GridTools: A C++ Library for Computations on Grids

Mauro Bianco, Paolo Crosetto, Oliver Fuhrer, Stefan Moosbrugger, Carlos Osuna, Hannes Vogt, Thomas C. Schulthess

Targeted Applications
- Weather and climate simulations
- Regional and global
- Regular grids
- Multidimensional arrays
- Structured grids
- Icosahedral
- A- and C-grids
- Regular tesselations
- Possibly others

Properties & Requirements
- Regular grids and stencils are main motifs
- Stencils are typically memory bandwidth bound
- Applications run many stencils in sequence
- Compilers find these problems hard to optimize
- Typical time-loop optimizations are unpractical
- Portability of performance
- Maintainability and separation of concerns
- Between application developers and performance specialists

Stencil Dependencies
- GridTools is for complex problems
- Operator Splitting for
  - Tractability
  - Maintainability
- Dependencies b/w operators
- Time and space dependencies
- GridTools provides easy composition of the stages

Multi-Stage Stencils
- New multi-stage stencils
- Maintainability and portability
- Tractability
- Enable design and run time optimizations
- Easy re-usability
- Separation of concerns
- Encapsulation

Structured grids
- Regular grids
- Weather and climate simulations

Regular tesselations
- Icosahedral
- Other

GridTools Ecosystem
- Computations on Grids
- Data structures
- Dependencies

Multi-Stage Stencil Composition
auto horizontal_diffusion = make_computation<BACKEND> {
  make_multistage_executor<forward>,
  make_stages<
    lap_f, (lapf, in),
    make_independent(
      make_stage<
        flx_f, (in, lapf),
        make_stage<
          fly_f, (in, flx_f, lapf),
          make_stage<
            comb_f, (out),
            make_arg_list<
              make_in_out_accessor, cells),
            data_fields, coords>
    )
  );
  horizontal_diffusion->run();
}

Stencil Operator
struct lap_function {
  typedef in_accessor> out;
  typedef in_accessor<, extent<1,1,1>> in;
  typedef arg_list = make_arg_list<out, in>;
  template<typename Evaluator>
  static void do(Evaluator& eval) {
    eval(out()) = eval(in()) +
      in(0, 0, 0) +
      in(-1, 0, 0) +
      in(0, 1, 0);
  }
};

Grid-Independent Operator
template < uint_t Color > struct sum_on_cells {
  using in = in_accessor<, cells, extent<1,1,1>>;
  using out = inout_accessor<, cells>;
  using arg_list = make_arg_list<in, out>;
  template < typename Evaluator >
  static void do(Evaluator& eval) {
    eval(out()) = eval(in_cells(<
      [double, double], return _s_r(),
      0.0, in());
  }
};

Traditional Approach
for i in 1, k = 1, j = 1
  for k = 1, j = 1
    lap(i, j, k) = d*(in(i, j, k) -
      (in(i-1, j, k) +
      in(i, j, k-1) +
      in(i-1, j, k-1))
    )
    then fly(i, j, k) = 0;

for i in 1, k = 1, j = 1
  for k = 1, j = 1
    flx(i, j, k) = lap(i+1, j, k)-
      (flx(i+1, j, k) +
      flx(i+1, j, k-1) +
      flx(i+1, j, k-1))
    if (flx(i, j, k) +
      flx(i, j+1, k) +
      flx(i, j+1, k-1) +
      flx(i, j+1, k-1))
    then fly(i, j, k) = 0;

for i in 1, k = 1, j = 1
  for k = 1, j = 1
    out(i, j, k) = in(i, j, k) -
      (flx(i, j, k) -
      flx(i+1, j, k) +
      fly(i, j, k) -
      fly(i+1, j, k));

Impact of Fusion

Incremental Optimization

References

Contact
Mauro Bianco
Swiss National Supercomputing Centre
Email: mbianco@cscs.ch
Phone: +41 (0) 91 610 8279