DSLs, DLA, DxT, and MDE in CSE

Bryan Marker, Robert van de Geijn, Don Batory
The University of Texas at Austin

I'M ASSUMING SOME CSE KNOWLEDGE, SO ASK QUESTIONS

Glossary

- DLA – dense linear algebra
  - Often found at the bottom of a CSE software stack
  - Often leads the way in programming models since it’s such an “easy” domain
  - Has to be re-visited with every major architecture shift
- DSL – domain-specific language
  - Enables experts to write algorithms at a level of abstraction that makes them effective in producing (hopefully) high-performance code
  - Could just be an API provided by a library
- MDE – model driven engineering
  - Models represent (software) systems
  - Can start with an abstract design and iteratively add implementation details
  - Encode knowledge about how to implement domain (software) components

The Problem

- Different DSLs are needed for each architecture
  - GPU code won't work well for shared-memory or distributed-memory or …

- When a new architecture comes out, experts must revisit all common domain operations, revisit all of their algorithms, and code them for the new target

- Experts are rare, so their time is valuable
  - So much of what they do is rote development by applying their knowledge repeatedly
  - Why are they doing it all by hand?
  - Let's automate this!

Design by Transformation

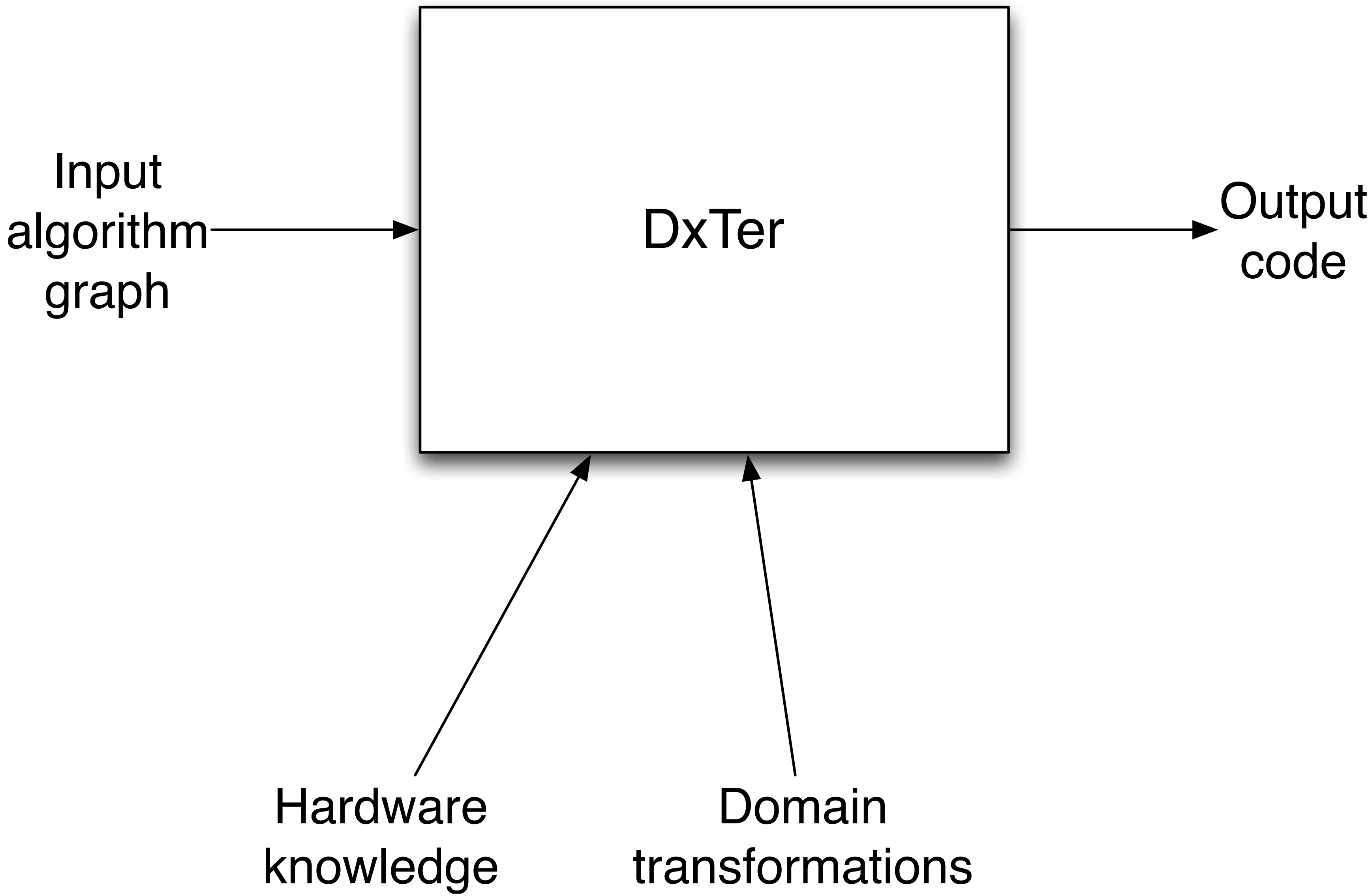
- Design by Transformation (DxT) for automatic program generation
- Encode domain algorithms as models / data flow graphs

- Nodes represent functionality
  - An interface has no implementation details (works for any architecture)
  - A primitive has an implementation in DSL code for the target architecture
- Start with a graph of all interfaces and end with a graph of all primitives

- Encode expert design knowledge as graph transformations
  - Iteratively replace interfaces with implementations (refinement)
  - The result is functional code
  - Iteratively replace inefficiencies with better code (optimization)
  - The result is high-performance code

DxT

- Basically, the system searches a space of implementation choices, just like an expert, but it does it automatically so an expert can relax
- Our prototype is called DxTer
    - Input graph, get DSL code for particular target


Input
algorithm
graph
DxTer
Output
code
Hardware
knowledge
Domain
transformations

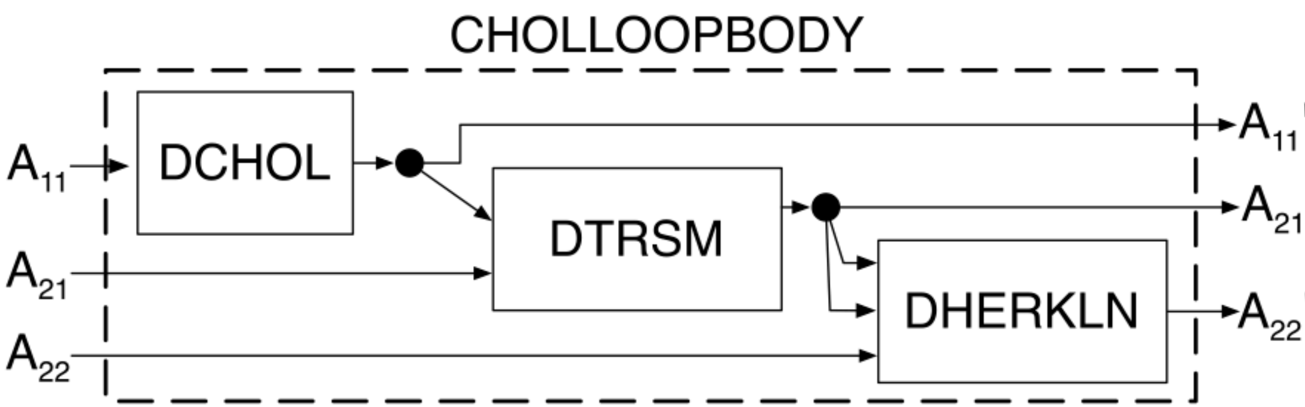
DxT for DLA

- Automatically generating code for distributed memory
- Targeting Elemental library
  - Modern (C++, object-oriented) replacement for ScaLAPACK

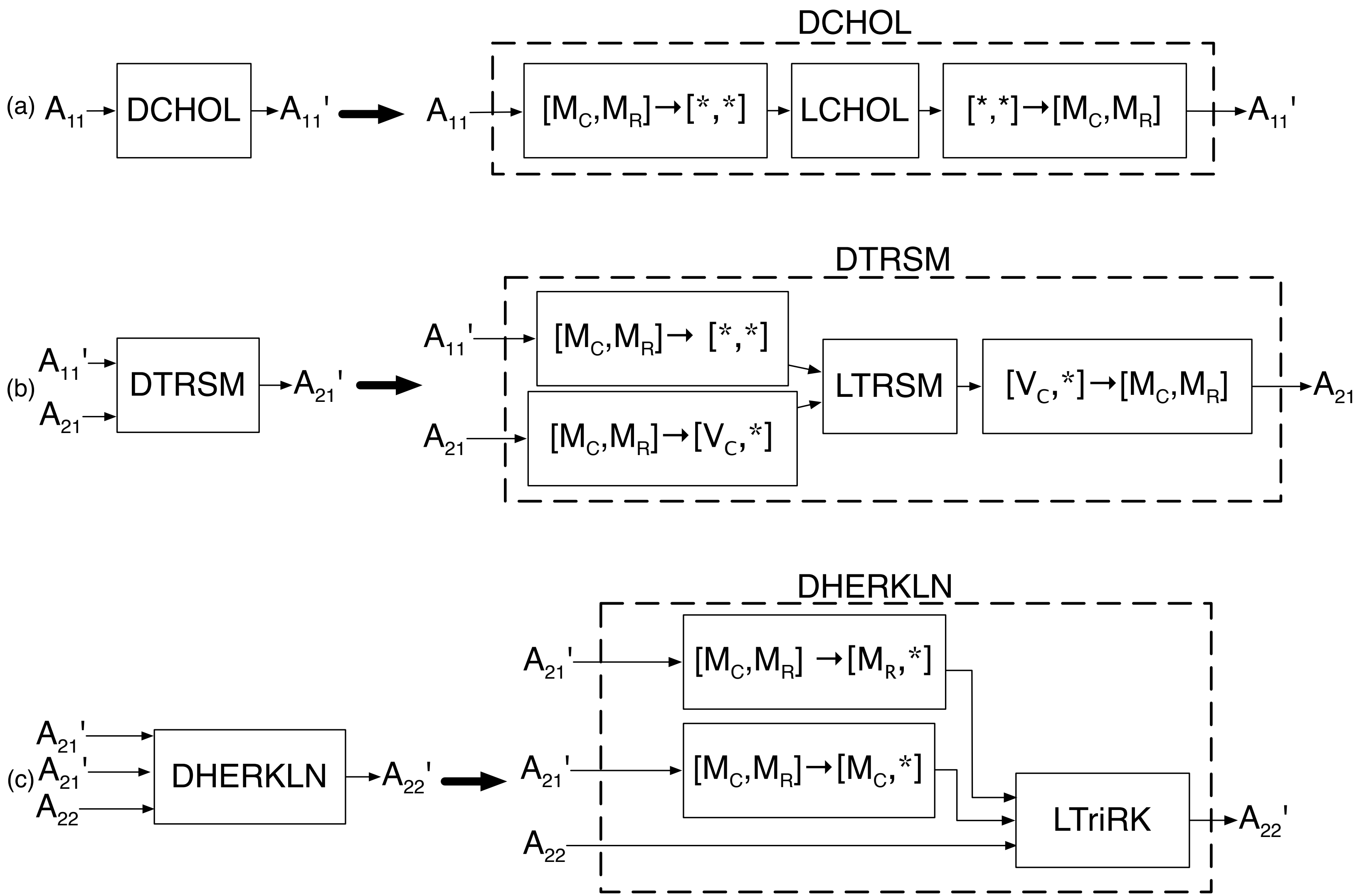
- In all cases, generated same or better than an expert
    - Experts forget algorithms or optimizations
    - Experts make coding errors
    - DxTer does not

- Code runs significantly faster than ScaLAPACK

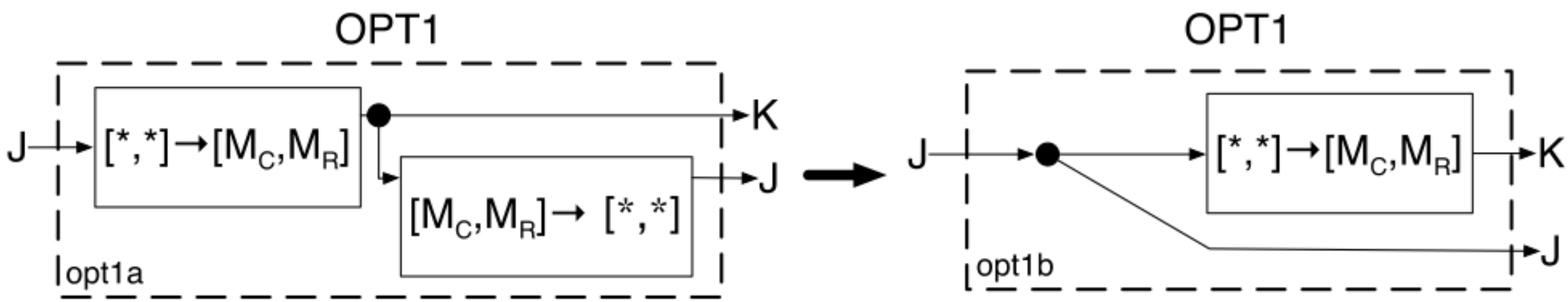
View as DAG
Notice that this is hardware-agnostic



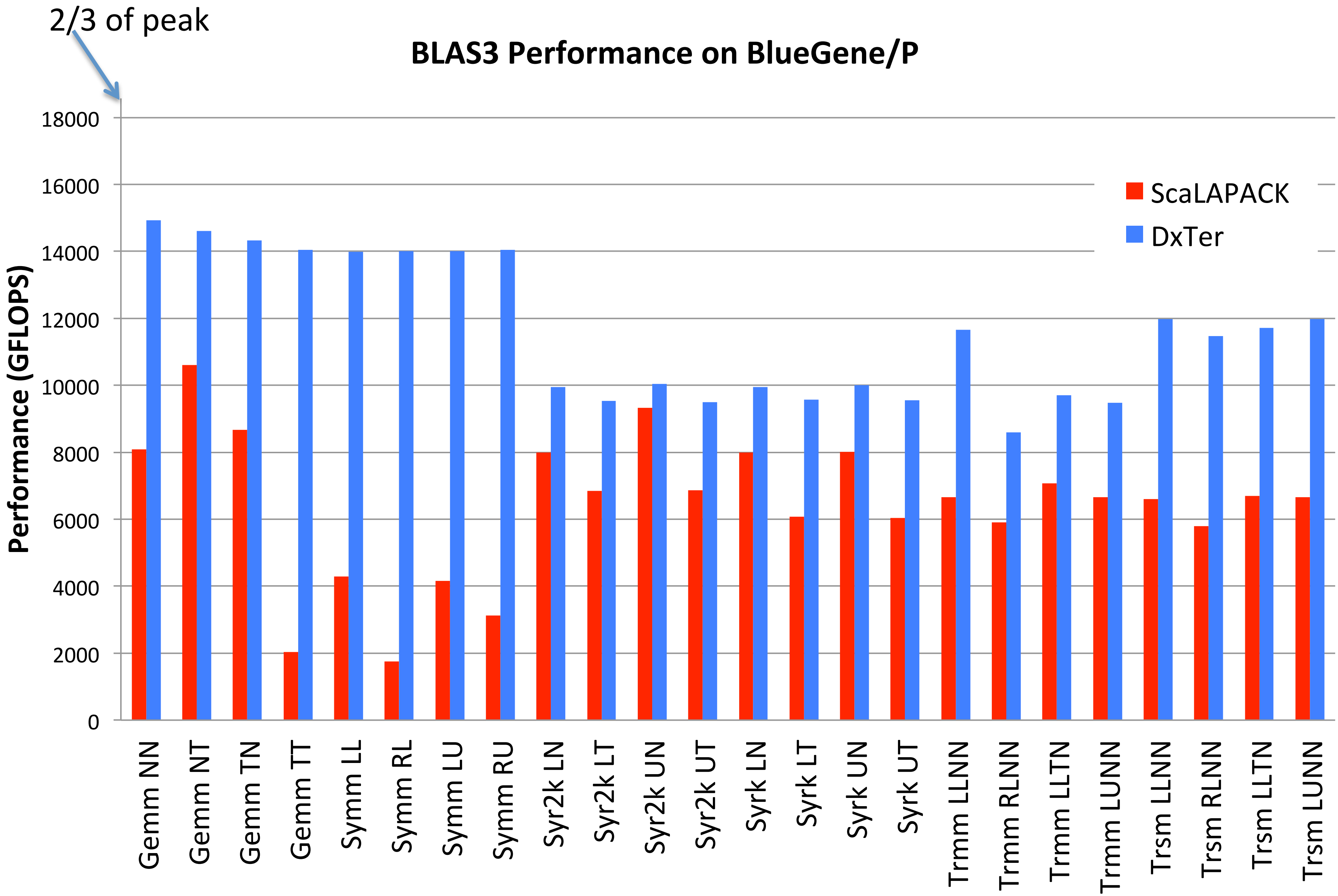
Transform with Implementations



Transform to Optimize



WHO KNOWS OF THE BLAS?



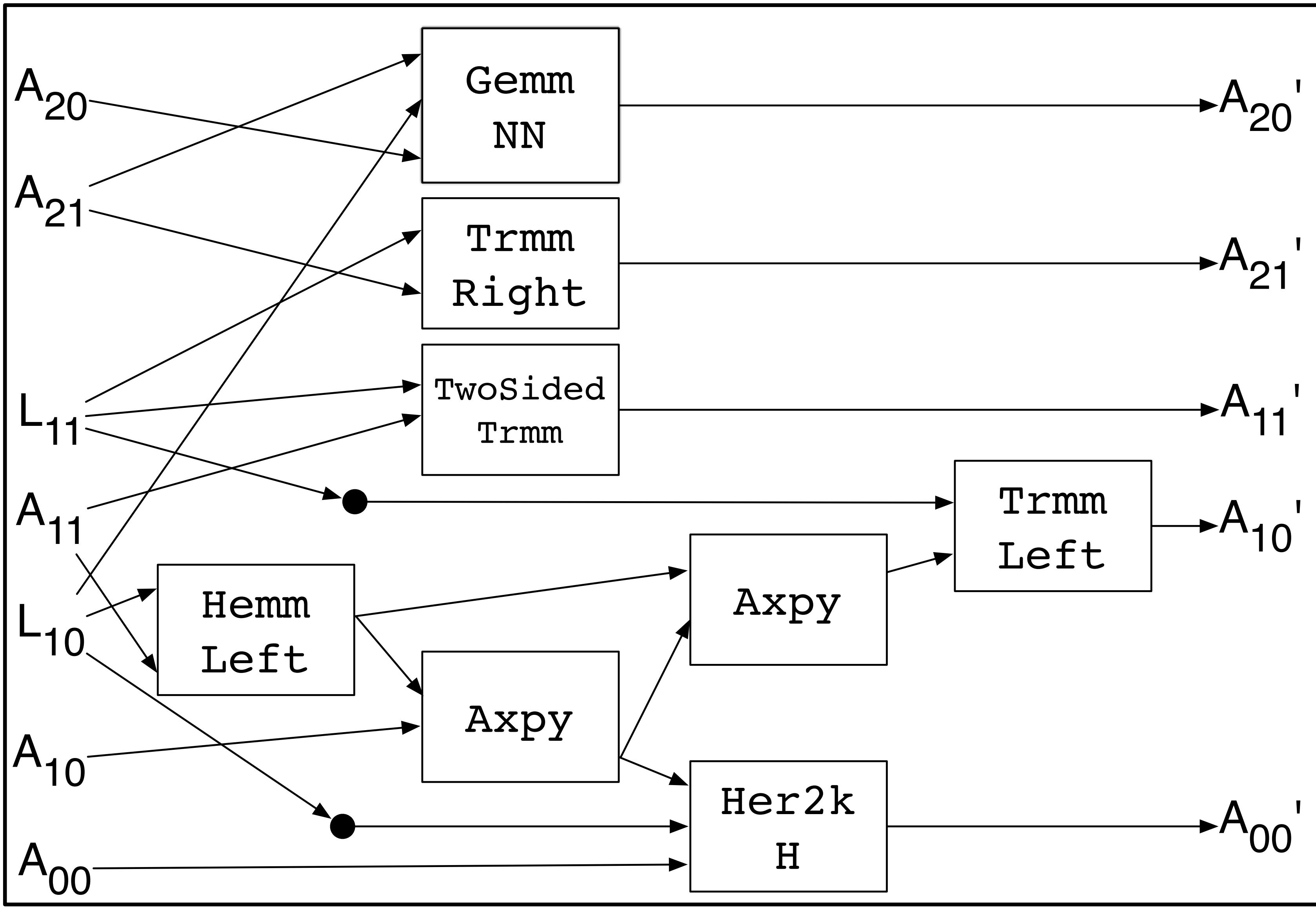
*8,192 cores on Argonne's Intrepid machine

Building Blocks

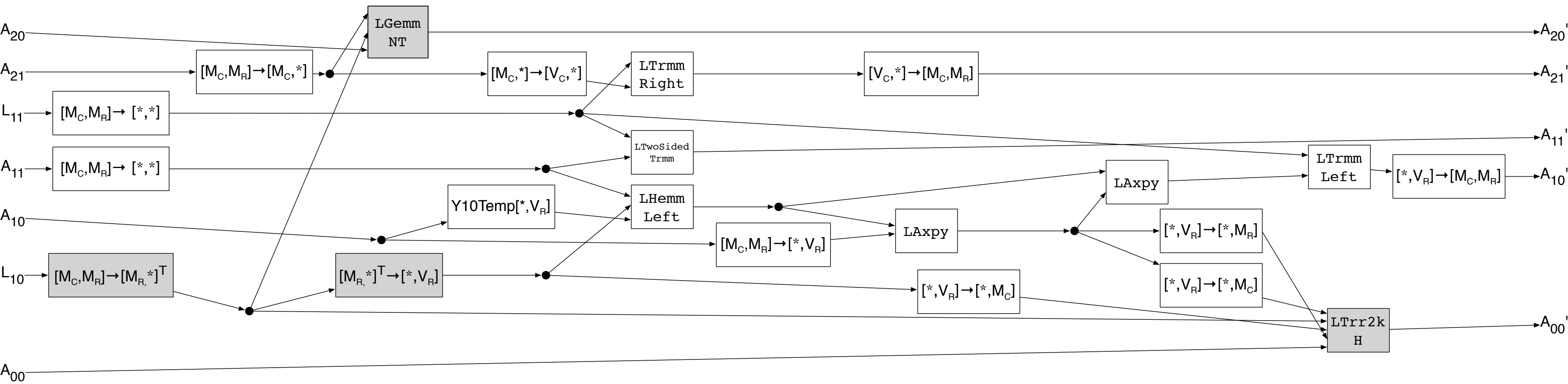
- The knowledge to generate that code forms a set of domain building blocks
  - The BLAS are at the bottom of DLA software stacks

- More complicated algorithms use that knowledge
    - When done by hand, it's rote re-application of knowledge
    - When done by DxTer, who cares?

Starting Graph

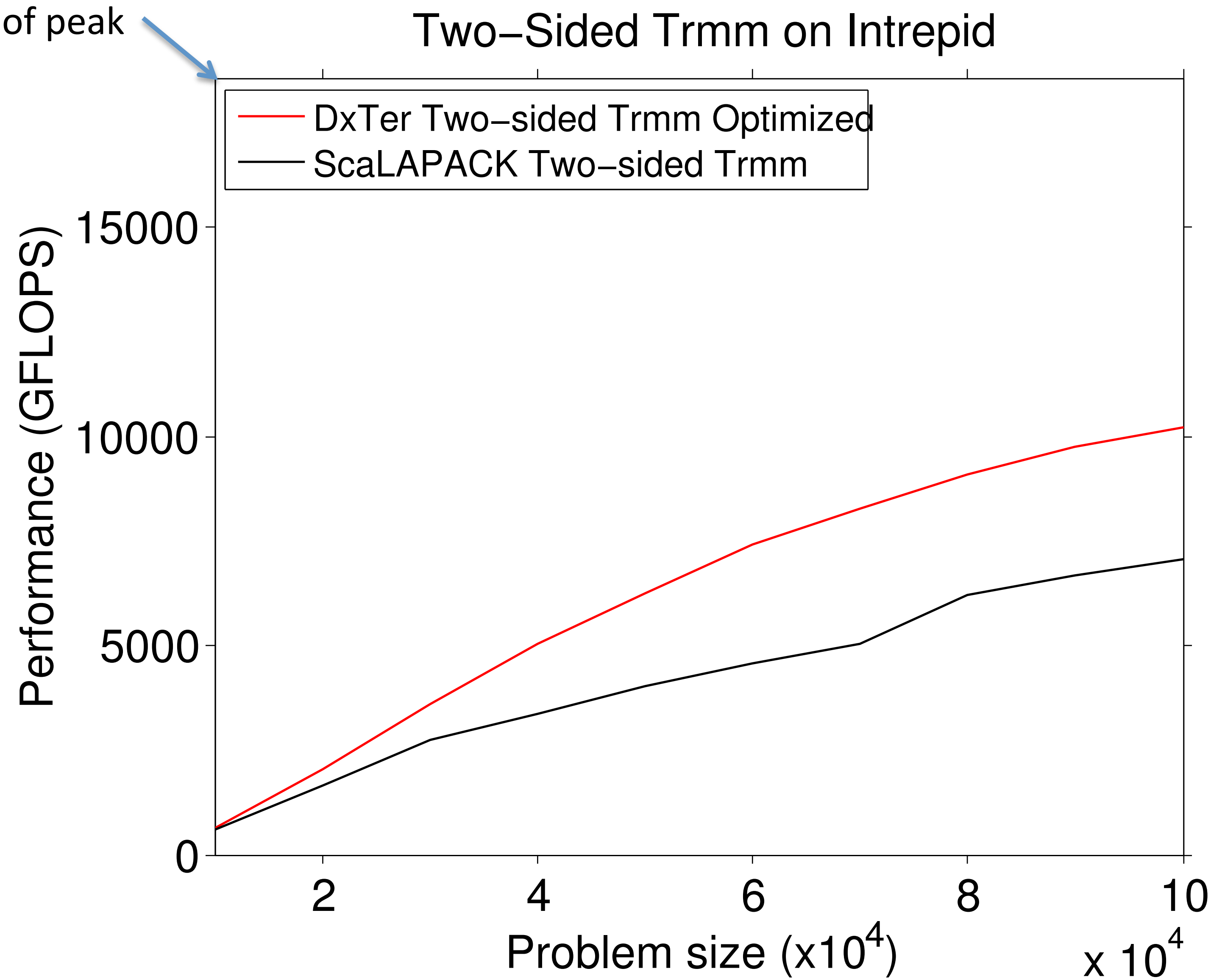


Final Implementation



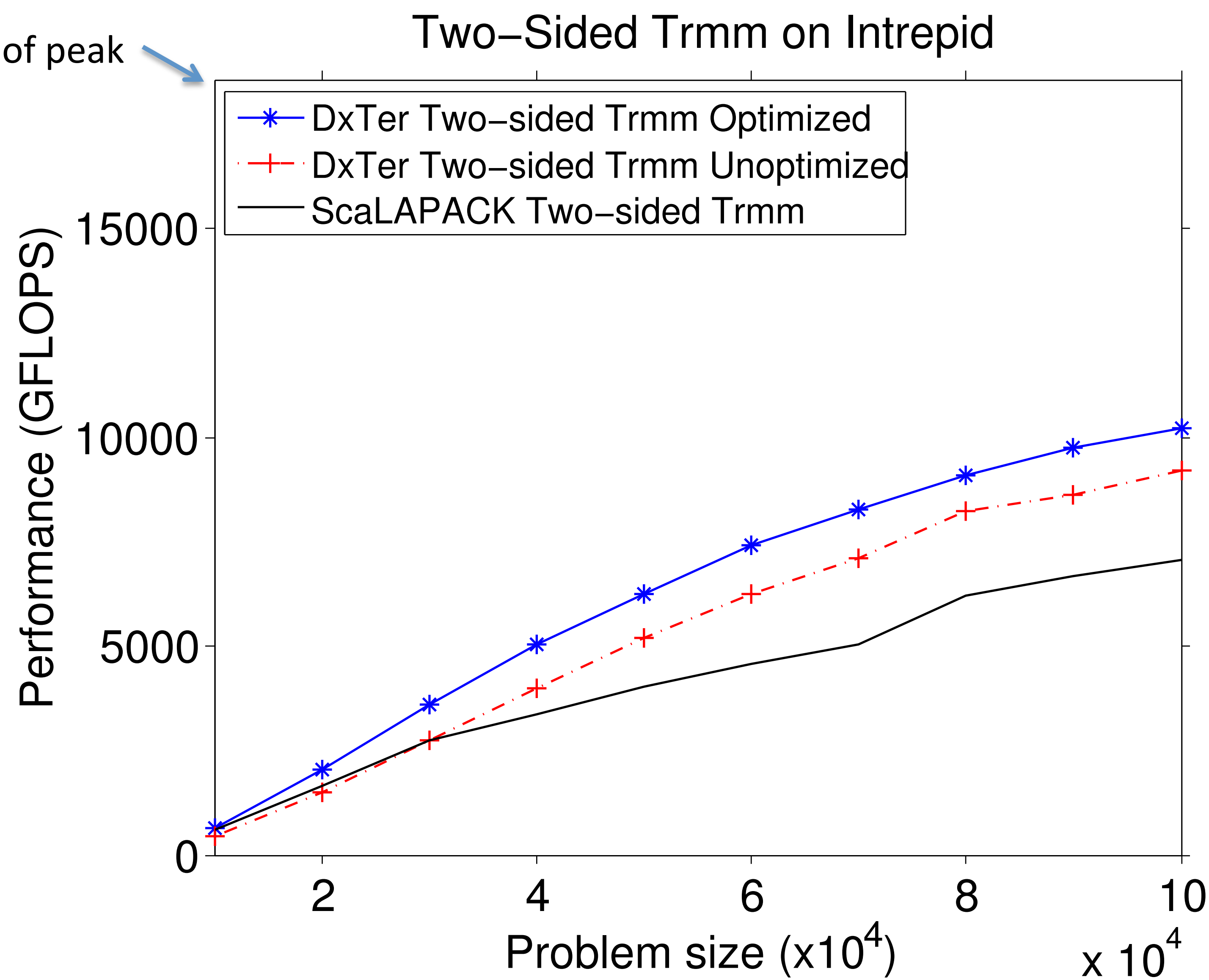
DxT

2/3 of peak

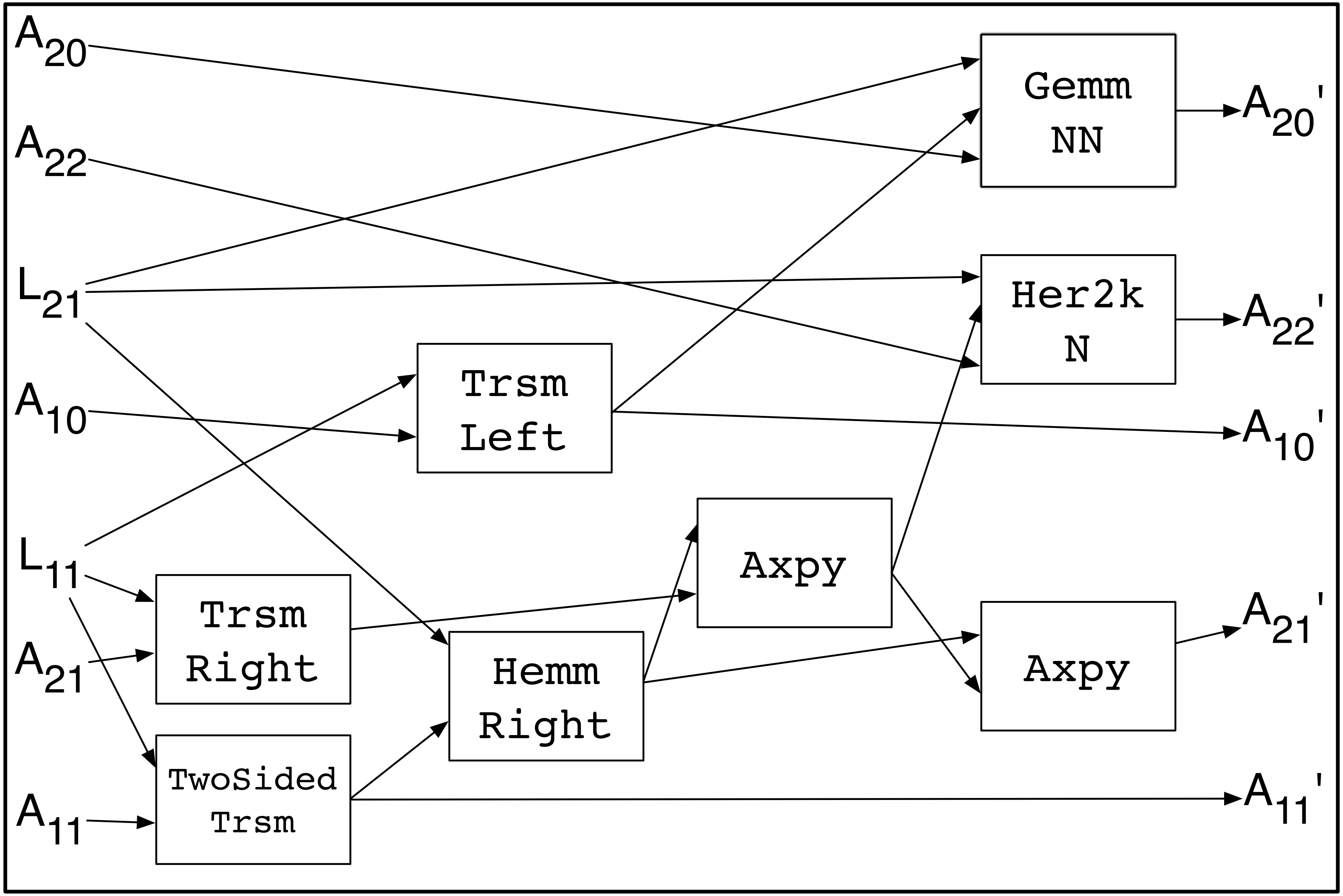


*8,192 cores on Argonne’s Intrepid machine

2/3 of peak



*8,192 cores on Argonne's Intrepid machine



A20
A22
L21
A10
L11
A21
A11
Gemm NN
Her2k N
Trsm Left
Axpy
Axpy
Trsm Right
Hemm Right
TwoSided Trsm
A20'
A22'
A10'
A21'
A11'

How Can We Do This?

- Requires DEEP domain knowledge
    - Without domain understanding, we can’t do what experts do

- Requires software layering
    - Need to be able to abstract key domain ideas and functionality
    - DSLs are great at hiding minutia of domain
    - Enable people to focus on important decisions
    - Enables us to encode important knowledge

- We’re not encoding knowledge for arbitrary C++ programs

Moving Forward

- Many CSE domains similarly have experts doing rote work
    - Implementing similar (but sufficiently different) algorithms repeatedly for one architecture
    - Re-implementing the same algorithms for a new hardware target

Moving Forward

- Let's work towards encoding expert knowledge and automating the tedious part of the expert's job

- Let's work toward getting the human out of the software development cycle
    - Better performing code
    - More trustworthy code
    - Faster development times
    - More scientific approach to software engineering (encoding knowledge/ patterns of domain instead of resulting code)

Questions?

bamarker@cs.utexas.edu