Empowering Scientists with Domain Specific Languages

Julian Kunkel, Nabeeh Jum’ah

Scientific Computing
Department of Informatics
University of Hamburg

SciCADE2017
2017-09-13
Outline

1. Developing Scientific Applications
2. Domain-specific Languages
3. AIMES Project
4. Summary
Developing Scientific Applications

Runtime perspective

- Performance demanding
- Earth system modelling is an example
  - More precise forecasts \( \Rightarrow \) higher resolution grids
  - Ensemble computation
- Should exploit available compute resources
- HPC landscape increasingly inhomogeneous

Development view

- Productivity should be the goal
- Software readability/maintainability is a challenge
  - Continuous code changes due to experimental character
  - Branches to optimize code for different systems
- Software engineering concepts rarely used (agile development)
Example

- Goal: multiplication of two matrices
- Scientists perspective: \( C = A \cdot B \)

Programming

- In Matlab: \( C = A \ast B \) alternatively \( C = \text{mtimes}(A, B) \)
- In Mathematica: \( C = A.B \)
- In R: \( C = A \%*\% B \)
- In Fortran: \( C = \text{matmul}(A, B) \)
- In NumPy: \( C = \text{np.matmul}(A, B) \)
- Optimized math library – BLAS for C/Fortran:
  
  \[ \text{DGEMM}(\text{TransA}, \text{TransB}, M, N, K, \text{ALPHA}, A, \text{LDA}, B, \text{LDB}, \text{BETA}, C, \text{LDC}) \]
Code Optimizations Lead to Diversification

BLAS levels

- BLAS1 Vector operations
- BLAS2 Matrix-Vector operations
- BLAS3 Matrix-Matrix operations

Reason for additional levels

- Reduce coding effort
- Efficient reuse of cache \( \Rightarrow \) minimize memory transfers

Outlook

- Optimize calling multiple BLAS3 routines? BLAS4+?
- Compile-time or runtime system needed!
Stencil Computation

Usage

- Finite difference methods in climate/weather
- Numerical methods (explicit or implicit)
- Potentially low arithmetic density, needs cache reuse!

Cache Reuse with Stencils

- Example with three stencils:
  
  for each timestep:
  
  applyStencil(S1, in:{varA, varB}, out:varC)
  applyStencil(S2, in:{varA, varC}, out:varD)
  applyStencil(S3, in:{varB, varD}, out:varA)

- Mandatory to optimize across stencils
- Machine dependent optimizations, autotuning necessary
Capabilities of Compilers

Limitations of optimization strategies

- E.g., Vectorization, loop unrolling, interprocedural analysis
- Needs information about execution to perform optimization
- Must follow the semantics of the (general purpose) language
- Based on pattern matching, often full potential is not used
- **Not available:** memory layout adaption, cache management

Optimization time

- Traditionally: at compile time (also true for C++ templates)
- Profile guided optimization provides some runtime information
- Just-in-time compilers (runtime, may create special versions)
- Runtime: Lazy execution by library compilers (Big Data tools)
Runtime: Lazy execution by Library Compilers

Concept

- GPL is used to setup control flow
  - GPL compiler won’t optimize performance critical code-regions
- Library provides functions to register and start computation
- Library generates (optimal) architecture-specific code
  - Exploiting semantics of the library
  - All information needed to create code is available in memory

Example

```
registerStencil(S1, in:{varA, varB}, out:varC)
registerStencil(S2, in:{varA, varC}, out:varD)
registerStencil(S3, in:{varB, varD}, out:varA)
executeStencils(timesteps)
```
Development Approaches

■ Manual optimization of source code:
  ■ Adjust code to be easily consumable/optimizable by compilers
  ■ Reduces code readability, many branches
  ■ Complicates maintainability

■ Libraries:
  ■ Provide optimized codes usable across applications
  ■ Address multiple target architectures
  ■ Machine-dependent solutions
  ■ Optimization across library calls often not possible

■ Just-in-time and runtime compilers:
  ■ Complex to develop and understand
  ■ Compile overhead (to machine representation) at runtime

■ Domain-specific languages:
  ■ High-level semantics of application users possible
  ■ Potentially code-preparation at compile time or runtime
### Domain-Specific Languages

**Domain-specific language (DSL)**

Language assisting to describe (solutions for) problems within a certain domain

---

**Technical vs. domain-oriented DSLs**

- **Technical DSL helps to formulate technical requirements**
  - Instructions for the “compiler” to perform certain optimizations
  - Need further effort and technical knowledge from scientists.
  - Example: OpenMP, OpenACC, ...

- **Domain-oriented DSL**
  - Serves the scientists productivity (expressive, ease of use)
  - At best: write code as you describe the problem in the domain
  - System can exploit the semantics to optimize on different levels
  - Generates (optimized) code for a specific architecture
  - Acceptance from scientists is crucial
Domain-Specific Languages: Classification

**Standalone vs. language extensions**

- **Standalone DSLs**
  - Enables paradigm shift to, e.g., declarative programming
  - Complete language, requires rewrite of existing code

- **Language extensions**
  - Built on an existing general-purpose language
  - Introduces constructs not understood by the GPL compiler:
    - Needs an own compiler, preprocessor, or
    - Source-to-source code translation (DSL ⇒ GPL code)
  - May support incremental porting of code
Domain-Specific Languages

**ATMOL**

- A domain-specific language
- Used for atmospheric modeling
- Declarative high-level constructs
  - Declare independent variables
  - Declare dependent variables
    - Data types
    - Lower/Upper bounds
    - Units
    - Including scalars and fields
    - Declare new PDE operators
    - PDEs are defined with arithmetic expressions
    - Boundary conditions support via conditional expressions
- Translated into efficient numerical codes
ATMOL code examples

% Declare spatial and time dimensions:
  space (x(i), y(j), z(k)) time t.

% Declare grid size variables n, m, and l:
  n :: integer (1..infinity); m :: integer (1..infinity); l :: integer (2..infinity).

% For convenience, define macros for two grid domains spanning (i, j, k):
  atmosphere := i=1..n by j=1..m by k=1..l; surface := i=1..n by j=1..m.

% Set coordinate system for symbolic derivation with chain—rule:
  coordinates := [x, y]; coefficients := [h x, h y].

% Declare the model fields:
  u :: float dim "m/s" field (x(half), y(grid), z(grid)) on atmosphere.
  v :: float dim "m/s" field (x(grid), y(half), z(grid)) on atmosphere.
  u_aux :: float dim "Pa m/s" field (x(half), y(grid), z(half)) on atmosphere.
  v_aux :: float dim "Pa m/s" field (x(grid), y(half), z(half)) on atmosphere.
  p :: float (0..107000) dim "Pa" field (x(grid), y(grid), z(grid)) monotonic k(+) on atmosphere.
  p_s_t :: float dim "Pa/s" field (x(grid), y(grid)) on surface.

% Define macro for the horizontal wind velocity vector components:
  V := [u_aux, v_aux].

% Equations:
  p_s_t = −int(nabla .* V, z=1..l).
  V = [u, v] * d p/d z.
PATUS DSL

- A code generation and auto-tuning framework
- Domain: Stencil computations
  - Stencil specifications embedded in a C-like DSL
- Optimization strategy
  - A special DSL is provided to specify a strategy
  - Parametrized for autotuning
- Architecture-specific optimized C code is generated
PATUS Stencil Specification Example

stencil uxx1
{
    domainsize = (nxb .. nxe, nyb .. nye, nzb .. nze);
    t_max = 1;

    operation (  
        const float grid d1(-1..nx+2, -1..ny+2, -1..nz+2),
        float grid u1(-1..nx+2, -1..ny+2, -1..nz+2),
        const float grid xx(-1..nx+2, -1..ny+2, -1..nz+2),
        const float grid xy(-1..nx+2, -1..ny+2, -1..nz+2),
        const float grid xz(-1..nx+2, -1..ny+2, -1..nz+2),
        float param dth)
    {
        float c1 = 9./8.;
        float c2 = -1./24.;

        float d = 0.25 * d1[x,y,z] + d1[x,y-1,z] +
                  d1[x,y,z-1] + d1[x,y-1,z-1]);
        u1[x,y,z; t+1] = u1[x,y,z; t] + (dth / d) * (  
            c1 * (  
                xx[x,y,z] - xx[x-1,y,z] +  
                xy[x,y,z] - xy[x, y-1,z] +  
                xz[x,y,z] - xz[x, y, z-1]) +  
            c2 * (  
                xx[x+1,y,z] - xx[x-2,y,z] +  
                xy[x, y+1,z] - xy[x, y-2,z] +  
                xz[x, y, z+1] - xz[x, y, z-2])  
        );
    }
}
strategy cacheblocking (domain u, auto dim cb,
    auto int chunk)
{
    // iterate over time steps
    for t = 1 .. stencil.t_max
    {
        // iterate over subdomain
        for subdomain v(cb) in u(:, t)
            parallel schedule chunk
            {
                // calculate the stencil for each point
                // in the subdomain
                for point p in v(:, t)
                    v[p; t+1] = stencil (v[p; t]);
            }
    }
}
STELLA DSL

- A domain-specific extended language
  - Uses template metaprogramming within C++
  - A user writes a single code
  - An operator is defined with stages
  - Python support with stencil formulation

- Code is translated at compile time for a specific architecture
  - Loops are generated for the architecture
  - A user-provided functor is used to generate the stencil code

- Multiple backends
  - Multicore CPUs with OpenMP
  - GPUs with CUDA

- A specific memory layout is used for each backend
- Automatically fuses operator stages to enhance locality
The Laplacian operator as a stage for Horizontal Diffusion

```c++
// declarations
IJKRealField data;
Stencil horizontalDiffusion;

// declare stencil stage
template<typename TEnv>
struct Laplace {
    STENCIL STAGE(TEnv)
    STAGE PARAMETER(FullDomain , phi)
    STAGE PARAMETER(FullDomain , lap)

    static void Do(Context ctx , FullDomain) {
        ctx[lap::Center()] = -4.0 * ctx[phi::Center()] +
        ctx[phi::At(iplus1)] + ctx[phi::At(iminus1)] +
        ctx[phi::At(jplus1)] + ctx[phi::At(jminus1)];
    }
};
```
STELLA code example

Two-stage horizontal diffusion, with Laplacian and Divergence

```cpp
// define and initialize the stencil
StencilCompiler::Build(
    horizontalDiffusion,
    // define the input/output parameters,
    pack parameters(
        Param<res, cInOut>(dataOut), Param<phi, cIn>(data)
    ),
    define temporaries(
        StencilBuffer<lap, double, KRange<FullDomain,0,0>>( ),
    ),
    define loops(
        define sweep<cKIncrement>(
            define stages(
                StencilStage<Laplace, IJRange<cIndented,-1,1,-1,1>,
                KRange<FullDomain,0,0>>( ),
                StencilStage<Divergence, IJRange<cIndented,0,0,0,0>,
                KRange<FullDomain,0,0>>( ),
            )
        )
    )
);  // execute the stencil instance
horizontalDiffusion.APply();
```
AIMES Project

Address key issues of icosahedral earth-system models

- Enhance programmability and performance-portability
- Increase storage efficiency
- Provide a common benchmark for ICO models

Covered models

ICON
DYNAMICO
NICAM
AIMES higher level coding approach

- Re-arrange model development workload
  - Domain scientists develop domain logic in source code
  - Scientific programmers write hardware configurations
- Source code written with extended language
  - Closer to domain scientists logic
  - Scientists do not need to learn optimization
  - Write code once, get performance for various configurations
- Hardware configurations define software performance
  - Written by programmers with more experience in platform
  - Comprise information on target run environment
AIMES Approach

**Approach**

- We build a translation tool that is configurable
  - Language can be adjusted for the needs of the scientists
  - Processing engine should reduce repeating patterns (in GPL)
  - GGDML language example discussed with ICO* model teams

- Parses language extension of GPL code

- Can be used for a bottom up approach for simplifying code
  - Incremental adoption possible (if memory layout is unchanged)

- Lightweight compiler infrastructure (self maintainable)
  - Providing cross kernel optimizations

**The Laplacian operator with GGDML (as part of Fortran/C)**

```
FOREACH cell IN GRID
    lap(cell) = 4*h(cell) - (REDUCE(+, N={1..4}, h(cell%neighbour(N)))
END FOREACH
```
AIMES Experiments to Show Layout Dependency

Test kernel

- Part of an icosahedral modeling testbed
- Two target architectures: CPU and GPU (unified memory)
- Parallelization: OpenACC for GPU and OpenMP for CPU
- Two memory layouts (3D vs. 1D)
- 5, 7, and 9 point stencils

Configuration

- CPU: Ivy Bridge E5-2690 v2 3.0GHz (SP: 240 GFLOP/s)
- GPU: Nvidia K80 (SP: 6 TFLOP/s)
- GPU: P100 (SP: 9-10 TFLOP/s)
- Compiler: PGI 17.5 C
## AIMES Experiments

### Performance

<table>
<thead>
<tr>
<th>Stencil</th>
<th>CPU Performance (GFlops/s)</th>
<th>K80 GPU Performance (GFlops/s)</th>
<th>P100 GPU Performance (GFlops/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal 3D array</td>
<td>1D addressing</td>
<td>Normal 3D array</td>
</tr>
<tr>
<td>5</td>
<td>71</td>
<td>72</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>97</td>
<td>97</td>
<td>93</td>
</tr>
<tr>
<td>9</td>
<td>112</td>
<td>117</td>
<td>102</td>
</tr>
</tbody>
</table>

- Memory layout’s impact on performance is high
- Caching on the GPU added ~25% performance
CPU Measurements Compiler Stuff

- Previous experiments for CPU
  - Div, Rad, Grad stencil kernels
  - Skylake CPU
- Explored opt. of mem. layout
  - 3D and 1D transformation
  - Hilbert filling curves & HEVI
  - With various compilers
    - Intel, GCC, CLang
- Best layout depends on compiler!
Summary

- Scientists should harness methods to improve readability
  - As close as possible to the domain's typical code formalization
  - Abstracting from technical details
    - Compute backend, memory layout, loops, cache mgmt
    - Supporting (semi) automatic optimization / autotuning
- Separation of concerns eases understanding/speeds up dev.
  - Scientist – scientific programmer – computer scientist
  - Abstraction from memory layout
- We are working on a generic tool to reduce code replication
  - Providing a customizable DSL suitable for any domain
  - Exploiting optimization strategies beyond compiler capabilities
- Community could define language(s) to express their problems

Other tools relevant for atmospheric modeling:
- BLAS-Like Library Instantiation Framework (BLIS)
- Firedrake (PDE solver system)
Workshop Exascale I/O for Unstructured Grids (EIUG)

When: Monday/Tuesday 25th/26th of Sept.
Where: Hamburg, DKRZ
Speakers: Storage experts, domain experts
Funding is available!
https://wr.informatik.uni-hamburg.de/events/2017/eiug