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.

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

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 $L$:

$$\Delta = ||L.X((L + A)L)||$$

includes use of semiring algebraic operations and masked matrix multiplies.

Spiral wraps GILT 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*/),
];
```

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

```c
uint64_t triangle_count(Matrix<bool> const &L) {
    Matrix<uint64_t> B(L.nrows(), L.ncols()), L.nncns();
    // Masked matrix multiply: B = L * (L + A)L
   (mask(L, NoAccum()), PlusAndSemiring<uint64_t> t3)(B, L);
    //Perform reduction: ||B||
    uint64_t count;
    reduce(count, NoAccum(), PlusMonoid<uint64_t>(), B);
    return count;
}
```

References