficiently on increasingly complex HW/SW architectures is challenging.
- Too few engineers have the expertise to optimize algorithms for the wide variety of hardware currently available.

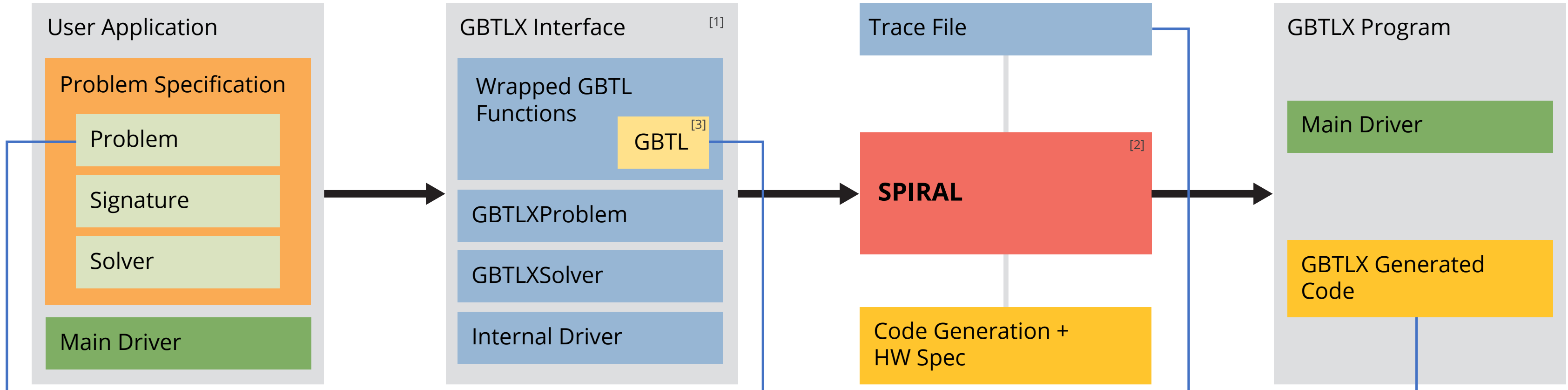
Solution

- Automatic code generation for data-intensive computations
- Simultaneous, automatic co-optimization for targeted hardware

Approach

- Identify and encode data-intensive compute primitives into CMU's SPIRAL code generation technology.
- Develop and encode hardware performance models into Spiral.
- Use Spiral to co-optimize for a set of target hardware platforms.

Hardware-software co-optimization promises timely, high-performance, and cost-effective implementation and re-implementation of AI/ML workloads on new **DoD** hardware platforms.



Graph algorithms in the language of linear algebra supports a rich notation for specifying graph, ML and AI algorithms. For example, counting triangles in graph **L**:

$$\Delta = \| \mathbf{L} .x (\mathbf{L} +. \wedge \mathbf{L}) \|$$

includes use of semiring algebraic operations and masked matrix multiplies.

[6]

GBTL implements the GraphBLAS specification that allows simpler implementation of the math in code:

```
uint64 _ t triangle _ count(Matrix<bool> const &L) {
    Matrix<uint64 _ t> B(L. nrows(), L.ncols());

    // Masked matrix multiply: B = L .* (L +.^ L)
    mxm(B, L, NoAccum(), PlusAndSemiring<uint64 _ t>(), L, L);

    //Perform reduction: ||B||
    uint64 _ t count;
    reduce(count, NoAccum(), PlusMonoid<uint64 _ t>(), B);
    return count;
}
```

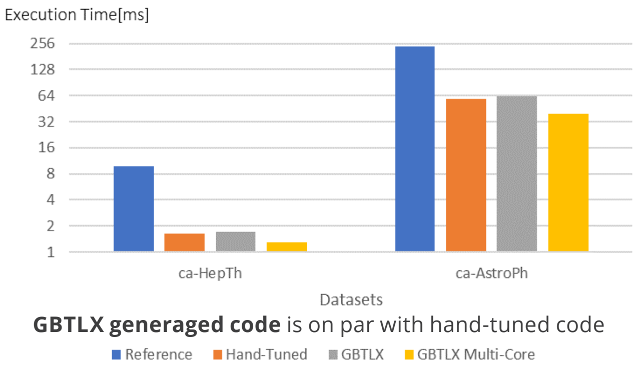
[3,4]

Spiral wraps GBTL functions to build a trace file used for analysis during code generation:

```
spiral _ session := [
    rec(op := "triangle _ count"), //function name
    rec(op := "MatrixCreation",row:= 9877,col:= 9877,
        ptr := 0x7fffff45bb60, mat = 0x7fffff45bb60),
    rec(op := "Matrix Multiplication",
        output = IntHexString("0x7fffff45bb60") ,
        mask   = IntHexString("0x7fffff45ba30"),
        inputA = IntHexString("0x7fffff45bb30"),
        inputB = IntHexString("0x7fffff45bb30"),
        semiring = "PlusAnd"),
    rec(op := "reduce(matrix->scalar)",
        /*many more arguments*/),
];
```

[1]

Performance of GBTLX on Triangle Counting



GBTLX generaged code is on par with hand-tuned code

[1,5]

References

1. S. Rao, A. Kutuluru, S. McMillan, F. Franchetti, "GBTLX: A First Look", in 2020 IEEE High Performance Extreme Computing Conference (HPEC), 2020. ***Outstanding Student Paper Award.***
2. SPIRAL Project, Version 8.1.2. Available at **https://www.spiral.net.**
3. GraphBLAS Template Library (GBTL), Version 3.0. Available at **https://github.com/cmu-sei/gbtl**, June 2020.
4. A. Buluç, T. Mattson, S. McMillan, J. Moreira, and C. Yang, "Design of the GraphBLAS API for C," in 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 643–652, 2017.
5. T. M. Low, V. N. Rao, M. Lee, D. Popovici, F. Franchetti, and S. McMillan, "First look: Linear algebra-based triangle counting without matrix multiplication," in2017 IEEE High Performance Extreme ComputingConference (HPEC), pp. 1–6, 2017.
6. J. Kepner, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, A. Lumsdaine, T. Mattson, S. McMillan, et al., "Mathematical Foundations of the GraphBLAS," in 2016 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–9, 2016.

Carnegie Mellon University
Software Engineering Institute

Dr. Scott McMillan (SEI PI), Prof. Franz Franchetti (CMU PI), Prof. Tze Meng Low (CMU PI),
Dr. Daniele Spampinato, Mark Blanco, Anurag Kutuluru, Sanil Rao, Upasana Sridhar
info@sei.cmu.edu