Why DSLs are Important to the DoE Exascale Mission

Saman Amarasinghe

MIT

D-TEC

Outline

Problem of (exa)Scaling High Performance Programs

Broader Impact of DSLs: The Halide Story

DoE Application Need for DSLs – Anshu Dubey

D-TEC

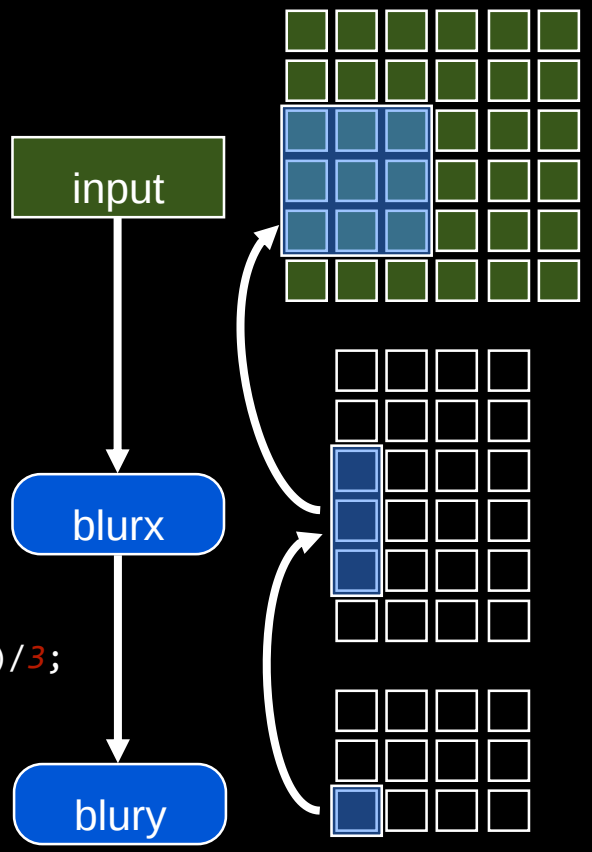
Evolving Code With Machines

- Recent history: HPC machines evolving at incredible rate
- New topologies, new architectures, accelerators, GPUs, new interconnects, increased hierarchies, etc.
- Traditional approach means rewriting code to perform optimally on each new platform

D-TEC

A simple example: 3x3 image blur

```
void box_filter_3x3(const Image &in, Image &blury) {
        blurx(in.width(), in.height());  // allocate blurx array


    for (int y = 0; y < in.height(); y++)
       for (int x = 0; x < in.width(); x++)
         blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;


    for (int y = 0; y < in.height(); y++)
       for (int x = 0; x < in.height(); x++)
        blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;
```



Hand-optimized C++

11x faster
(quad core x86)

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21846);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    __m128i blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         b = _mm_loadu_si128((__m128i*)(inPtr+1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

- Tiled, fused
- Vectorized
- Multithreaded
- Redundant computation
- *Near roof-line optimum*

D-TEC

(Re)organizing computation is hard

Optimizing parallelism, locality requires **transforming program & data structure.**

What transformations are *legal?*

What transformations are *beneficial?*

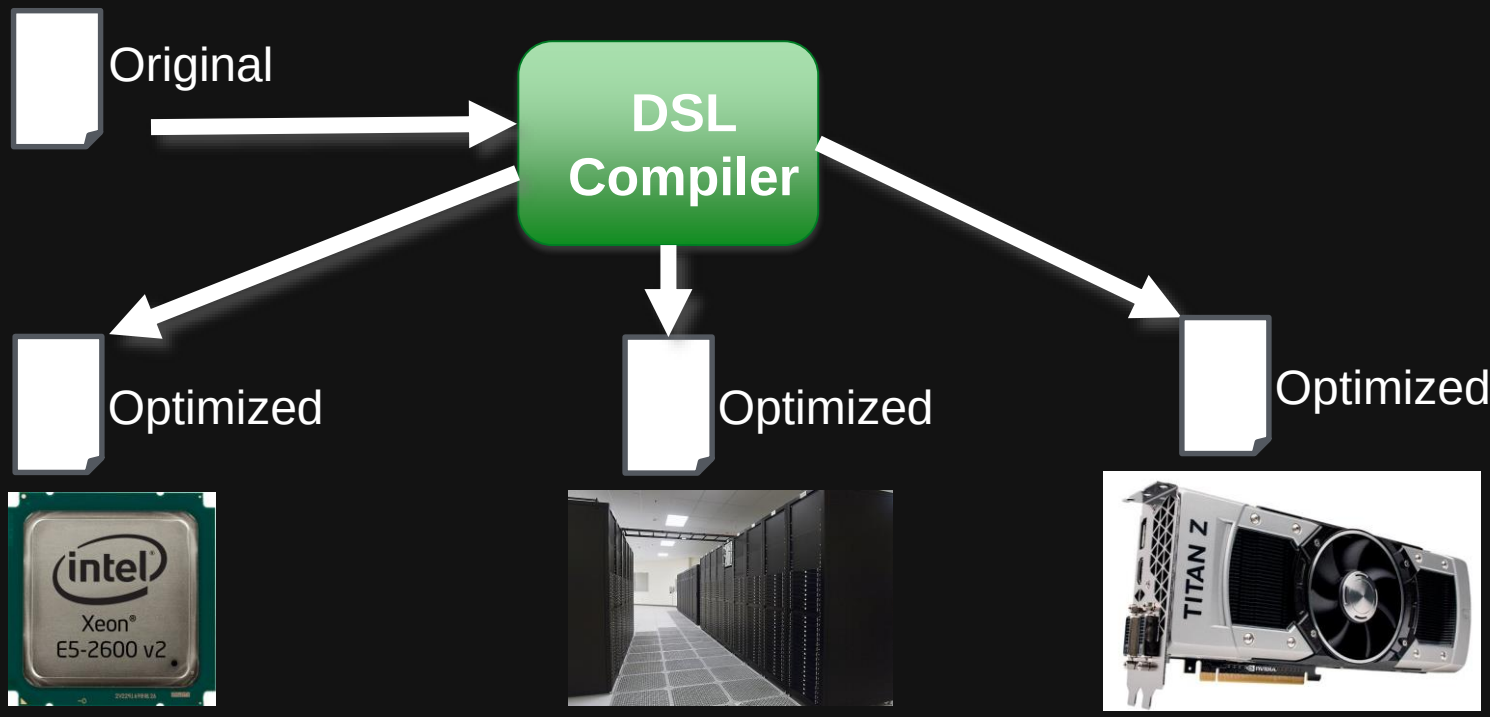
Libraries don't solve this:
BLAS, IPP, MKL, OpenCV, MATLAB

optimized kernels compose into inefficient pipelines (no fusion)

D-TEC

Domain-Specific Languages: The Last Rewrite

- Instead of rewriting every application on every new platform, express as a set of domain-specific code
- Separate *algorithm* from optimizations



Halide: Simpler, Faster, Scalable

with Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand

Reference: 300 lines C++

Adobe: 1500 lines

3 months of work

10x faster *(vs. reference)*

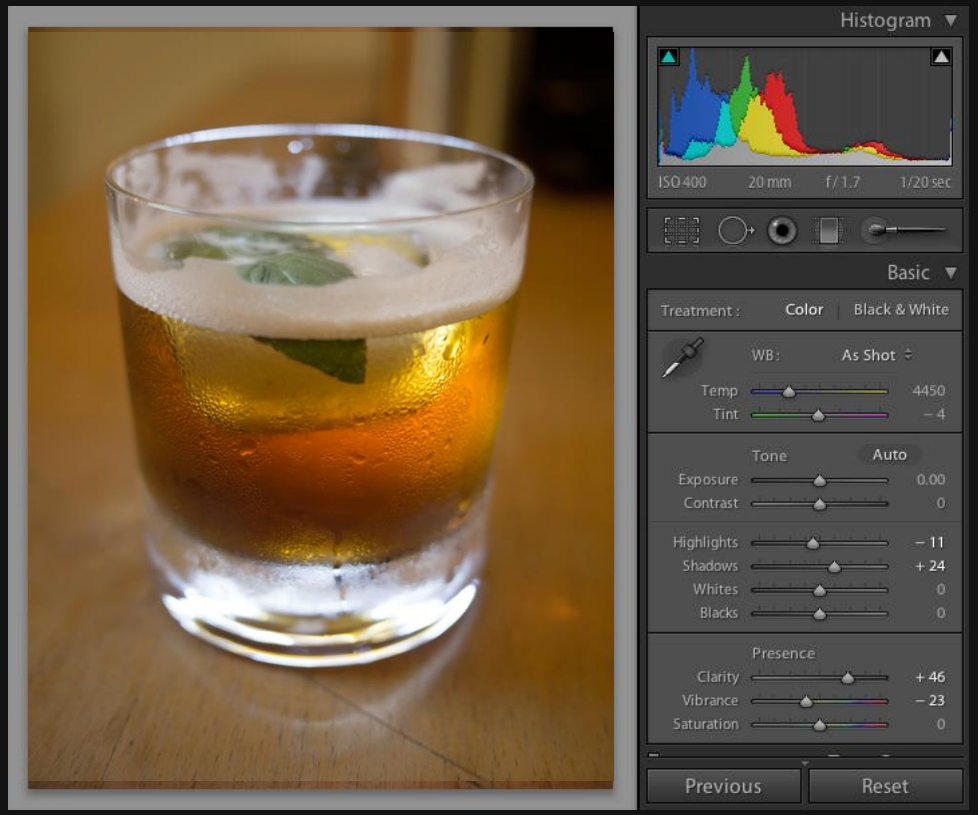
Halide: 60 lines

1 intern-day

20x faster (vs. reference)

2x faster (vs. Adobe)

GPU: 90x faster
(vs. reference)



Halide's answer:

Decouple algorithm from schedule

Algorithm: ***what*** is computed

Schedule: ***where*** and ***when*** it's computed

D-TEC

Decupling Algorithm from Schedule

Optimized C++

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21846);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    __m128i blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         b = _mm_loadu_si128((__m128i*)(inPtr+1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
```

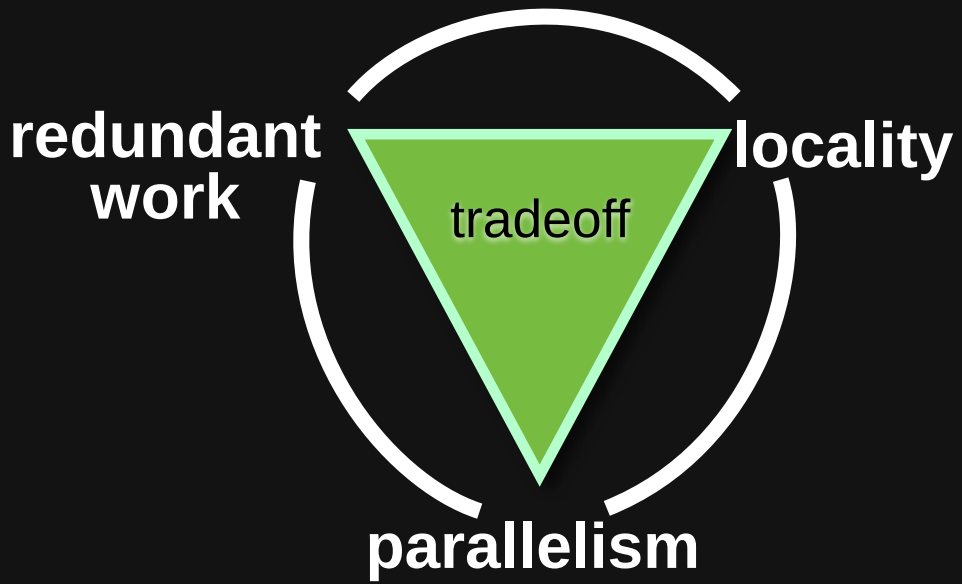
Halide I: Algorithm

```
void box_filter_3x3(const Image &in, Image &blury) {
  Image blurx(in.width(), in.height());  // allocate blurx array

  for (int y = 0; y < in.height(); y++)
    for (int x = 0; x < in.width(); x++)
      blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

  for (int y = 0; y < in.height(); y++)
    for (int x = 0; x < in.width(); x++)
      blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;
}
```

Halide II: Schedule

Single Program, Multiple Optimization Paths

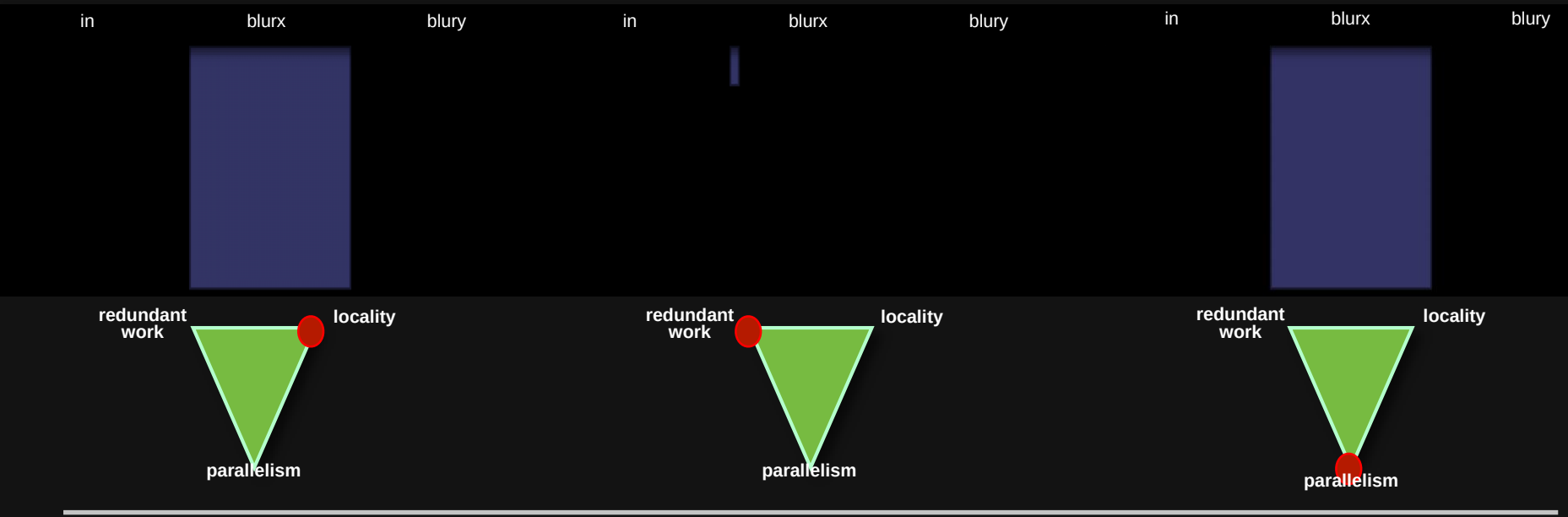
```
void box_filter_3x3(const Image &in, Image &blury) {
  Image blurx(in.width(), in.height());  // allocate blurx array

  for (int y = 0; y < in.height(); y++)
    for (int x = 0; x < in.width(); x++)
      blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

  for (int y = 0; y < in.height(); y++)
    for (int x = 0; x < in.width(); x++)
      blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x,
y+1))/3;
}
```

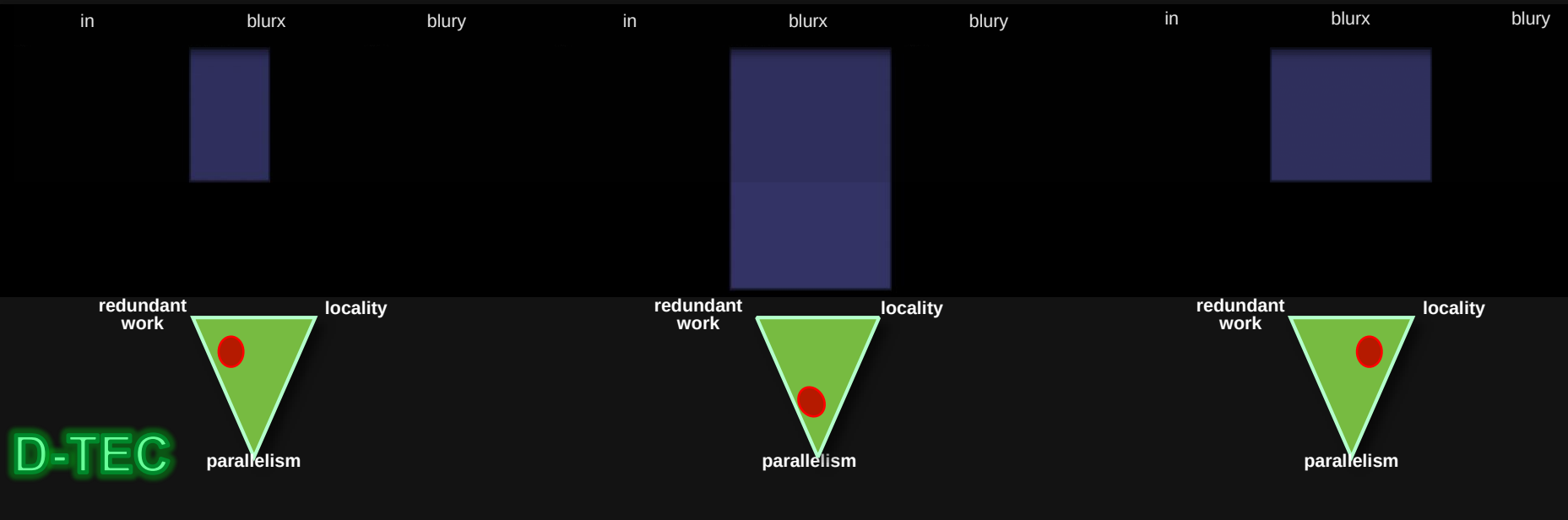


Once the algorithm is provided, can find the optimal schedule

- Complex tradeoff space, no obvious winner
- Each different tradeoff will lead to a “total rewrite of the program”
- Best schedule depends….on architecture, rest of the program, inputs etc. etc.
- Can search the tradeoff space manually
  - ultra-fast Hypothesis-ScheduleGen-CodeGen-Evaluation cycle
- Or can search the tradeoff space using autotuning

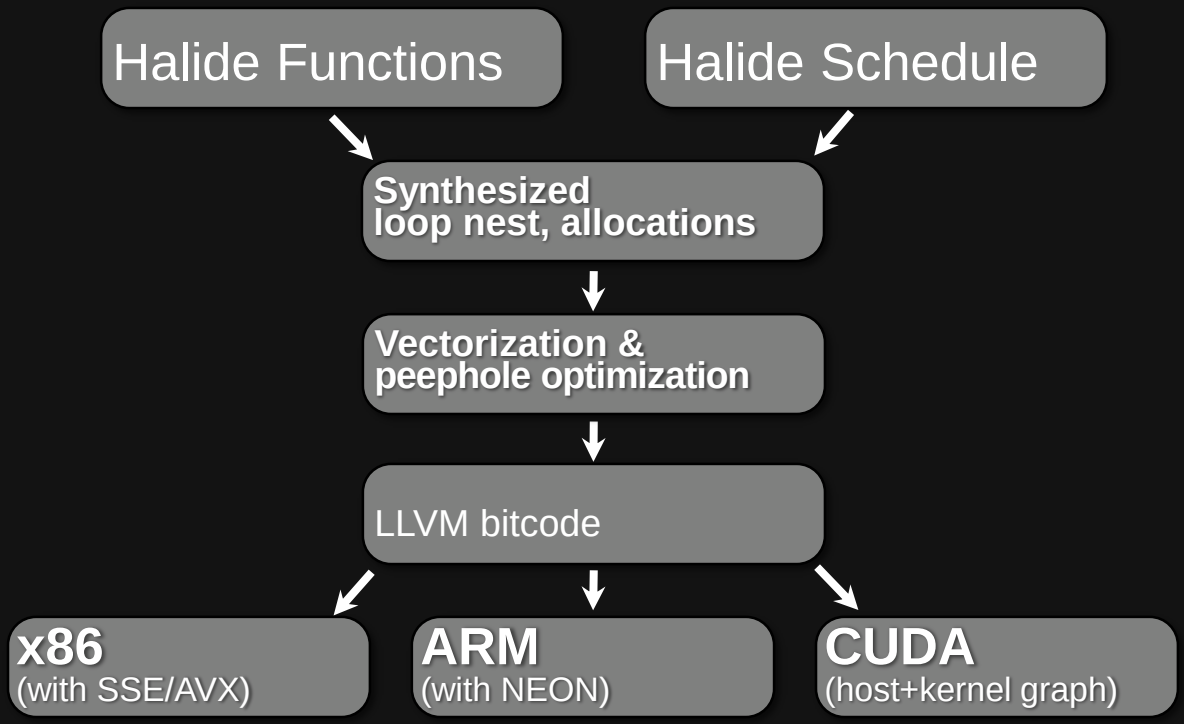
Single Algorithm multiple schedules



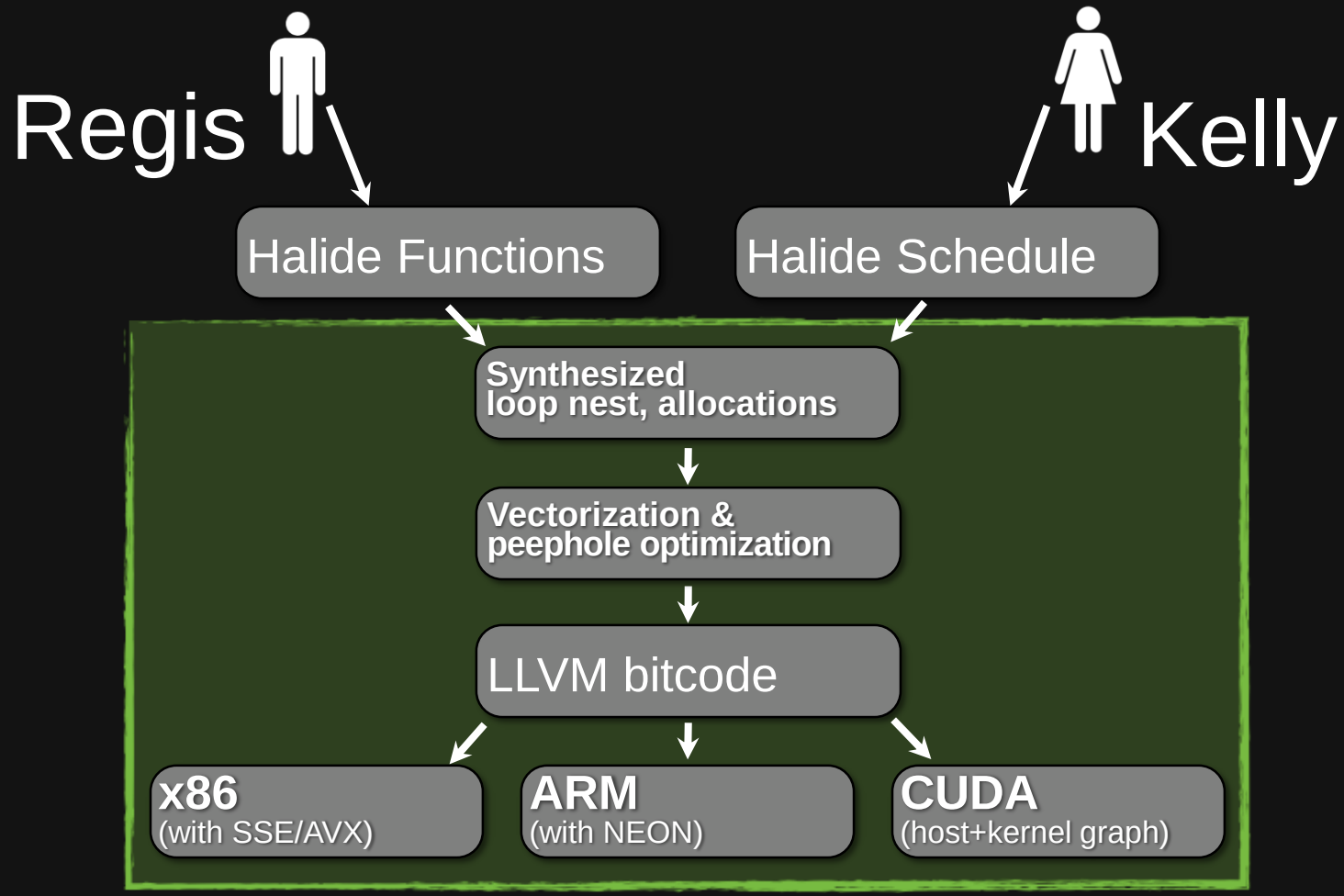


D-TEC

The Halide Compiler

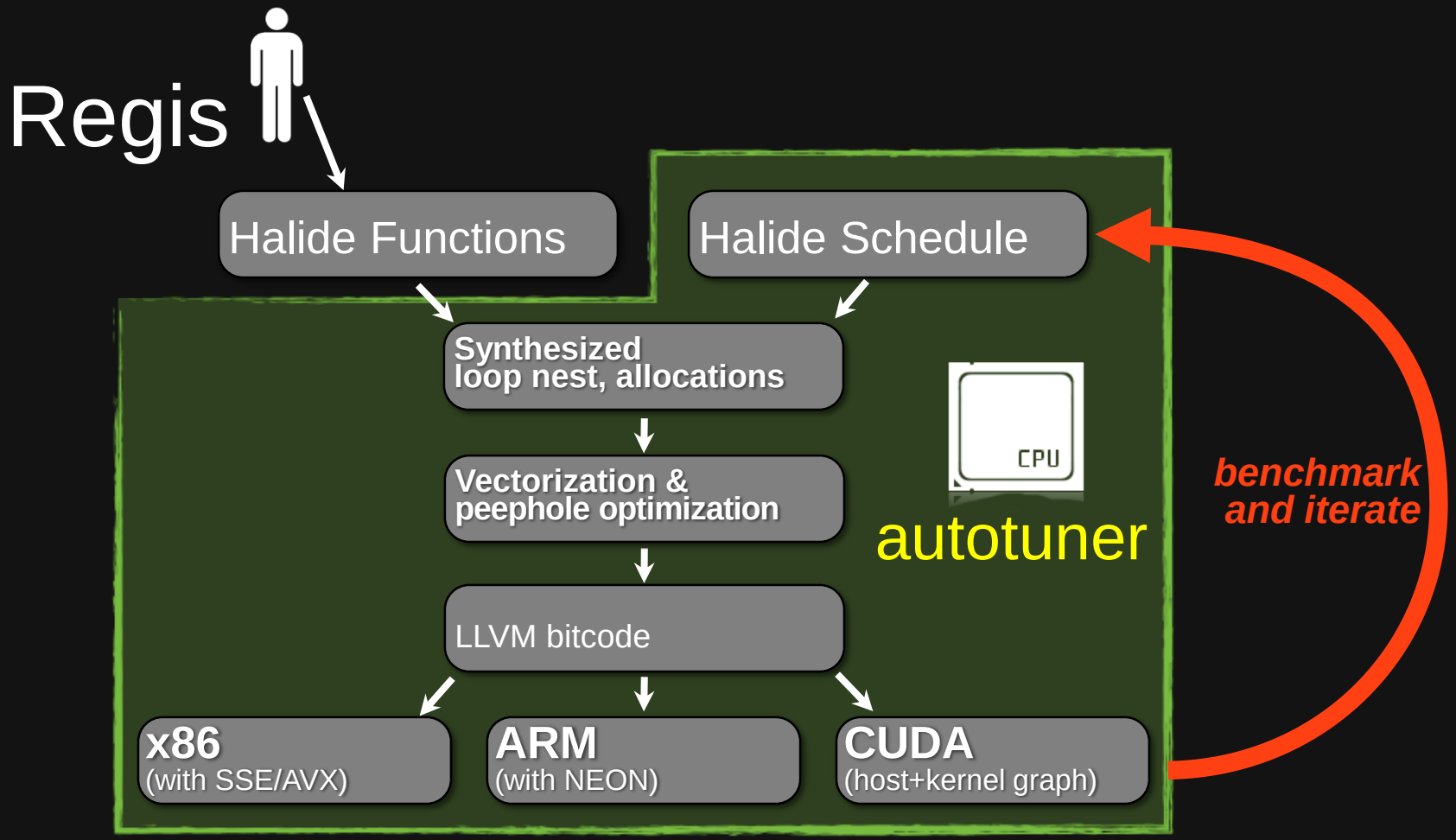


The Halide Compiler



D-TEC

The Halide Compiler



Uses OpenTuner framework

(at the technology marketplace)

D-TEC

x86	Speedup	Factor shorter
Blur	1.2 ×	18 ×
Bilateral Grid	4.4 ×	4 ×
Camera pipeline	3.4 ×	2 ×
“Healing brush”	1.7 ×	7 ×
Local Laplacian	1.7 ×	5 ×

GPU	Speedup	Factor shorter
Bilateral Grid	2.3 ×	11 ×
“Healing brush”	*5.9**	*7**
Local Laplacian	*9**	*7**

ARM	Speedup	Factor shorter
Camera pipeline	1.1 ×	3 ×

Autotuning time: **2 hrs to 2 days**

(single node) ***85% within < 24 hrs***

D-TEC

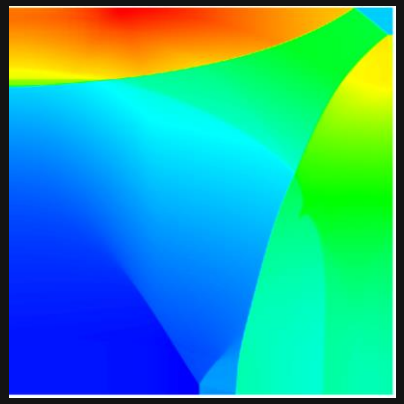
Mini app 1: Cloverleaf

Mantevo benchmark implementing Lagrangian-Eulerian hydrodynamics in 2D

>40% of runtime spent in `advec_mom` computational kernel (~255 LoC)

`advec_mom` ported to Halide and run on a 2x12 core Ivy Bridge Xeon

Ported to Halide by Shoaib Kamil



D-TEC

Mini app 1: Cloverleaf

Original OpenMP (partial)

```
!$OMP DO
    DO k=y_min-2,y_max+2
      DO j=x_min-2,x_max+2
        post_vol(j,k)= volume(j,k)+vol_flux_y(j  ,k+1)-vol_flux_y(j,k)
        pre_vol(j,k)=post_vol(j,k)+vol_flux_x(j+1,k  )-vol_flux_x(j,k)
      ENDDO
    ENDDO
  !$OMP DO
    DO k=y_min,y_max+1
      DO j=x_min-2,x_max+2
        ! Find staggered mesh mass fluxes, nodal masses and volumes.
        node_flux(j,k)=0.25_8*(mass_flux_x(j,k-1  )+mass_flux_x(j  ,k)  &
                       +mass_flux_x(j+1,k-1)+mass_flux_x(j+1,k)) ! Mass Flux
      ENDDO
    ENDDO
!$OMP END DO
!$OMP DO
    DO k=y_min,y_max+1
      DO j=x_min-1,x_max+2
        ! Staggered cell mass post advection
        node_mass_post(j,k)=0.25_8*(density1(j  ,k-1)*post_vol(j  ,k-1)
                                   +density1(j  ,k  )*post_vol(j  ,k  )
                                   +density1(j-1,k-1)*post_vol(j-1,k-1)
                                   +density1(j-1,k  )*post_vol(j-1,k  ))

      ENDDO
    ENDDO
!$OMP END DO
!$OMP DO
    DO k=y_min,y_max+1
      DO j=x_min-1,x_max+2
        ! Stagered cell mass pre advection
        node_mass_pre(j,k)=node_mass_post(j,k)-node_flux(j-1,k)+node_flux(j,k)
      ENDDO
    ENDDO
!$OMP END DO
```

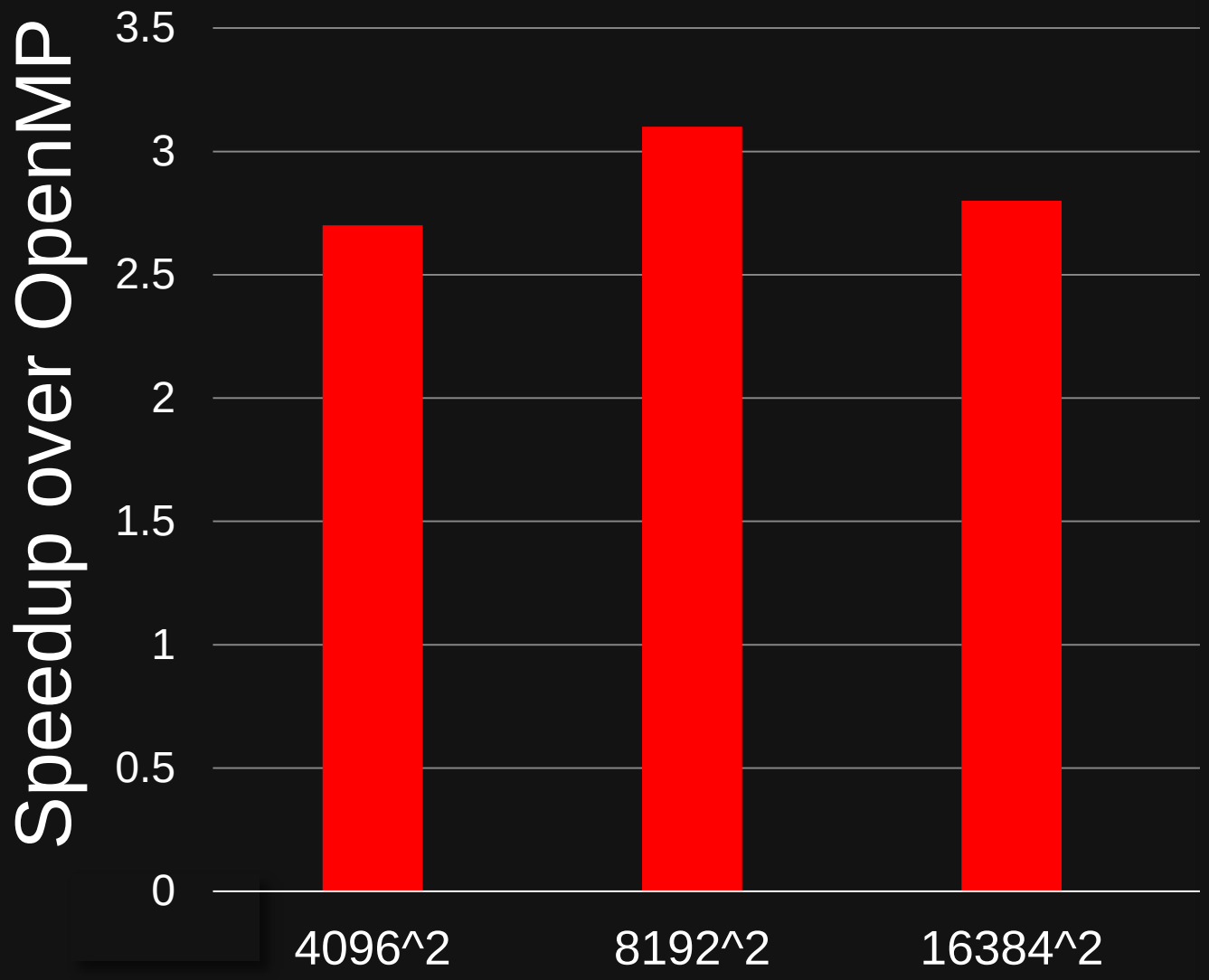
```
!$OMP DO PRIVATE(upwind,downwind,donor,dif,sigma,width,limiter,vdiffuw,vdiffdw,auw,adw,wind)
    DO k=y_min,y_max+1
      DO j=x_min-1,x_max+1
        IF(node_flux(j,k).LT.0.0)THEN
          upwind=j+2
          donor=j+1
          downwind=j
          dif=donor
        ELSE
          upwind=j-1
          donor=j
          downwind=j+1
          dif=upwind
        ENDIF
        sigma=ABS(node_flux(j,k))/(node_mass_pre(donor,k))
        width=celldx(j)
        vdiffuw=vel1(donor,k)-vel1(upwind,k)
        vdiffdw=vel1(downwind,k)-vel1(donor,k)
        limiter=0.0
        IF(vdiffuw*vdiffdw.GT.0.0)THEN
          auw=ABS(vdiffuw)
          adw=ABS(vdiffdw)
          wind=1.0_8
          IF(vdiffdw.LE.0.0) wind=-1.0_8
          limiter=wind*MIN(width*((2.0_8-sigma)*adw/width+(1.0_8+sigma)*auw/celldx(dif))/6.0_8,auw,adw)
        ENDIF
        advec_vel(j,k)=vel1(donor,k)+(1.0-sigma)*limiter
        mom_flux(j,k)=advec_vel(j,k)*node_flux(j,k)
      ENDDO
    ENDDO
!$OMP END DO
!$OMP DO
    DO k=y_min,y_max+1
      DO j=x_min,x_max+1
        vel1 (j,k)=(vel1 (j,k)*node_mass_pre(j,k)+mom_flux(j-1,k)-mom_flux(j,k))/node_mass_post(j,k)
      ENDDO
    ENDDO
!$OMP END DO
```

Halide (program only)

```
Expr e_post_vol = volume(j,k) + vol_flux_y(j,k+1) - vol_flux_y(j,k);
f_post_vol(j,k) = e_post_vol;

Expr e_pre_vol = f_post_vol(j,k) + vol_flux_x(j+1,k) - vol_flux_x(j,k);
f_pre_vol(j,k) = e_pre_vol;

Expr e_node_flux = 0.25f * (mass_flux_x(j,k-1)
                          + mass_flux_x(j,k)
                          + mass_flux_x(j+1,k-1)
                          + mass_flux_x(j+1,k));
f_node_flux(j,k) = e_node_flux;

Expr e_node_mass_post = 0.25f * (density1(j,k-1) * f_post_vol(j,k-1)
                               + density1(j,k) * f_post_vol(j,k)
                               + density1(j-1,k-1) * f_post_vol(j-1,k-1)
                               + density1(j-1,k) * f_post_vol(j-1,k));
f_node_mass_post(j,k) = e_node_mass_post;

Expr e_node_mass_pre = f_node_mass_post(j,k) - f_node_flux(j-1,k) + f_node_flux(j,k);
f_node_mass_pre(j,k) = e_node_mass_pre;

Expr upwind = select(f_node_flux(j,k) < 0.0f, j+2, j-1);
Expr donor = select(f_node_flux(j,k) < 0.0f, j+1, j);
Expr downwind = select(f_node_flux(j,k) < 0.0f, j, j+1);
Expr dif = select(f_node_flux(j,k) < 0.0f, donor, upwind);

Expr sigma = abs(f_node_flux(j,k)) / f_node_mass_pre(donor, k);
Expr width = celldx(j);
Expr vdiffuw = vel1(donor,k) - vel1(upwind,k);
Expr vdiffdw = vel1(downwind,k) - vel1(donor,k);

Expr auw = abs(vdiffuw);
Expr adw = abs(vdiffdw);
Expr wind = select(vdiffdw <= 0.0f, -1.0f, 1.0f);


Expr limiter = select(vdiffuw*vdiffdw > 0.0f, wind*min(width*((2.0f-sigma)*
                                                        adw/width+(1.0f+sigma)*auw/celldx(dif))/6.0f,
                                                        min(auw,adw)),
                                              cast(Float(64), 0.0f));

Expr e_advec_vel = vel1(donor,k) + (1.0f-sigma)*limiter;
f_advec_vel(j,k) = e_advec_vel;

Expr e_mom_flux = f_advec_vel(j,k) * f_node_flux(j,k);
f_mom_flux(j,k) = e_mom_flux;

Expr e_vel1 = (vel1(j,k) * f_node_mass_pre(j,k)
          + f_mom_flux(j-1, k)
          - f_mom_flux(j,k)) / f_node_mass_post(j,k);
```

D-TEC

Mini app 1: Cloverleaf



Mini app 2: CNS solver

Compressible Navier Stokes equation for Constant Viscosity and Thermal Conductivity.

Halide Port

- **Ported two kernals to Halide which consumes >50% of the run time (diffterm and hypterm).**
- **Used an auto-tuner to obtain high performance schedules for a SMP CPU**
- **Used a hand-generated schedule to get GPU results**
- **Halide uses loop fusion in diffterm VS. original fortran code to outperform in all kernal sizes that were tested**

Ported to Halide by Charith Mendis

D-TEC

Mini app 2: CNS solver

Fortran code

Halide code

C code

Halide advantages

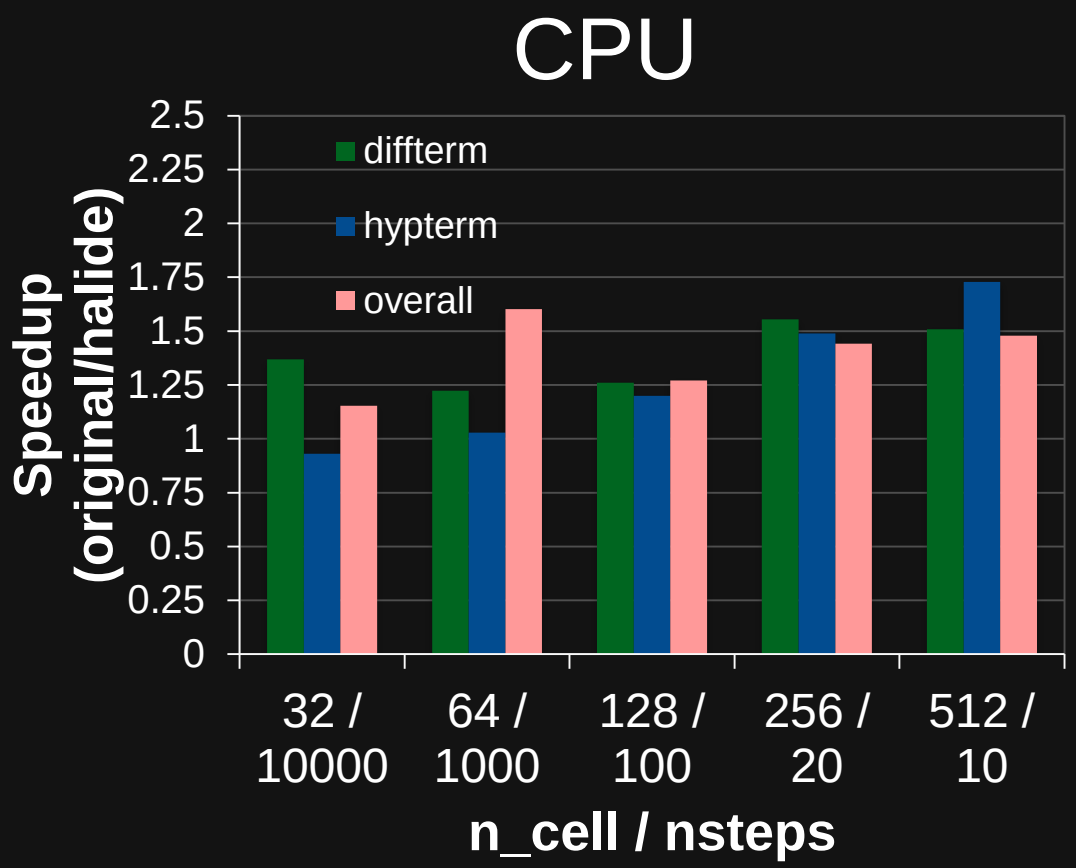
1. Can be directly ported without syntax changes from Fortran to Halide

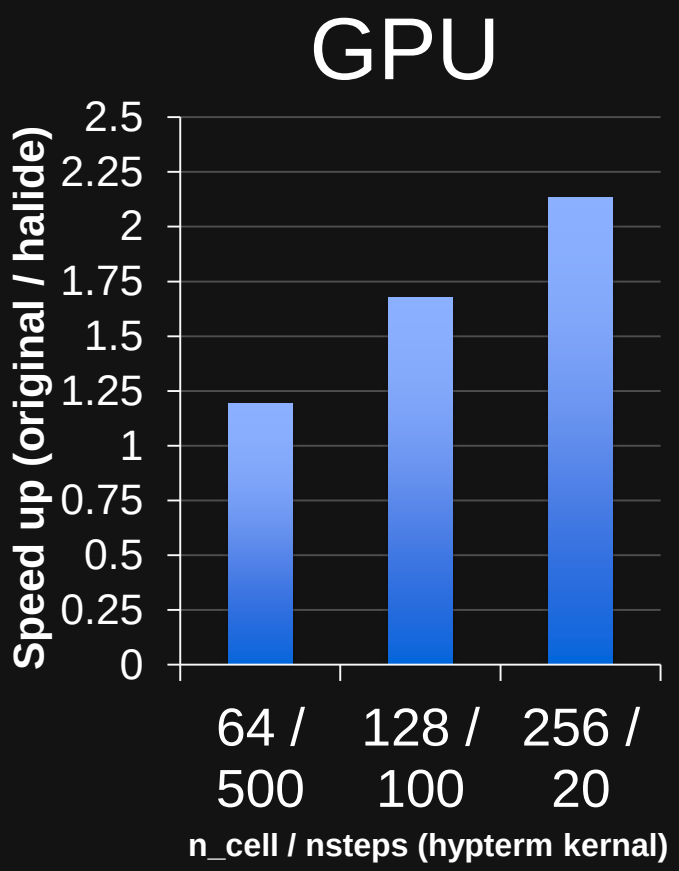
2. Abillity to try out various schedules without obscuring the algorithm

3. Lines of Code approximately equal to Fortran code and is much less than the equivalent C code

D-TEC

Mini app 2: CNS solver

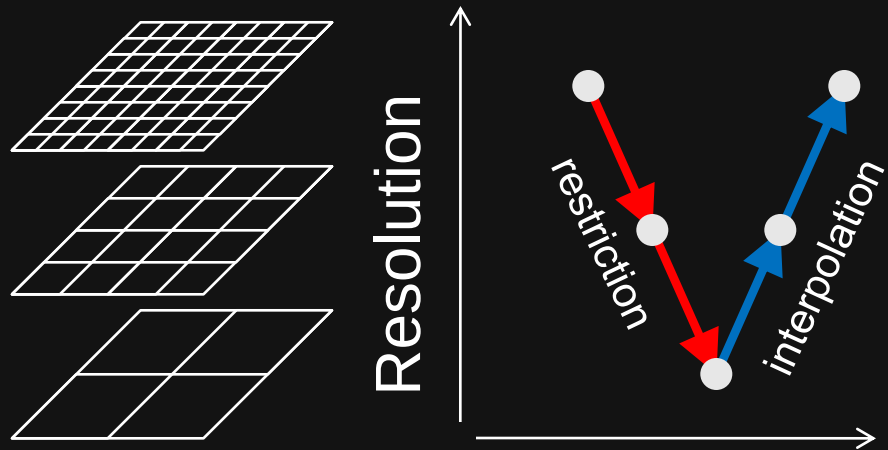




D-TEC

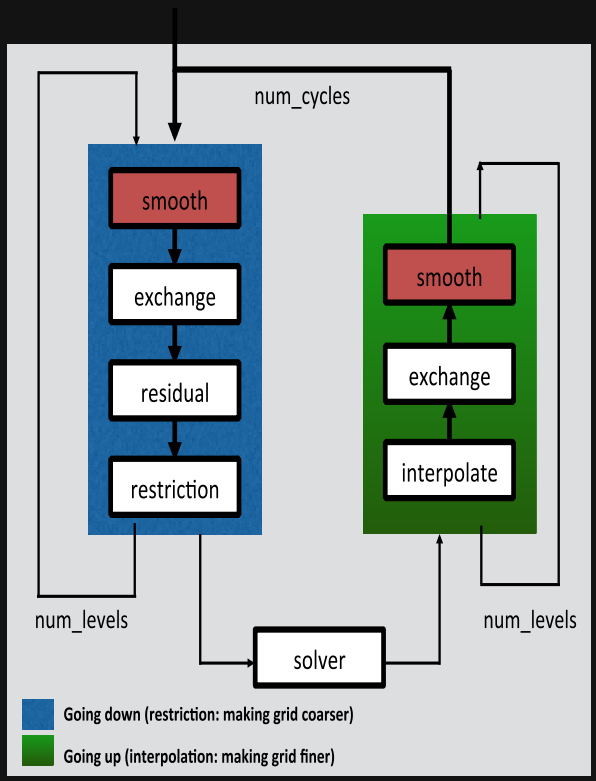
Miniapp 3: HPGMG

HPGMG is a compact benchmark designed to proxy the geometric MG solves found in applications built from AMR MG frameworks like CHOMBO or BoxLib.



Started with Sam's hand optimized code with fusion as our ORIGINAL

Ported to Halide by Rishi Khan



Mini app 3: HPGMG

Original Hand optimized

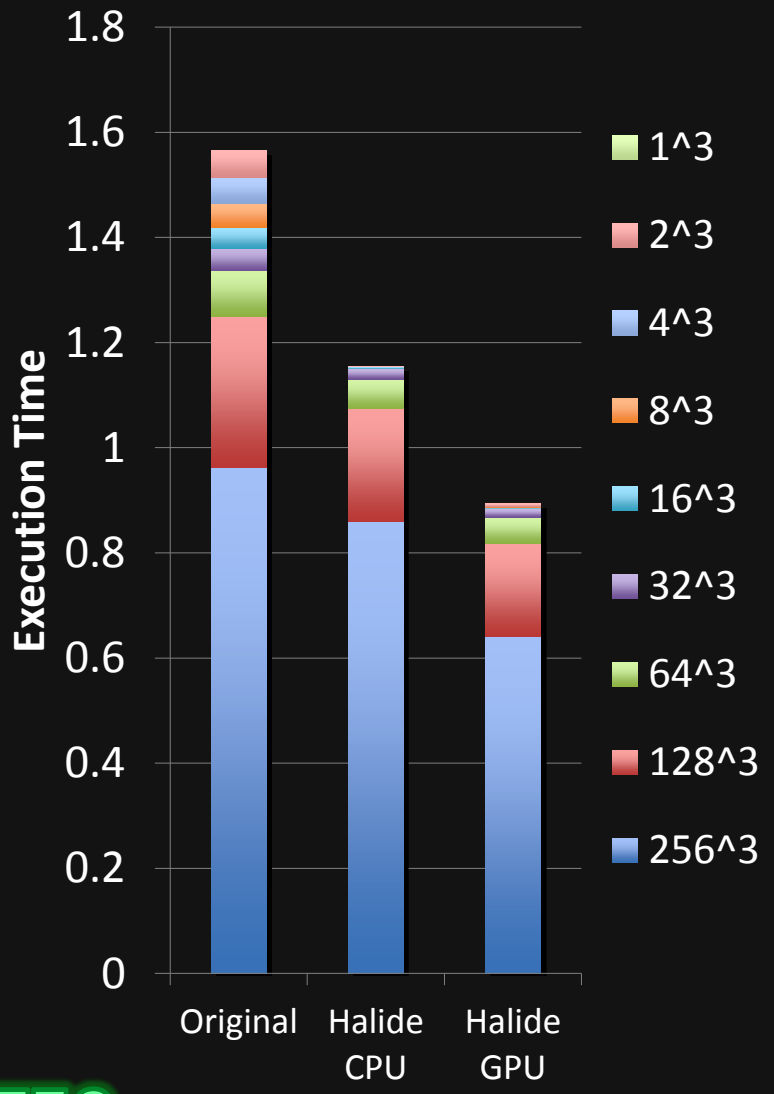
```
#define OMP_THREAD_WITHIN_A_BOX(threads_per_team) \
  if(threads_per_team>1) num_threads(threads_per_team) collapse(2)
int i,j,k;
#pragma omp parallel for private(k,j,i) OMP_THREAD_WITHIN_A_BOX(level->threads_per_box)
for(k=0-ghostsToOperateOn;k<dim+ghostsToOperateOn;k++){
for(j=0-ghostsToOperateOn;j<dim+ghostsToOperateOn;j++){
for(i=0-ghostsToOperateOn;i<dim+ghostsToOperateOn;i++){
  int ijk = i + j*jStride + k*kStride;
  double Ax_n = a*alpha[ijk]*x_n[ijk] - b*h2inv*(
     beta_i[ijk        ]*(valid[ijk-1      ]*( x_n[ijk] + x_n[ijk-1      ]) - 2.0*x_n[ijk])
   + beta_j[ijk        ]*(valid[ijk-jStride]*( x_n[ijk] + x_n[ijk-jStride]) - 2.0*x_n[ijk])
   + beta_k[ijk        ]*(valid[ijk-kStride]*( x_n[ijk] + x_n[ijk-kStride]) - 2.0*x_n[ijk])
   + beta_i[ijk+1      ]*(valid[ijk+1      ]*( x_n[ijk] + x_n[ijk+1      ]) - 2.0*x_n[ijk])
   + beta_j[ijk+jStride]*(valid[ijk+jStride]*( x_n[ijk] + x_n[ijk+jStride]) - 2.0*x_n[ijk])
   + beta_k[ijk+kStride]*(valid[ijk+kStride]*( x_n[ijk] + x_n[ijk+kStride]) -
2.0*x_n[ijk]));
  double lambda = 1.0 / (a*alpha[ijk] - b*h2inv*(
     beta_i[ijk        ]*(valid[ijk-1      ] - 2.0)
   + beta_j[ijk        ]*(valid[ijk-jStride] - 2.0)
   + beta_k[ijk        ]*(valid[ijk-kStride] - 2.0)
   + beta_i[ijk+1      ]*(valid[ijk+1      ] - 2.0)
   + beta_j[ijk+jStride]*(valid[ijk+jStride] - 2.0)
   + beta_k[ijk+kStride]*(valid[ijk+kStride] - 2.0)
   )) ;
  x_np1[ijk] = x_n[ijk] + c1*(x_n[ijk]-x_nm1[ijk]) + c2*lambda*(rhs[ijk]-Ax_n);
}}}

//… scheduling constraints in a different file
level->concurrent_boxes = level->num_my_boxes;
if(level->concurrent_boxes > omp_threads)level->concurrent_boxes = omp_threads;
if(level->concurrent_boxes <           1)level->concurrent_boxes = 1;
level->threads_per_box = omp_threads / level->concurrent_boxes;
if(level->threads_per_box > level->box_dim*level->box_dim)
   level->threads_per_box = level->box_dim*level->box_dim; // JK collapse
if(level->threads_per_box > level->box_dim*level->box_dim*level->box_dim/64)
   level->threads_per_box = level->box_dim*level->box_dim*level->box_dim/64;
if(level->threads_per_box<1)level->threads_per_box = 1;
```

Halide program

```
Func Ax_n("Ax_n"), lambda("lambda"), chebyshev("chebyshev");
Var i("i"),j("j"),k("k");
Ax_n(i,j,k) =  a*alpha(i,j,k)*x_n(i,j,k) - b*h2inv*(
    beta_i(i,j,k)  *(valid(i-1,j,k)*(x_n(i,j,k) + x_n(i-1,j,k)) - 2.0f*x_n(i,j,k))
  + beta_j(i,j,k)  *(valid(i,j-1,k)*(x_n(i,j,k) + x_n(i,j-1,k)) - 2.0f*x_n(i,j,k))
  + beta_k(i,j,k)  *(valid(i,j,k-1)*(x_n(i,j,k) + x_n(i,j,k-1)) - 2.0f*x_n(i,j,k))
  + beta_i(i+1,j,k)*(valid(i+1,j,k)*(x_n(i,j,k) + x_n(i+1,j,k)) - 2.0f*x_n(i,j,k))
  + beta_j(i,j+1,k)*(valid(i,j+1,k)*(x_n(i,j,k) + x_n(i,j+1,k)) - 2.0f*x_n(i,j,k))
  + beta_k(i,j,k+1)*(valid(i,j,k+1)*(x_n(i,j,k) + x_n(i,j,k+1)) - 2.0f*x_n(i,j,k)));
lambda(i,j,k) = 1.0f / (a*alpha(i,j,k) - b*h2inv*(
    beta_i(i,j,k)  *(valid(i-1,j,k) - 2.0f)
  + beta_j(i,j,k)  *(valid(i,j-1,k) - 2.0f)
  + beta_k(i,j,k)  *(valid(i,j,k-1) - 2.0f)
  + beta_i(i+1,j,k)*(valid(i+1,j,k) - 2.0f)
  + beta_j(i,j+1,k)*(valid(i,j+1,k) - 2.0f)
  + beta_k(i,j,k+1)*(valid(i,j,k+1) - 2.0f)));
chebyshev(i,j,k) = x_n(i,j,k) + c1*(x_n(i,j,k)-x_nm1(i,j,k))+
                   c2*lambda(i,j,k)*(rhs(i,j,k)-Ax_n(i,j,k));
```

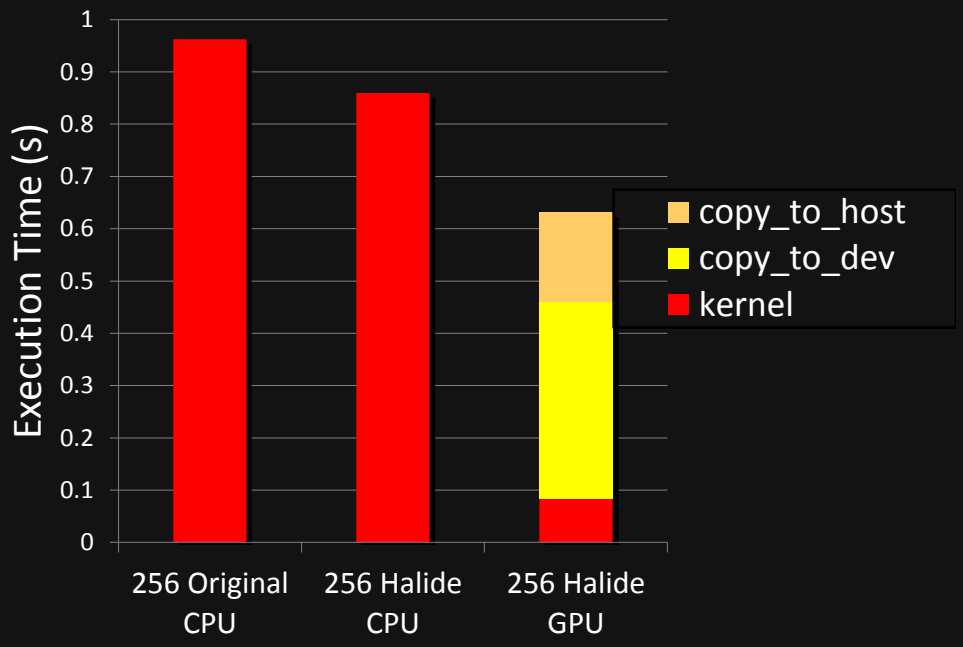
D-TEC

Mini app 3: HPGMG



Speedups

Stencil Size	Halide CPU	Halide GPU
256^3	1.12	1.50
128^3	1.34	1.63
64^3	1.54	1.73
32^3	2.14	2.75
16^3	11.52	11.31
8^3	67.43	67.73
4^3	326.96	143.10
2^3	548.78	8.89



D-TEC

Halide's Broader Impact

Google Glass camera pipeline

Google+ image pipeline

Exploring use in other Google projects

- 20+ engineers writing halide code & 2 experts writing the schedules

Adobe also exploring use in products

D-TEC

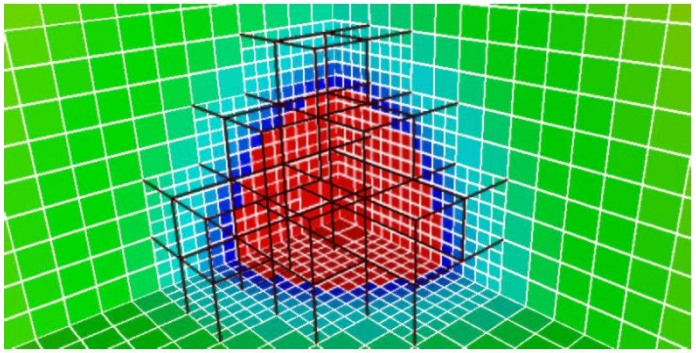
Why DSLs Are Desirable

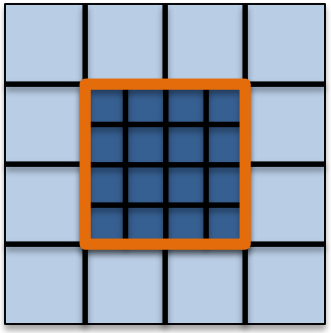
- **There is a fundamental tension between code maintainability, portability and performance**
- **Heterogeneity in hardware architecture exacerbates the problem**
  - Nearly impossible to write code that is portable across platforms in current high level languages that also performs well everywhere
  - The only option is code transformation to retain portability
- **Urgent need for abstraction in programming model between high-level math and currently available languages**

Why Are Embedded DSLs Attractive to Applications

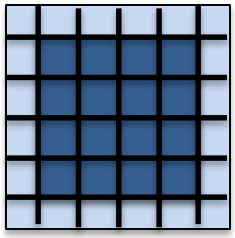
- **Often scientists inadvertently write code with optimization blockers**
  - Typical scientist coders not conversant with constraints of the compiler optimization
    - Compiler optimizations are by definition conservative: when in doubt don't optimize
  - Richer constructs help translation of algorithm into code without optimization blockers
  - if you make the problem easier for the compiler, you have a fighting chance to get good code.
- **Boutique solutions do not translate into production grade software**
  - Scientists are reluctant to add dependencies which have difficulty getting on new platforms
- **Applications won't use a language unless its longevity is guaranteed**
  - Enhancements embedded within an existing language make longevity more likely

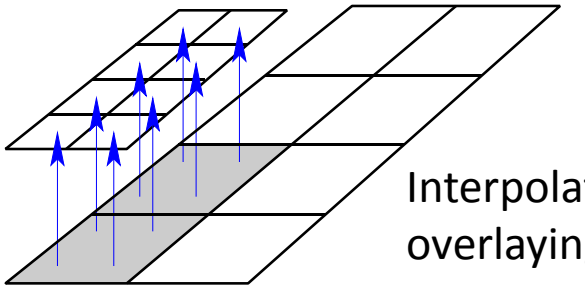
Applying a two-level AMR Operator



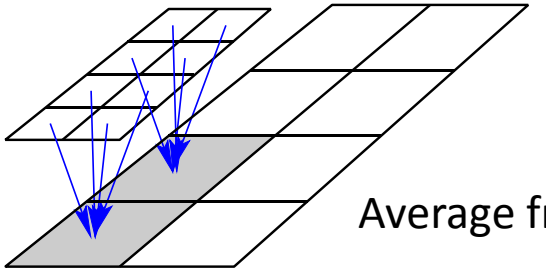


- Apply operator on the coarse grids
- Save fluxes at coarse-fine boundaries
- Fill ghost cells on the fine grids
- Apply operator on fine grids
- Increment fluxes at coarse-fine boundaries
- Apply flux correction at fine-coarse boundaries

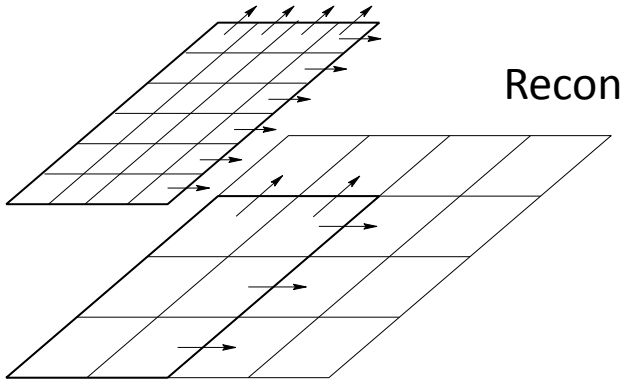




Interpolation from overlaying coarse cells



Average from fine cells

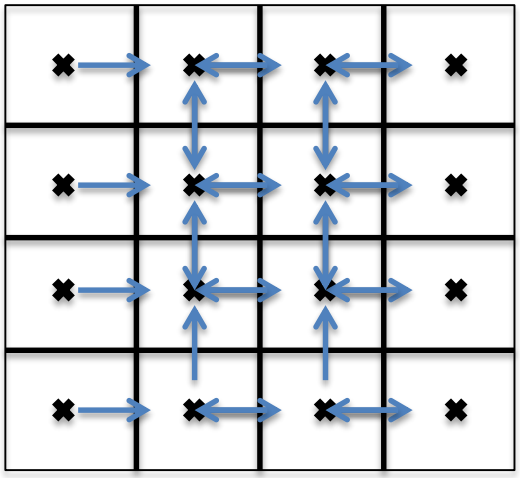


Reconcile fluxes

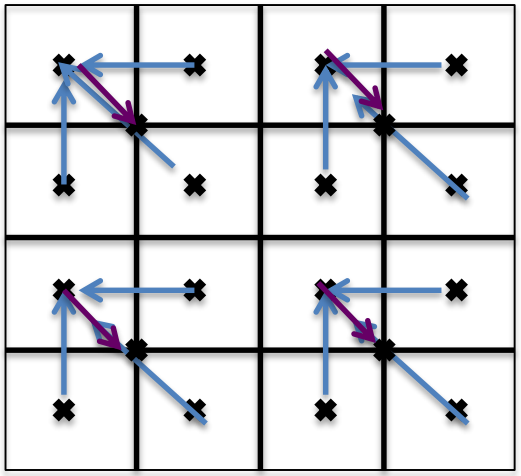
Basis for an EDSL - AMR Shift Calculus

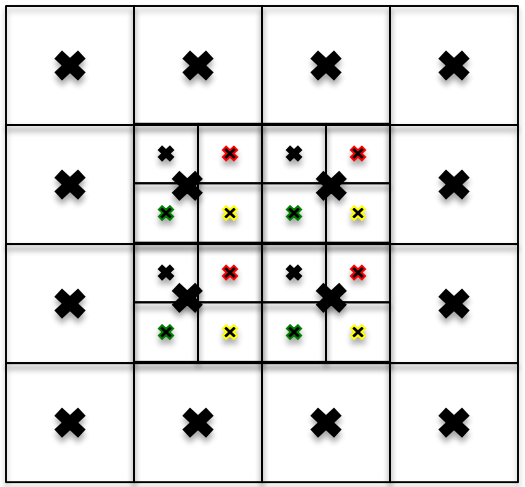
- **A stencil operator is a sum of shifts multiplied by corresponding coefficients**
  - Offset specified by the shift relative to the target
    - No explicit ijk indexing (dimension independent code)
  - Shifts don't say where they are applied
  - The coefficients could be scalars or tensors
  - + and * operators for adding and composing stencils
- **Applying the stencil operator**
  - Weighted sum of some points on the mesh
  - Support nested hierarchies (example in Dan's talk)
- **Stencils are known at compile time, where to apply them is specified at run time**
  - Provides rich set of opportunities for compiler optimizations provided there is suitable runtime support

Level Shift



Shifts between Levels





Conclusions

- The code is written in higher level semantics –more opportunities for optimization
  - Functional dependencies articulated through composition of stencils
    - Possible to fuse procedures knowing the stencil composition
  - Spatial component expressed through the source and destination points
    - Possible to do custom decomposition/coalition of space depending upon the target architecture
    - Also possible to do over-decomposition to exploit pipelining potential through runtime management
- Having an embedded DSL useful
  - A very small API for compilers/code translation tools to work with
  - Flexibility of high level language for the complex logic of composition
    - Also for parts of the algorithm that do not map to shift calculus

Extra slides

Operations defined on Stencils

- **(S1+S2) =>union(S1,S2); coefficients of common shifts get added**
  - S1=<1,0|C1>,<0,0|C2>, S2=<0,0|C3>,<0,-1|C4>
  - S1+S2= <1,0|C1>,<0,0|C2+C3>,<0,-1|C4>
  - Defined for Level S1 and S2
- **(S1*S2) =>convolve(S1,S2); Shifts get added, coefficients get multiplied**
  - S1=<1,0|C1>,<0,0|C2>, S2=<0,0|C3>,<0,-1|C4>
  - S1*S2=<1,0|C1*C3>, <1,-1|C1*C4>,<0,0|C2*C3>,<0,-1|C2*C4>
  - Defined when
    - S1 and S2 of the same type (Level, CtoF or FtoC)
    - One of S1 and S2 is Level and the other is a half shift or multilevel shift