Domain-specific programming methodologies for domain-specific and emerging computing systems

Jeronimo Castrillon
Chair for Compiler Construction (CCC)
TU Dresden, Germany

San Diego CA, USA
June 14, 2022
Evolution of computing: Breaking walls

- Single-core architectures
- Use multi-core architectures (~2005)
- Dark Si: specialize
- Post CMOS

- Massive parallel and heterogeneous systems
- Specialization: TPUs, AI engines, Bio-inf, ...
- Novel interconnects & distributed computing
- Emerging memories & in-memory computing

Evolution of computing: Breaking walls

- single-core architectures
- Use multi-core architectures
- Dark Si: specialize

~2005

- Massive parallel and heterogeneous systems
- Specialization: TPUs, AI engines, Bio-inf, ...
- Novel interconnects & distributed computing
- Emerging memories & in-memory computing


https://www.hpcwire.com/2017/04/10/nvidia-responds-google-tpu-benchmarking/

Samsung @ ISCA 2021

© Prof. J. Castrillon. Keynote, LCTES 2022
The golden era in computer architecture requires major changes in programming (toll-free abstractions) to democratize heterogeneous and emerging computing across domains. 

Need for higher-level programming abstractions and next-gen compilers as well as novel computational and costs models for emerging accelerators.
Why new abstractions?

What we want

What we (naively) code

What compilers see
**Semantic gap ➔ performance gap**

\[
v_{ijk,e} = \sum_{i'=0}^{p} \sum_{j'=0}^{p} \sum_{k'=0}^{p} A_{kk'} A_{jj'} A_{ii'} u_{i'j'k'} e
\]

What we want

What we (naively) code

What performance experts code

© Prof. J. Castrillon. Keynote, LCTES 2022
Semantic gap $\Rightarrow$ performance gap

What we want

What we (naively) code

100X

What performance experts code

https://www.hpcwire.com/2017/04/10/nvidia-responds-google-tpu-benchmarking/

Example for tensors expressions (CFD, ML)

- Expression-language for tensor operations and optimizations
- Originally for spectral element methods in computational fluid dynamics

\[ v_e = (A \otimes A \otimes A) u_e \]

Interpolation kernel

Fortran and C++ integration


© Prof. J. Castrillon. Keynote, LCTES 2022
Closing the performance gap

- Not really optimization magic
- Leverage expert knowledge
- Algebraic identities

\[
v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot (A_{jm} \cdot (A_{il} \cdot u_{lmn}))
\]

\[
v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot A_{jm}) \cdot (A_{il} \cdot u_{lmn})
\]

\[
v_{ijk} = \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))
\]

Closing the performance gap

- Not really optimization magic
  - Leverage expert knowledge
  - Algebraic identities

\[
\begin{align*}
v_{ijk} &= \sum_{l,m,n} (A_{kn} \cdot (A_{jm} \cdot (A_{il} \cdot u_{lmn}))) \\
v_{ijk} &= \sum_{l,m,n} (A_{kn} \cdot A_{jm}) \cdot (A_{il} \cdot u_{lmn}) \\
v_{ijk} &= \sum_{l,m,n} (A_{kn} \cdot ((A_{jm} \cdot A_{il}) \cdot u_{lmn}))
\end{align*}
\]


Easy to generate, hard to transform

Actual code variants
TeML: Meta-programming for tensor optimizations

- Generalize for cross-domain tensor expressions
- Formal semantics and composition of transformations

Formally defined transformation primitives

Higher-level transformations via composition

Cross-domain tensor optimization

Performance of Pluto could be reproduced

Higher abstraction \(\rightarrow\) more optimization potential

TelL: Formal language – added value

- Core common to multiple tensor languages
- Index-free notation and strong type system
- **Provably** no out-of-bound accesses

```python
A = placeholder(((m,h), name='A'))
B = placeholder((n,h), name='B'))
k = reduce_axis((0, h), name='k')
C = compute((m, n), lambda i, j:
    sum(A[k, i] * B[k, j], axis=k))
```

TeIL in MLIR

- Primitive ops instead of index maps
- Easier to express identities (big-O trfs)
- Uses symbolic math, infinite precision

Scalar types
- ScIL provides scalar operators
- ScIL provides Rationals, Neutrals, …
- Base2 provides parametric binary number types
- Based2 models (custom) hardware
Multi-level lowering with MLIR

- Encoding transformations

%y = teil.contr 6 9 %x : ...
%1 = teil.prod %S %S : ...
%2 = teil.prod %1 %S : ...
%3 = teil.prod %2 %u : ...
%y = teil.red "add" 6 %3

%1 = teil.diag 6 9 %x : ...
%y = teil.red "add" 6 %1 :

%1 = teil.prod %S %u : ...
%2 = teil.diag 2 5 %1 : ...
%3 = teil.red "add" 2 %2 : ...
%4 = teil.prod %S %3 : ...
%y = teil.prod %S %4 :

%b = linalg.tensor_reshape %B [(i,j,k)->(i), (i,j,k)->(j,k)] ...
linalg.matmul
ins(%A, %b : ...) outs(%y : ...) ...
Domain-specific optimization

- Encode algebraic transformations (Interpolation as example)
- Direct feedback to expert via DSL export

\[ t = (S \otimes S \otimes S \otimes u)_{xyz}^{by} \]
FPGA code generation: Bus-attached FPGAs

- H2020 EU Project: Convergence HPC, Big Data and ML
- Inverse Helmholtz Kernel

\[
\begin{align*}
    v_e &= (S \otimes S \otimes S) D_e^{-1} (S^T \otimes S^T \otimes S^T) u_e \\
    t &= S \# S \# S \# u \ [1] \\
    r &= D \ast t \\
    v &= S \# S \# S \# r \ [0]
\end{align*}
\]

Lifetime analysis (polyhedral analysis)

Menosyne
mem-subsystem gen (buffer sharing)

FPGA code generation: Bus-attached FPGAs

- H2020 EU Project: Convergence HPC, Big Data and ML
- Exploring configurations on small FPGAs
- Example on Inverse Helmholtz kernel


https://everest-h2020.eu
FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- HBM-FPGA and Cloud FPGA (ongoing)

FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- HBM-FPGA and Cloud FPGA (ongoing)


https://everest-h2020.eu
Emerging data-centric architectures

- Compute (almost) in-place, avoid data movement, transformations to match primitives
- Novel architectures for near-memory and in-memory computing


Emerging memories and in-memory computing

- Compute in-place, avoid data movement, transformations to match primitives

**In-PCM Computing**


**In-RTM Computing**


Emerging memories and in-memory computing

- Compute in-place, avoid data movement, transformations to match primitives

**In-PCM Computing**

Crossbar of memristive devices

- **What are primitives in in-memory computing?**
  - In-PCM *dot-products*, non-linear functions, ...
  - In-RTM bulk logic and *majority operations*, efficient counting, ...
  - Others: (approx.) *content-addressable memories* (FeFETs, ...)

**In-RTM Computing**


End-to-end compiler for PCM acceleration

- MLIR frontend for general tensor expressions
  - Reuse GEMM transformations from \texttt{linalg} (in MLIR)
  - High-level transformations and lowering to \texttt{CIM dialect}

---

Lowering examples

```python
def contr(int16(K, L, M) A, int16(L, K, N) B) -> (int16(M, N) C)
{
    C(m, n) += A(k, l, m) * B(l, k, n)
}

\[\downarrow\] lowers to
%0 = linalg.transpose(A, {2, 0, 1})
%1 = linalg.transpose(B, {1, 0, 2})
%2 = linalg.reshape(%0, {0, {1, 2}})
%3 = linalg.reshape(%1, {{0, 1}, 2})
// eligible for offloading to CIM
linalg.matmul(%2, %3, %C)

// loop interchanged GEMM
scf.for %k = %c0 to %numTiles step %c1 {
    scf.for %j = %c0 to %tiledCols step %c1 {
        %tileB = cim.copyTile(%B, %k, %j)
        cim.write(%id, %tileB)
        scf.for %i = %c0 to %tiledRows step %c1 {
            %tileC = cim.copyTile(%C, %i, %j)
            . . .
            cim.storeTile(%tileC, %C, %i, %j)
        }
    }
}
```

```c
linalg.matmul(%A, %B, %C)
\[\downarrow\] lowers to
// tiled GEMM in the CIM dialect
%c0 = constant 0 : i32
%c1 = constant 1 : i32
%id = constant 0 : i32 // tile id
scf.for %i = %c0 to %tiledRows step %c1 {
    scf.for %j = %c0 to %tiledCols step %c1 {
        %tileC = cim.copyTile(%C, %i, %j)
        %tempTile = cim.allocDuplicate(%tileC)
        scf.for %k = %c0 to %numTiles step %c1 {
            %tileA = cim.copyTile(%A, %i, %k)
            %tileB = cim.copyTile(%B, %k, %j)
            cim.write(%id, %tileB)
            cim.matmul(%id, %tileA, %tempTile)
            cim.barrier(%id)
            // tileC += tempTile
            cim.accumulate(%tileC, %tempTile)
        }
    }
}
```
Optimization results: Beyond Matmult
Latency highly depends on allocation and address traces
- System-level simulators (interoperable w/ e.g. gem5)
- Compiler optimizations for scalars, arrays and instructions

Khan, et al. “ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0”, ACM TACO 2019
Tensor expressions on RTMs

- Consecutive accesses can be pre-shifted
- Avoid “rewinding the tape”
Latency comparison vs SRAM

- Un-optimized and naïve mapping: Even worse latency than SRAM
- 24% average improvement (even with very conservative circuit simulation)


Higher savings due to less leakage power
74% average improvement (in addition to savings due to DRAM placement)

A. A. Khan, et al,  "Optimizing Tensor Contractions for Embedded Devices with Racetrack Memory Scratch-Pads", LCTES’19, pp. 5-18, 2019
Generalization: RTM optimizing compiler

- Jointly optimize layout and operation scheduling
- Interplay with other (polyhedral) loop optimizations: Tiling, fusion, ...

Example: Optimizations for stencils for RTM

- Average improvements in performance (~20%) and energy consumption (~40%)

Random forests

- Popular way for decision making @ edge
- Example of less-predictable access
- Use training statistics for tree placement

C. Hakert, “BLOwing Trees to the Ground: Layout Optimization of Decision Trees on Racetrack Memory”, DAC 2021
In-RTM computing

- Transverse reads
  - Pass current through nanowire (not for shifting)
  - Sensed resistance correlates with the amount of ones (group XOR)

- Studying how to generalize from this through hand-crafted designs…

K. Roxy, IEEE T Nano 2020
Example: Hyper dimensional computing (HDC)

- **HDC**: Embed data in 10 k-dimensions – Von-Neuman Bottleneck!

- Leverage bulk-wise binary operations

![Diagram of HDC process](attachment:image.png)

Example: Hyper dimensional computing (HDC)

Summary

- Tame ever-increasing system complexity
  - Domain-specific abstractions, compilation flows, ...
  - Example for tensor expressions
  - Optimization for CPU, in-memory, RTM placement, ...

- Challenges
  - Understanding and modeling primitives from down below
  - Simulators, prototypes in interdisciplinary research efforts
  - Optimization/DSE: ML? simpler heuristics useful again?
  - Joint work across stack layers will be key!

Thanks! & Acknowledgements

References


