AMRStencil: An Embedded Domain Specific Language (eDSL)

What is it?

• Class and template library for writing AMR Stencil codes.
• Embedding language is C++11
  • *Embedded*: can be used with vendor compiler, gets better with special compilers.
  • Very big improvement in language with lots of support.
• Captures what we have found to be the essential aspects of AMR Stencil codes
• If you have the ROSE compiler extension for AMRStencil, THEN you can enjoy cross-platform high performance execution.
• ROSE by no means has a lock on the AMRStencil API. We welcome other implementations
• AMRStencil default implementation ships with serial, MPI, and OpenMP in C++11
• Main abstractions
  • Stencil, Box, RectMDArray, LevelData
  • Stencils have well-defined symbolic calculus
    • S1(S2(A)) = (S1*S2)(A) \quad S1(c1*A)+S2(A) = (c1*S1+S2)(A)
    • fusion, performance models, polyhedra, etc are all much easier to analyze.

Brian Van Straalen (bvstraalen@lbl.gov)
The Goal

• Write it in C++
• Get rid of our Fortran kernels (our current pointwise and stencil DSL and multi-dim array)
  • Bury ChomboFortran in the revision control system.
• John Bell’s Paraphrase: “I’m willing to rewrite my code, once, then I’ll retire”
  • In reality, Chombo will be rewritten.
  • A Better, more agile Chombo
• AMRStencil DSL does not specify any parallelism or data layout.
  • Jeff Larkin’s “descriptive”, taken further.
  • In RAJA speak, user code has no exec policy
  • In Kokkos-speak, user code has no Space or Layout statements.
  • no pragmas, no directives, no memory model, no placement, no mapping, no target
• Move real application frameworks onto the real target DSL completely.
  • I’m sort of done with mini-apps.
• Create the correct place to put CS effort (profile hooks, control of loops)
A Question of Binding

• AMRStencil attempts to be very clear about what is compile-time bindable and what must remain runtime bound
• A Stencil object is compile-time (requires lots of constexpr use in header files to get it all pinned down)
• Box is a run-time object (low and high corners), subject to adaptive mesh refinement
  • Array location and bounds are run-time
  • certain properties of Box are compile-time (modular sizes).
• Stencils meet Boxs at run-time.
  • A significant difference from traditional stencil DSLs, which associate Stencils with arrays at compile time.
AMR Stencil -- Example: Geometric Multigrid

Multigrid::relax(LevelData<double, 1> & a_phi,
const LevelData<double, 1> & a_rhs)
{
    for (int iter = 0; iter < m_maxRelax; iter++)
    {
        a_phi.exchange();
        BoxLayout bl = a_phi.getBoxLayout();
        BLIterator blit(bl);
        for (blit.begin(); blit != blit.end(); ++blit)
        {
            Box bx = bl[*blit];
            RectMDArray<double, 1> temp(bx);
            temp |= m_Laplacian(phi[*blit], bx);
            temp -= a_rhs[*blit];
            temp *= m_lambda;
            a_phi[*blit] += temp;
        }
    }
};

\[ \phi := \phi + \lambda (\Delta^h(\phi) - \rho) \]

Apply stencil expressed as a linear combination of shift operators. Replaces multiple nested loops.

Iterate over patches

Increment solution with multiple of residual.

Highly expressive: complete implementation ~ 150 LOC.

Uses high-level description of block-structured stencil operations. Structured-grid stencil language, plus BoxLib / Chombo abstractions for unions of rectangles.

Opportunities for parallelism: over patches, over points in a patch. High-level expression of dependencies (e.g. stencil operators; exchange(), iterators).

Other examples under development: AMR Elliptic, compressible flow benchmarks.

ONLY program the algorithm essentials, leave everything else to DSL.
AMR Shift Calculus DSL optimization for x86/CPU with SIMD extensions

- High-level, user-friendly description of stencils, domain-specific information enable the generation of clean loop-based code that can be optimized with ROSE/PolyOpt
- Dedicated high-order stencil optimization pass in PolyOpt:
  1. Program transformations using associative/commutative properties of stencil convolutions
  2. Target-specific code synthesis for SIMD ISA of the stencil application

Results
- Setup: 4-core Intel Core i7-4770K Haswell processor with AVX2 SIMD, Intel ICC compiler
- box size=64. Execution of the stencil, double-precision data, no fusion across operators

<table>
<thead>
<tr>
<th>Theoretical Peak: 112</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
</tr>
<tr>
<td>8th-order mixed 2nd deriv 2d 64pts</td>
</tr>
<tr>
<td>Simple codegen</td>
</tr>
<tr>
<td>+ parallelization</td>
</tr>
<tr>
<td>+ PolyOpt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSL input</td>
</tr>
<tr>
<td>Generated code</td>
</tr>
</tbody>
</table>
How to overcome exascale challenges

- Generation of Complex Code for 10 Levels of Memory Hierarchy with SW managed cache
  - 4th order stencil computation from CNS Co-Design Proxy-App

<table>
<thead>
<tr>
<th>Memory Hierarchy</th>
<th>2 Level</th>
<th>3 Level</th>
<th>4 Level</th>
<th></th>
<th>10 level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSL Code</td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto Generated Code</td>
<td>446</td>
<td>500</td>
<td>553</td>
<td></td>
<td>819</td>
</tr>
</tbody>
</table>

- Code size of autogenerated code
Challenges

• Default AMRStencil uses some template metaprogramming
  • Most ideas seen here already: forall, lambda bodies, multidimensional array
  • Metaprogramming helps the vendor compiler do a decent job
    • Template spec will generate 1 output. Like a good language spec should
    • Performance models and auto-tuning need to explore hundreds of variants
  • Hot-shot template techniques create more headaches for an augmented compiler tool.
    • Giving Dan Quinlan something to do on his weekends.

• lambdas with side-effects can really mess up debugging.
  • …I can almost make Stencils constexpr

• Not every Stencil is knowable at compile-time
  • Embedded Boundary Chombo
    • Stencil points and weights from least-squares solve
  • Currently using runtime stencil playback

• As a parting shot: virtual functions are the modern callback
  • virtual functions are how you plug physics into frameworks
    • Separation of Concerns (SoC)
    • Layered Designs

Brian Van Straalen (bvstraalen@lbl.gov)
Anshu Dubey, Dan Quinlan, Phil Colella, Dan Graves, Terry Ligocki