Practical Solutions that Are Performant, Portable AND Productive for Programming Extreme Scale Systems

Tim Mattson
timothy.g.mattson@intel.com
Parallel Computing Lab
Intel Corp
Disclaimer

• The views expressed in this talk are those of the speaker and not his employer.

• If I say something “smart” or worthwhile:
  – Credit goes to the many smart people I work with.

• If I say something stupid...
  – It’s my own fault

I work in Intel’s research labs. I don’t build products. Instead, I get to poke into dark corners and think silly thoughts… just to make sure we don’t miss any great ideas.

Hence, my views are by design far “off the roadmap”.

The three talks I’d give if there was time

• The challenge of Hierarchical, Heterogeneous systems:
  • Hardware complexity is out of control (for good reason ... hierarchical heterogeneous systems to optimize Perf/Watt). This is great if you are a hardware architect. This is a nightmare if you develop software.

• “Short Term” Solution: oneAPI with Distributed Computing
  • The oneAPI industry initiative defines an open stack of APIs for programming heterogenous nodes ... so one API maps onto multiple hardware targets. We just need to extend it to cover distributed computing.

• Long term Solution: Automate Software Development
  • Machines are now really good at learning. Maybe they can handle programming too? They better because the programmers we are generating these days choose languages such as Python that abstract away ALL the hardware details.
The three talks I’d give if there was time

• The challenge of Hierarchical, Heterogeneous systems:
  • Hardware complexity is out of control (for good reason ... hierarchical heterogeneous systems to optimize Perf/Watt). This is great if you are a hardware architect. This is a nightmare if you develop software.

• “Short Term” Solution: oneAPI with Distributed Computing
  • The oneAPI industry initiative defines an open stack of APIs for programming heterogenous nodes ... so one API maps onto multiple hardware targets. We just need to extend it to cover distributed computing.

• Long term Solution: Automate Software Development
  • Machines are now really good at learning. Maybe they can handle programming too? They better because the programmers we are generating these days choose languages such as Python that abstract away ALL the hardware details.

I think I’ll give this talk today
Programming Language trends 2005 to 2020

Professional programmers use Java, C, and C++. Professionals who develop software use Python.

http://pypl.github.io/PYPL.html
Programmers and their needs

The Triangle of programmer needs

Portability

Python
Matlab

OpenMP
MPI

CUDA

Productivity

Performance

Delivering 2/3 is reasonable,
Getting 3/3 is Prohibitively difficult

3rd party names are the property of their owners
Programmers and their needs

The Triangle of programmer needs

Delivering 2/3 is reasonable,

Getting 3/3 is Prohibitively difficult

We really need 3/3 … it’s challenging, but not impossible.

It’s been done before

3rd party names are the property of their owners
The Relational Model of Databases

• In 1970 Edgar Codd (IBM) published one of the most important papers in the history of computer science.

• It defined a formal algebra* for building databases ... the relational model.
  • Object: A relation.
    • A set of tuples that share a set of attributes.
    • The set of attributes is defined by a schema
    • A relation is typically represented as a table.
  • A set of operators that act on relations. This set includes:
    • Select \( \sigma \)
    • Rename \( \rho \)
    • Join \( \bowtie \)
    • Project \( \pi \)

* Note: An “algebra” is a set of objects, operators that act on those objects, and rules for how those operators interact with each other.
Database Queries and the Relational Model

• Users interact with the relational database by issuing queries.
• Codd proposed elegant and mathematically rich procedural queries:

\[ \pi_{\text{name}} \left( \sigma_{\text{salary} > \text{salary}} \left( \rho_{\text{employee}} \bowtie_{\text{manager} = \text{name}} \rho_{\text{employee}} \right) \right) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>45,000</td>
<td>Harker</td>
</tr>
<tr>
<td>Jones</td>
<td>40,000</td>
<td>Smith</td>
</tr>
<tr>
<td>Baker</td>
<td>50,000</td>
<td>Smith</td>
</tr>
<tr>
<td>Nelson</td>
<td>55,000</td>
<td>Baker</td>
</tr>
</tbody>
</table>

• When applied to this relation:

• The output is **Baker** and **Nelson** …. employees who earn more than their managers

*This example comes from “Early History of SQL” by D. Chamberlin, IEEE Annals of the history of computing, 2012*
Queries and the Structured Query Language

- Codd’s notation was too obtuse for the general user

\[ \pi_{\text{name}} \left( \sigma_{\text{e.salary} > \text{m.salary}} \left( \rho_{\text{e}(\text{employee})} \bowtie_{\text{e.manager} = \text{m.name}} \rho_{\text{m}(\text{employee})} \right) \right) \]

- In 1974, Codd’s colleagues at IBM (Ray Boyce and Don Chamberlin) created a Query Language for Codd’s relational Algebra called the Structured Query Language (SQL … Pronounced as Sequel)

- The SQL Query equivalent to Codd’s notation above reads:

```sql
select e.name
from employee e, employee m
where e.manager = m.name and e.salary > m.salary
```

- Codd’s notation is procedural. SQL is Declarative … That difference is an important reason for why SQL is the most successful DSL of all time.
Productivity, Performance and Portability in one programming framework

• By the 1980s, database researchers at IBM and UC Berkeley exploited the declarative nature of SQL to build systems that delivered on the “3 Ps” …

- Query written in SQL
- Query compiled into a “logical query plan”
- Optimizer generates optimized “physical query plan”

Enabled by a formal, Relational Algebra

• Can we replicate this strategy for programming heterogeneous systems?

- Program with DSL
- Program compiled into a “logical exec plan”
- Optimizer generates optimized “physical exec plan”

Enabled by a formal Algebra
These two diagrams are equivalent representations of a graph.

\[ A^T \]

A = the adjacency matrix
Elements nonzero edges connect vertices

Source: Pictures from Aydin Buluc and my colleagues in the GraphBLAS Forum
Multiple-source breadth-first search

Multiplication of sparse matrices captures Breadth first search and serves as the foundation of all algorithms based on BFS

- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
Halide: Focusing on programmer intent

Halide

separates the

Algorithm:

• What the program does,
• Written by a domain specialist

from the

Schedule:

• How the program runs
• Written by SW/HW expert

### Algorithm

```cpp
Func blur_3x3(Func input) {
    Func blur_x, blur_y;
    Var x, y, xi, yi;

    // The algorithm - no storage or order
    blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
    blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

    return blur_y;
}
```

### Schedule

```cpp
// The schedule - defines order, locality; implies storage
blur_y.tile(x, y, xi, yi, 256, 32).vectorize(xi, 8).parallel(y);
blur_x.compute_at(blur_y, x).vectorize(x, 8);
```

The common theme between SQL and Halide?
Separation of concerns

• SQL separates logical plan from the physical plan
  – The query optimizer invents new algorithms using the relational algebra to drive transformations into the emerging plan.
  – The query optimizer then adapts the emerging plan into the physical plan to match the hardware.

• Halide separates programmer intention (the DSL) from how it adapts onto the final hardware

• Isolates the three fundamental concerns of programming
  – Intension
  – Invention
  – adaptation
Separation of concerns

- Let's break up the software development process and consider each aspect separately.
Separation of concerns

• Let’s break up the software development process and consider each aspect Separately

Programmers should just worry about expressing their intent. We will automate the Invention and Adaptation work.
A position paper laying out our vision for how to solve the machine programming problem. The three Pillars:

- **Intention**: Discover the intent of a programmer
- **Invention**: Create new algorithms and data structures
- **Adaption**: Evolve in a changing hardware/software world
Three Pillar Examples*

*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI’18, arxiv.org/pdf/1803.07244.pdf

- **Intention**

- **Invention**
  - “Speedy Point: Optimized Cpu code from Cuda”, Adam Herr and the Intel Inteon group

- **Adaptation**
  - “Learning to Optimize Halide with Tree Search and Random Programs” (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019

---

- **Put all three together ... and something awesome happens**
  - “ScaMP: Scalable Machine Programming” a research program at MIT funded by Intel and MIT … launched fall'22
Three Pillar Examples*  

*2nd ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL), PLDI’18, arxiv.org/pdf/1803.07244.pdf

- **Intention**
  - “Verified Lifting of Stencil Computations”, S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI’16

- **Invention**
  - “Speedy Point: Optimized Cpu code from Cuda”, Adam Herr and the Intel Inteon group

- **Adaptation**
  - “Learning to Optimize Halide with Tree Search and Random Programs” (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019

- **Put all three together … and something awesome happens**
  - “ScaMP: Scalable Machine Programming” a research program at MIT funded by Intel and MIT … launched fall’22
MetaLift and Verified Lifting

Syntax guided generator to transform input code into a high-level DSL for automated analysis.
(STNG ... Fortran → halide)

Figure 3. The STNG compiler toolchain
VC Computation: Verified condition generator

http://metalift.uwplse.org
MetaLift and Verified Lifting

Syntax guided generator to transform input code into a high-level DSL for automated analysis.
(STNG ... Fortran → halide)

• Input: serial Fortran (reference for speedup), a stencil code
• Output: Halide

• Results: NAS Parallel Benchmarks MG15, speedups
  – 17.51 CPU,
  – 4.3 GPU

• CPU: 24-core dual-socket Intel Xeon E5-2695v2 CPUs at 2.4 Ghz, 128 GB mem.
• GPU: NVIDIA K80 ... NOTE: their autotuning framework did not optimize the GPU so don’t read too much into the GPU numbers

3rd party names are the property of their owners

http://metalift.uwplse.org
Three Pillar Examples

- **Intention**
  - “Verified Lifting of Stencil Computations”, S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI’16

- **Invention**
  - “Speedy Point: Optimized Cpu code from Cuda”, Adam Herr and the Intel Inteon group

- **Adaptation**
  - “Learning to Optimize Halide with Tree Search and Random Programs” (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019

- Put all three together … and something awesome happens
  - “ScaMP: Scalable Machine Programming” a research program at MIT funded by Intel and MIT … launched fall’22
Speedy Point#: Automatically optimizing Sycl code

- Extend Sycl Compiler with Machine Programming (MP) flags
- Use Polygeist* to lower device code to MLIR
- Translate MLIR-SYCL to Halide
- MP Codegen to Halogen
- Link result with SYCL runtime and auto-scheduling
- Produce SYCL complete binary

Performance Optimization as a search problem:
- AI-based cost model to guide heuristic search (alpha-go), trained with millions of wall-time measurements of real workloads

#From Intel's Inteon group.  *https://polygeist.mit.edu/  Third Party Names are the Property of their owners.
From CUDA to optimized CPU code

- Syclomatic: A oneAPI tool from Intel to convert CUDA into Sycl
- SPEEDY POINT: Convert code to Halide and then auto-schedule to optimize performance

Preliminary results show that gains of 40x or more are possible vs. the translated but unoptimized SYCL code.
Three Pillar Examples

- **Intention**
  - "Verified Lifting of Stencil Computations", S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI’16

- **Invention**
  - "Speedy Point: Optimized Cpu code from Cuda", Adam Herr and the Intel Inteon group

- **Adaptation**
  - "Learning to Optimize Halide with Tree Search and Random Programs" (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019

- **Put all three together ... and something awesome happens**
  - "ScaMP: Scalable Machine Programming" a research program at MIT funded by Intel and MIT … launched fall'22
Halide Learned Schedules

Superhuman Performance

A new automatic scheduling algorithm for Halide

<table>
<thead>
<tr>
<th>Speed-up (higher is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5 x</td>
</tr>
<tr>
<td>2 x</td>
</tr>
<tr>
<td>1.5 x</td>
</tr>
<tr>
<td>1 x</td>
</tr>
<tr>
<td>0.5 x</td>
</tr>
<tr>
<td>0 x</td>
</tr>
</tbody>
</table>

Prior work (Mullapudi 2016)  Expert Humans  This paper

Larger search space
- includes more Halide scheduling features
- extensible

Hybrid cost model
- Mix of machine learning and hand-designed terms
- Can model complex architectures

Three Pillar Examples*

- **Intention**
  - “**Verified Lifting of Stencil Computations**”, S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama, PLDI’16

- **Invention**
  - “**Speedy Point: Optimized Cpu code from Cuda**”, Adam Herr and the Intel Inteon group

- **Adaptation**
  - “**Learning to Optimize Halide with Tree Search and Random Programs**” (Adams, Ma, Anderson, Baghdadi, Li, Gharbi, Steiner, Johnson, Fatahalian, Durand, Ragan-Kelley) SIGGRAPH 2019

- **Put all three together ... and something awesome happens**
  - “**ScaMP: Scalable Machine Programming**” a research program at MIT funded by Intel and MIT ... launched fall'22
ScaMP: Scalable Machine Programming
A five-year research program at MIT funded by Intel and NSF (Launched Oct 2022)

I want to build a web based workflow management system for fluid dynamics simulations

Source code expressed in terms of composable DSLs

Correct by construction code generators

Verified DSL compilers

Learned autoscanners

Learned cost models

Clusters of heterogeneous nodes

Performance History database

Initial “spec”

Incremental Multimodal Specification (IMS)

Progressive Reification

DSL Synthesis

Coordination Synthesis

Conversation (iterative refinement)

Source:

MIT PIs
- Saman Amarasinghe
- Michael Carbin
- Adam Chlipala
- Jonathan Ragan-Kelley
- Armondo Solar-Lezama
Programming in the 23rd Century: Conversational Computing

• With the three pillars we are on our way to 23rd century programming
  – **Intention**: Natural language processing plus visual information
  – **Invention**: Lifting into a DSL, ML to invent algorithms, Theorem prover to verify.
  – **Automation**: Autotuning + ML to optimize for “any” HW

source: Star Trek IV: the journey home.

But there is much more we need to do on each of the pillars (especially the intention pillar)

… and we have partnered with NSF and MIT to make it happen!
Conclusion

• Hardware is heterogeneous … we need to support heterogeneous programming … i.e. a single programs that runs on all processors (GPUs, CPUs, and more)

• We need Productivity, portability and Performance (the 3 Ps).

• The Database community pointed the way back in the 1970s … a declarative programming interface build on a formal algebra

• Our research … Machine Programming to map intention onto HPC hardware