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

Jeronimo Castrillon
Chair for Compiler Construction (CCC), TU Dresden
SCADS.AI Dresden/Leipzig & Center for Advancing Electronics (cfaed) Dresden

8th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)
SC23: The International Conference on High Performance Computing, Networking, Storage and Analysis, Denver, USA
November 13, 2023
Evolution of computing: Breaking walls

- **Transistors (thousands)**
- **Performance**
- **Frequency (MHz)**
- **Typical power (W)**

**Core count**

- Single-core architectures
- Multi-core architectures
- Dark Si: specialize

Post CMOS, non Von Neumann


Emerging systems: Examples

- High-bandwidth memory
- AI accelerators + Prog. logic
- Near-memory computing

Extreme heterogeneity, non Von Neumann paradigms, custom number representations, custom data mapping, complex APIs, ...
### Abstractions and compilation

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

#### What we want

```c
void cfd_kernel(
    double A[restrict 7][7],
    double u[restrict 216][7][7][7],
    double v[restrict 216][7][7][7][7])
{
    /* element loop: */
    for(int e = 0; e < 216; e++) {
        for(int i0 = 0; i0 < 7; i0++) {
            for(int j0 = 0; j0 < 7; j0++) {
                for(int k0 = 0; k0 < 7; k0++) {
                    v[e][i0][j0][k0] += A[i0][i1] * A[j0][j1] * A[k0][k1] * u[e][i1][j1][k1];
                }
            }
        }
    } /* end of element loop */
}
```

#### What we (naively) code

```c
void cfd_kernel(
    double A[restrict 7][7],
    double u[restrict 216][7][7][7],
    double v[restrict 216][7][7][7][7])
{
    /* element loop: */
    for(int i0 = 0; i0 < 7; i0++) {
        for(int j0 = 0; j0 < 7; j0++) {
            for(int k0 = 0; k0 < 7; k0++) {
                v[e][i0][j0][k0] += A[i0][i1] * A[j0][j1] * A[k0][k1] * u[e][i1][j1][k1];
            }
        }
    } /* end of element loop */
}
```

#### What performance experts code

```c
void cfd_kernel(
    double A[restrict 7][7],
    double u[restrict 216][7][7][7],
    double v[restrict 216][7][7][7][7])
{
    /* element loop: */
    for(int i0 = 0; i0 < 7; i0++) {
        for(int j0 = 0; j0 < 7; j0++) {
            for(int k0 = 0; k0 < 7; k0++) {
                t9 = A[i0][i1] * A[j0][j1] * A[k0][k1] * u[e][i1][j1][k1];
                t10 = A[i0][i1] * A[j0][j1] * A[k0][k1] * u[e][i1][j1][k1];
                v[e][i0][j0][k0] += t9;
            }
        }
    } /* end of element loop */
}
```
What we want

Need for higher-level programming abstractions and next-gen compilers as well as novel computational and costs models for emerging accelerators.
The power of abstractions
Example: Particle-mesh simulations

- Particle-mesh simulations in computational biology
  - Discrete/continuous
  - Deterministic/stochastic

Syntax for interact, evolve, automatic insertion of interpolation, ghost sync., …

\[
\begin{align*}
\frac{\partial u}{\partial t} &= Du \ast \nabla^2 u - u \ast v^2 + F \ast (1 - u) \\
\frac{\partial v}{\partial t} &= Dv \ast \nabla^2 v + u \ast v^2 - v \ast (F + k)
\end{align*}
\]


Vortex ring
Semantic gap ➔ Debugging gap

- OpenFPM library
  - Modern C++ template library (for CPUs and GPUs)
  - Support for dynamic load-balancing, checkpointing and communication abstractions

- Template meta-programming

\[ \frac{D\omega}{Dt} = (\omega \cdot \nabla)u + \nu \Delta \omega \]

What we want

What we code (already quite abstracted!)


3D
Model-to-model code generation

OpenPME DSL

Intermediate representation (IR)

```
while (mloop_iterator_h5a0.isNext())
{
  g_dwp.template get<rhs>(key)[x] =
  fac1*(g_vort.template get<vorticity>(key.move(x,1))[x] +
  g_vort.template get<vorticity>(key.move(x,-1))[x] +
  g_vort.template get<vorticity>(key.move(y,1))[x] +
  g_vort.template get<vorticity>(key.move(y,-1))[x] +
  fac2*(g_vort.template get<vorticity>(key.move(z,1))[x] +
  g_vort.template get<vorticity>(key.move(z,-1))[x] -
  2.0f*(fac1+fac2+fac3) +
  g_vort.template get<vorticity>(key)[x] +
  fac4*g_vort.template get<vorticity>(key)[x] +
  (g_vel.template get<velocity>(key.move(x,1))[x]-
  g_vel.template get<velocity>(key.move(x,-1))[x]) +
  fac5*g_vel.template get<vorticity>(key)[y] +
  (g_vel.template get<velocity>(key.move(y,1))[x] -
  g_vel.template get<velocity>(key.move(y,-1))[x]) +
  fac6*g_vort.template get<vorticity>(key)[z] +
  (g_vel.template get<velocity>(key.move(z,1))[x] -
  g_vel.template get<velocity>(key.move(z,-1))[x]) +
  g_vel.template get<vorticity>(key.move(z,1))[x] -
  g_vel.template get<vorticity>(key.move(z,-1))[x]) ;
  g_dwp.template get<rhs>(key)[y] =
  g_dwp.template get<rhs>(key)[z] =
}
```

Closing the performance gap

Lennard Jones
(particles, discrete)

Gray-Scott
(mesh, continuous)

Vortex in Cell
(hybrid, continuous)

57 LOC vs 151 LOC

40 LOC vs 100 LOC

73 LOC vs 580 LOC


© Prof. J. Castrillon. ESPM2 @ SC 2023
Higher-level optimizations

- Insertion of ghost-gets, based on high-level dataflow
- Model-based auto-tuning for discretization
- Theoretical convergence to steer search


© Prof. J. Castrillon. ESPM2 @ SC 2023
Higher-level optimizations

- Insertion of ghost-gets, based on high-level dataflow
- Model-based auto-tuning for discretization
- Theoretical convergence to steer search

1x, 8x, 16x more exploration time with various degrees of success

With comparable exploration time, oblivious auto-tuners orders of magnitude worse

Example: Tensor expressions (Physics, ML)

- **CFDlang**

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

```
source = ...
var input A : matrix &
var input u : tensorIN &
var input output v : tensorOUT &
var input alpha : [] &
var input beta : [] &

v = alpha * (A # A # A # u .
    [[5 8] [3 7] [1 6]]) + beta * v
```

```
auto A = Matrix(m, n), B = Matrix(m, n),
    C = Matrix(m, n);
auto u = Tensor<3>(n, n, n);
auto v = (A*B*C)(u);
```
Tensor intermediate language (TeIL) in MLIR

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


- Specialization path to custom hardware

K. F. A. Friebel, J. Bi, J. Castrillon, "BASE2: An IR for Binary Numeral Types" In ACM HEART 2023
Flow from DSL to system-level architecture

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

FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- Transformations for a **17x speedup** (same precision)

FPGA code generation: HBM FPGA

- H2020 EU Project: Convergence HPC, Big Data and ML
- Variants with up to **24x better energy efficiency**


https://everest-h2020.eu
**Base2: Custom precision analysis**

- **Interpolation**
  \[ v_{ijk,e} = \sum_{i'=0}^{P} \sum_{j'=0}^{P} \sum_{k'=0}^{P} A_{kk'} A_{jj'} A_{ii'} v_{i'j'k'e} \]

  - **Significand precision** \( p \)
    \( \text{(exp_bits = 6)} \)
  - **Exponent range** \( \text{ld } E \)
    \( \text{(frac_bits = 32)} \)

K. F. A. Friebe, J. Bi, J. Castrillon, "BASE2: An IR for Binary Numeral Types", In ACM HEART 2023
Towards a system development kit (SDK)

- Kernel integration into legacy (e.g., WRF)
- High-level (implicit) dataflow
- MLIR for heterogeneous integration: System-level design and rich tool interfacing
- Convergence: HPC, Big data and machine learning

Webinar: [https://youtu.be/h7sG9_JFqwk?si=UoM4CRCUvhJgUt3m](https://youtu.be/h7sG9_JFqwk?si=UoM4CRCUvhJgUt3m)

© Prof. J. Castrillon. ESPM2 @ SC 2023
Near and in-memory computing
Rich landscape of designs

- Near-memory: Processors, logic close to memory
- In-memory (aka processing using memory): Leverage device properties

**Samsung, Lee, Sukhan, et al. ISCA 2021**


**CAM accelerators:** Hu, Sharon, et al. 2021 IEDM
CINM: Generalized MLIR infrastructure

- From linear algebra abstractions (common to ML frameworks and beyond)
- Intermediate languages for in and near memory computing
- Pattern recognition, target-specific models and optimizations

A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", arXiv, Aug 2023
CINM: Generalized MLIR infrastructure

- From linear algebra abstractions (common to ML frameworks and beyond)
- Intermediate languages for in and near memory computing
- Pattern recognition, target-specific models and optimizations

A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", arXiv, Aug 2023
def mm(int32(64, 64) A, int32(64, 64) B) -> (int32(64, 64) C) {
    C(i, j) += A(i, k) * B(k, j)
    where i in 0:64, k in 0:64, j in 0:64
}

uint32_t mram_base_addr_A = (uint32_t) (DPU_MRAM_HEAP_POINTER);
uint32_t mram_base_addr_B = (uint32_t) (DPU_MRAM_HEAP_POINTER + ROWS * COLS *
sizeof(T));
uint32_t mram_base_addr_C = (uint32_t) (DPU_MRAM_HEAP_POINTER + 2 * ROWS * COLS
* sizeof(T));
for(int i = (tasklet_id * point_per_tasklet); i < (i++) {
    if( new_row != row ){
        mram_read((__mram_ptr void const*) (mram_base_addr_A + mram_offset_A),
        cache_A, COLS * sizeof(T));
    }
    mram_read((__mram_ptr void const*) (mram_base_addr_B + mram_offset_B),
    cache_B, COLS * sizeof(T));
    dot_product(cache_C, cache_A, cache_B, number_of_dot_products);
    ...
    mram_write( cache_C, (__mram_ptr void *) (mram_base_addr_C + mram_offset_C),
    point_per_tasklet * sizeof(T));
}
UPMEM example: Results

1.4x-1.5x faster than un-optimized versions (geomean)

2x-5x faster that CPU optimized
1.6x-2x faster than PrIM (manually optimized for UPMEM)
Optimization results: Crossbars beyond matmul
Content addressable memories (CAMs)

- NVM-based CAMs: Great for KNNs, One-shot learning, ...
- CINM support for similarity and CAM arch exploration
- Automatic flow from TorchScript matches manual designs

Summary

- Next generation programming for extreme heterogeneity
  - Domain-specific abstractions (implicit), compilation flows, ...
  - Reconfigurable HW, HBM, data placement, near and in-memory computing

- Challenges
  - Understanding and modeling primitives from down below
  - Maintainability and interoperability with low-level programming models
  - Optimization/DSE

\[
\begin{align*}
t &= \left( S \otimes (S \otimes (S \otimes u)_{xyz})_{byz} \right)_{ax} \\
\text{time loop} & \text{ start: 0 stop: 1000} \\
\text{temporal method: explicit euler} \\
\text{spatial method: DC-PSE} \\
\frac{\partial u}{\partial t} &= D_u \star \nabla^2 u - u \star v^2 + F \star (1 - u) \\
\frac{\partial v}{\partial t} &= D_v \star \nabla^2 v + u \star v^2 - v \star (F + k)
\end{align*}
\]
Thanks! & Acknowledgements

..., and previous members of the group (Norman Rink, Sven Karol, Sebastian Ertel, Andres Goens), and collaborators (J. Fröhlich, I. Sbalzarini, A. Cohen, T. Grosser, T. Hoefler, H. Härtig, H. Corporaal, C. Pilato, S. Parkin, P. Jääskeläinen, J-J. Chen, A. Jones, X.S. Hu)
References


[GINM’23] A. Khan et al, "CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms", arXiv, Aug 2023

