Should I port my code to a DSL?

Bahareh Davani · Ferran Marti · Laleh Beni · Saikiran Ramanan · Feng Liu
Aparna Chandramowlishwaran

October 27, 2017 — Scholas Dagstuhl
CONTEXT: HiPer
("HIGH PERFORMANCE TURBULENT FLOW SIMULATIONS")
CONTEXT: MoBo
(“MOVING BOUNDARIES”)

Citation: “Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures.” In SC’10.
Winner, Gordon Bell Prize. http://dx.doi.org/10.1109/SC.2010.42
Prior work with same physical fidelity

- **1,200 cells**: Sequential + integral equations
  Zinchenko et al. (2003)

- **14,000 cells**: IBM BG/P + Lattice Boltzmann
  O(10k) unknowns/cell
  Clausen et al. (2010)

MoBo: **260 million cells** (90 billion unknowns) on **200k cores** (Jaguar @ ORNL)

- CPU, GPU + integral equations + implicit AMR
  O(100) unknowns / cell

Key to scaling: Optimal n-body methods based on the fast multipole method (FMM) on highly non-uniform domains
Why N-body methods?

- One of the original seven dwarfs or motifs
- FMM listed among the top 10 algorithms having the greatest influence in 20th century
- EM is one of the top 10 algorithms having the highest impact in data mining

- Applications
  - Machine learning
  - Computer vision
  - Computational geometry
  - Scientific computing …
“Everyone is doing stencils.”
*Anonymous Wolverine.*

“Stencils are easy, they are structured”
*Anonymous Chipmunk.*

“We need separation of concerns” (drink!)
*Anonymous Chupacabras.*

“We need better performance models”
*Anonymous Axolotl.*

Do current frameworks capture stencil patterns in “real applications”?

What is the gap between stencil DSLs and hand-optimized code for “real applications”?

What is the right separation of concerns?

Story time!
“Everyone is doing stencils.”
Anonymous Wolverine.

“Stencils are easy, they are structured.”
Anonymous Chipmunk.

“We need separation of concerns”
Anonymous Chupacabra.

“We need better performance models”
Anonymous Axolotl.

Do current frameworks capture stencil patterns in "real applications"?
What is the gap between stencil DSLs and hand-optimized code for "real applications"?

What is the right separation of concerns?

Story time!
Computational fluid dynamics simulations
GOVERNING EQUATIONS

- 3D Unsteady Reynolds Averaged Navier-Stokes (URANS) equations
- Dual time-stepping scheme
  - Pseudo-time marching — multi-stage Runge-Kutta scheme
  - Marched to a steady state in pseudo time
- Spatial discretization of the residual
  - 2nd order accurate
STENCIL PATTERNS

- Cell-centered stencils
  - Most well-studied in literature

- Vertex-centered stencils
  - More complex memory access pattern
  - More memory-bound than cell-centered stencils
Stencil Patterns

- Cell-centered stencils
  - Most well-studied in literature

- Vertex-centered stencils
  - More complex memory access pattern
  - More memory-bound than cell-centered stencils
**Single- and Multi-core Optimizations**

(Cylinder flow with 2 million cells)

- **Haswell**
  - Speedup: ~105x
  - Number of threads: 1, 2, 4, 8, 16, 32
  - Regions: NUMA, SMT

- **Abu Dhabi**
  - Speedup: ~159x
  - Number of threads: 1, 2, 4, 8, 16, 32, 64
  - Regions: NUMA, SMT

- **Broadwell**
  - Speedup: ~160x
  - Number of threads: 1, 2, 4, 8, 16, 22, 44, 88
  - Regions: NUMA, SMT

Color Legend:
- +Strength Reduction
- +Fusion
- +Parallelism
- +NUMA
- +Blocking
- +SIMD Transformations
- +SIMD
The preceding optimizations were **manually coded**. Can such CFD solvers can be expressed in stencil DSL’s?
The preceding optimizations were manually coded. Can such CFD solvers can be expressed in stencil DSL’s?

Yes! 1 month effort in Halide.

Can stencil DSL’s deliver a sufficient combination of optimizations to compete with a hand-tuned code?
<table>
<thead>
<tr>
<th></th>
<th>Haswell</th>
<th>Abu Dhabi</th>
<th>Broadwell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand-tuned</td>
<td>Halide</td>
<td>Hand-tuned</td>
<td>Halide</td>
</tr>
<tr>
<td>Optimization</td>
<td>3.5x</td>
<td>1.5x</td>
<td>3x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.3x</td>
<td>3.1x</td>
</tr>
</tbody>
</table>

This gap is due to **strength reduction** and **inter-stencil fusion** in the hand-tuned code.
<table>
<thead>
<tr>
<th></th>
<th>Haswell</th>
<th>Abu Dhabi</th>
<th>Broadwell</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hand-tuned</td>
<td>Halide</td>
<td>Hand-tuned</td>
</tr>
<tr>
<td>Optimization</td>
<td>3.5x</td>
<td>1.5x</td>
<td>3x</td>
</tr>
<tr>
<td>+Vectorize</td>
<td>3.6x</td>
<td>1.1x</td>
<td>2.3x</td>
</tr>
<tr>
<td></td>
<td>Haswell</td>
<td>Abu Dhabi</td>
<td>Broadwell</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>Hand-tuned</td>
<td>Halide</td>
<td>Hand-tuned</td>
</tr>
<tr>
<td>Optimization</td>
<td>3.5x</td>
<td>1.5x</td>
<td>3x</td>
</tr>
<tr>
<td>+Vectorize</td>
<td>3.6x</td>
<td>1.1x</td>
<td>2.3x</td>
</tr>
<tr>
<td>+Parallelize</td>
<td>7.9x</td>
<td>5.8x</td>
<td>23.3x</td>
</tr>
</tbody>
</table>

This gap is partly due to **NUMA-aware parallelization** in the hand-tuned code. (Halide is currently not NUMA-aware)
Can stencil DSL’s deliver a sufficient combination of optimizations to compete with a hand-tuned code?

Not yet! But, there is hope.
N-body problems
Naive inefficient kernel code

```c
int
    kernel_laplace(const DblNumMat& srcPos, const DblNumMat& trgPos,
                    const DblNumVec& srcDen, DblNumVec& trgVal)
{
    int M = trgPos.n();
    int N = srcPos.n();
    DblNumMat mat(M,N);

    double OOFP = 1.0/(4.0*M_PI);
    for(int i=0; i<trgPos.n(); i++) {
        for(int j=0; j<srcPos.n(); j++) {
            double x = trgPos(0,i) - srcPos(0,j);
            double y = trgPos(1,i) - srcPos(1,j);
            double z = trgPos(2,i) - srcPos(2,j);
            double r2 = x*x + y*y + z*z;
            double r = sqrt(r2);
            if (r != 0.0)
                mat(i,j) = OOFP / r;
        }
    }
    dgemv(1.0, mat, srcDen, 1.0, trgVal);
}
```

\[ q_i = \sum_{j} K(r_{ij}) \phi_j \]

\[ r_{ij} = |x_i - y_j| \]

\[ K(r) = \frac{C}{r} \]
**Hand-optimized kernel code**

36x faster (dual-socket quad-core x86)

- Single-core, manually coded & tuned
  - *Data*: Structure reorg. (transpose or “SOA”), NUMA-aware memory allocation
  - *Traffic*: Matrix-free via interprocedural loop fusion, blocking/tiling
  - *Numerical*: rsqrtps + Newton-Raphson (x86)
  - *Low-level*: SIMD vectorization (x86)
  - OpenMP parallelization/scheduling
  - Algorithmic tuning

*Large, complex tuning spaces*
N-BODY CALCULATIONS

What do these have in common?

\[ \forall q \in Q : \quad F(q) = \sum_{r \in (Q \setminus \{q\})} \frac{C}{\|r - q\|^3} \]  

Force computation

\[ \forall q \in Q : \quad \text{AllNN}(q) = \arg\min_{r \in R} d(q, r) \]  

Nearest neighbors

\[ \forall q \in Q : \quad \text{KDE}(q) = \frac{1}{|R|} \sum_{r \in R} K(q, r) \]  

Kernel density estimation

\[ \forall q \in Q : \quad \text{Range}(q) = \sum_{r \in R} I(\text{dist}(q, r)) \le h \]  

Range count

Consider pairs of points – naïvely \( O(N^2) \)
COMMONALITY: OPTIMAL APPROXIMATION ALGORITHMS

\[ \forall q \in Q : \quad F(q) = \sum_{r \in (Q-\{q\})} C \frac{r - q}{||r - q||^3} \]

- Hierarchical tree-based approximation algorithms for force computations, e.g., Barnes-Hut or FMM

Evaluate interactions → Tree traversals

Store aggregate data at nodes, e.g., bounding box, mass
# N-BODY PROBLEMS IN OTHER DOMAINS

<table>
<thead>
<tr>
<th>Problem</th>
<th>Operators</th>
<th>Kernel Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Nearest Neighbors</td>
<td>$\forall, \arg\min$</td>
<td>$</td>
</tr>
<tr>
<td>All Range Search</td>
<td>$\forall, \cup\arg$</td>
<td>$I(h_{\min} &lt;</td>
</tr>
<tr>
<td>All Range Count</td>
<td>$\forall, \Sigma$</td>
<td>$I(h_{\min} &lt;</td>
</tr>
<tr>
<td>Naive Bayes Classifier</td>
<td>$\forall, \arg\max$</td>
<td>$\frac{1}{\sqrt{2\pi</td>
</tr>
<tr>
<td>Mixture Model E-step</td>
<td>$\forall, \forall, \forall,$</td>
<td>$\frac{1}{\sqrt{2\pi</td>
</tr>
<tr>
<td>K-means E-step</td>
<td>$\forall, \arg\min$</td>
<td>$</td>
</tr>
<tr>
<td>Mixture Model Log-likelihood</td>
<td>$\sum \log\sum$</td>
<td>$\frac{1}{\sqrt{2\pi</td>
</tr>
<tr>
<td>Kernel Density Estimation</td>
<td>$\forall, \Sigma$</td>
<td>$\frac{1}{\sqrt{2\pi</td>
</tr>
<tr>
<td>Kernel Density Bayes Classifier</td>
<td>$\forall, \arg\max\Sigma$</td>
<td>$I(\forall</td>
</tr>
<tr>
<td>2-point (cross-)correlation</td>
<td>$\Sigma, \Sigma$</td>
<td>$\phi(</td>
</tr>
<tr>
<td>Nadaraya-Watson Regression</td>
<td>$\forall, \Sigma$</td>
<td>$\phi(</td>
</tr>
<tr>
<td>Thermodynamic Average</td>
<td>$\Sigma, \Sigma$</td>
<td>$\phi(</td>
</tr>
<tr>
<td>Largest-span set</td>
<td>$\max, \ldots, \max$</td>
<td>$\Sigma(</td>
</tr>
<tr>
<td>Closest Pair</td>
<td>$\arg\min, \arg\min$</td>
<td>$</td>
</tr>
<tr>
<td>Minimum Spanning Tree</td>
<td>$\forall, \arg\min$</td>
<td>$</td>
</tr>
<tr>
<td>Coulombic Interaction</td>
<td>$\forall, \Sigma$</td>
<td>$\frac{1}{</td>
</tr>
<tr>
<td>Average Density</td>
<td>$\Sigma, \Sigma$</td>
<td>$I(</td>
</tr>
<tr>
<td>Wave function</td>
<td>$\forall, \Pi$</td>
<td>$\phi(</td>
</tr>
<tr>
<td>Hausdorff Distance</td>
<td>$\max, \min$</td>
<td>$</td>
</tr>
<tr>
<td>Intrinsic (fractal) Dimension</td>
<td>$\Sigma, \Sigma$</td>
<td>$I(</td>
</tr>
</tbody>
</table>

Each problem has a set of operators and a kernel function.
k-nearest neighbors \[ \forall q, \ \arg \min_r^k ||x_q - x_r|| \]

```cpp
Storage query(filePathString1);
Storage reference(filePathString2);
PortalExpr expr;
expr.addLayer(PortalOp(PortalOp::OP::FORALL), query);
expr.addLayer(PortalOp(PortalOp::OP::KARGMIN, k), reference,
              PortalFunc(PortalFunc::TYPE::EUCLIDEAN));
Storage knnoutput = expr.execute();
```
EXPERIMENTAL SETUP

• Architecture
  • Dual-socket Intel Xeon E5-2630 v3 processor (Haswell-EP)
  • Each socket has 8 cores
  • Theoretical peak performance of 614.4 GFlops

• Compiler
  • Intel C++ compiler (icpc v15.0.2)
  • Python v2.7.6 (Scikit-learn)
  • Java v1.8.0 (Weka)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$N$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo!</td>
<td>41904293</td>
<td>11</td>
</tr>
<tr>
<td>IHEPC</td>
<td>2075259</td>
<td>9</td>
</tr>
<tr>
<td>HIGGS</td>
<td>11000000</td>
<td>28</td>
</tr>
<tr>
<td>Census</td>
<td>2458285</td>
<td>68</td>
</tr>
<tr>
<td>KDD</td>
<td>4898431</td>
<td>42</td>
</tr>
</tbody>
</table>
CASE STUDIES (DIRECT)

- Nearest Neighbors
  \[ \forall q, \ \arg \min_r \| x_q - x_r \| \]

- Range-Search
  \[ \forall q, \bigcup \arg_r I (\| x_q - x_r \| \leq h) \]

- Kernel Density Estimation
  \[ \forall q, \frac{1}{N_r} \sum_r K \left( \frac{\| x_q - x_r \|}{\sigma} \right) \]

- Hausdorff Distance
  \[ \max_q, \min_r \| x_q - x_r \| \]
CASE STUDIES (ITERATIVE)

- Expectation Maximization (EM)

  E-step \[ \forall q, \forall r, \quad \frac{\pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n | \mu_j, \Sigma_j)} \]

  M-step

  Log-likelihood \[ \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k) \]

- Euclidean Minimum Spanning Tree \[ \forall q, \arg \min_r ||x_q - x_r|| \]
**Library Comparison**

- **MATLAB**: over 1,000,000 licensed users, uses C in backend
- **Weka**: 6,677,053 downloads, written in Java
- **Scikit-learn**: 121,841 downloads, written in Python
- **MLPACK**: exploits C++ language features to provide maximum performance

---

**EM**

- Yahoo!: 6.2 Base 6.3 5.3
- HIGGS: 7.5 Base 8.9 3.5
- Census: 14.5 Base 23.1 2.1
- KDD: 4.7 Base 12.3 2
- IHEPC: 4.5 2 13.3 Base

**kNN**

- Yahoo!: 18.4 Base 22.3 5.2
- HIGGS: 3.9 1.6 7.9 Base
- Census: 3.4 Base 6.1 1.4
- KDD: 7.7 Base 15.4 1.3
- IHEPC: 4.1 Base 6.1 1.5
### Speedup Breakdown

<table>
<thead>
<tr>
<th></th>
<th>KNN</th>
<th>EM</th>
<th>KDE</th>
<th>HD</th>
<th>RS</th>
<th>EMST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alg +Opt +Par</td>
<td>Alg +Opt +Par</td>
<td>Alg +Opt +Par</td>
<td>Alg +Opt +Par</td>
<td>Alg +Opt +Par</td>
<td>Alg +Opt +Par</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>3.1 12.1 173.1</td>
<td>1.6 3.2 53.7</td>
<td>2.1 9.1 92.1</td>
<td>2.5 11.5 161.1</td>
<td>2.2 9.1 126.8</td>
<td>2.9 11.9 166.7</td>
</tr>
<tr>
<td>HIGGS</td>
<td>2.1 7.3 108.1</td>
<td>1.5 6.8 117.6</td>
<td>1.7 4.7 50.1</td>
<td>1.9 6.1 89.6</td>
<td>1.9 6.3 86.5</td>
<td>2.0 6.9 102.8</td>
</tr>
<tr>
<td>Census</td>
<td>1.4 6.5 90.8</td>
<td>1.3 11.2 190.0</td>
<td>1.4 8.1 75.6</td>
<td>1.3 10.2 141.8</td>
<td>1.3 10.4 144.9</td>
<td>1.4 10.9 151.6</td>
</tr>
<tr>
<td>KDD</td>
<td>1.6 6.8 100.7</td>
<td>1.4 4.1 70.9</td>
<td>1.5 3.1 33.5</td>
<td>1.4 3.8 54.4</td>
<td>1.4 5.1 70.5</td>
<td>1.5 3.8 55.5</td>
</tr>
<tr>
<td>IHEPC</td>
<td>3.0 4.3 61.5</td>
<td>1.5 7.6 127.6</td>
<td>2.0 5.4 53.6</td>
<td>2.5 6.8 101.3</td>
<td>2.1 6.3 94.1</td>
<td>2.9 7.1 107.1</td>
</tr>
</tbody>
</table>
CONCLUSIONS

CFD solvers can be expressed in stencil DSL’s with minimal effort.

Portal can generate out-of-the-box new optimal N-body algorithms — $O(N \log N)$ EM and $O(N)$ Hausdorff distance.

Limitations

- Finding the optimal schedule for performance is non-trivial.
- Most stencil DSL’s are only optimized for cell-centered stencils.
- Does not support sufficient combination of optimizations to compete with hand-tuned code yet.