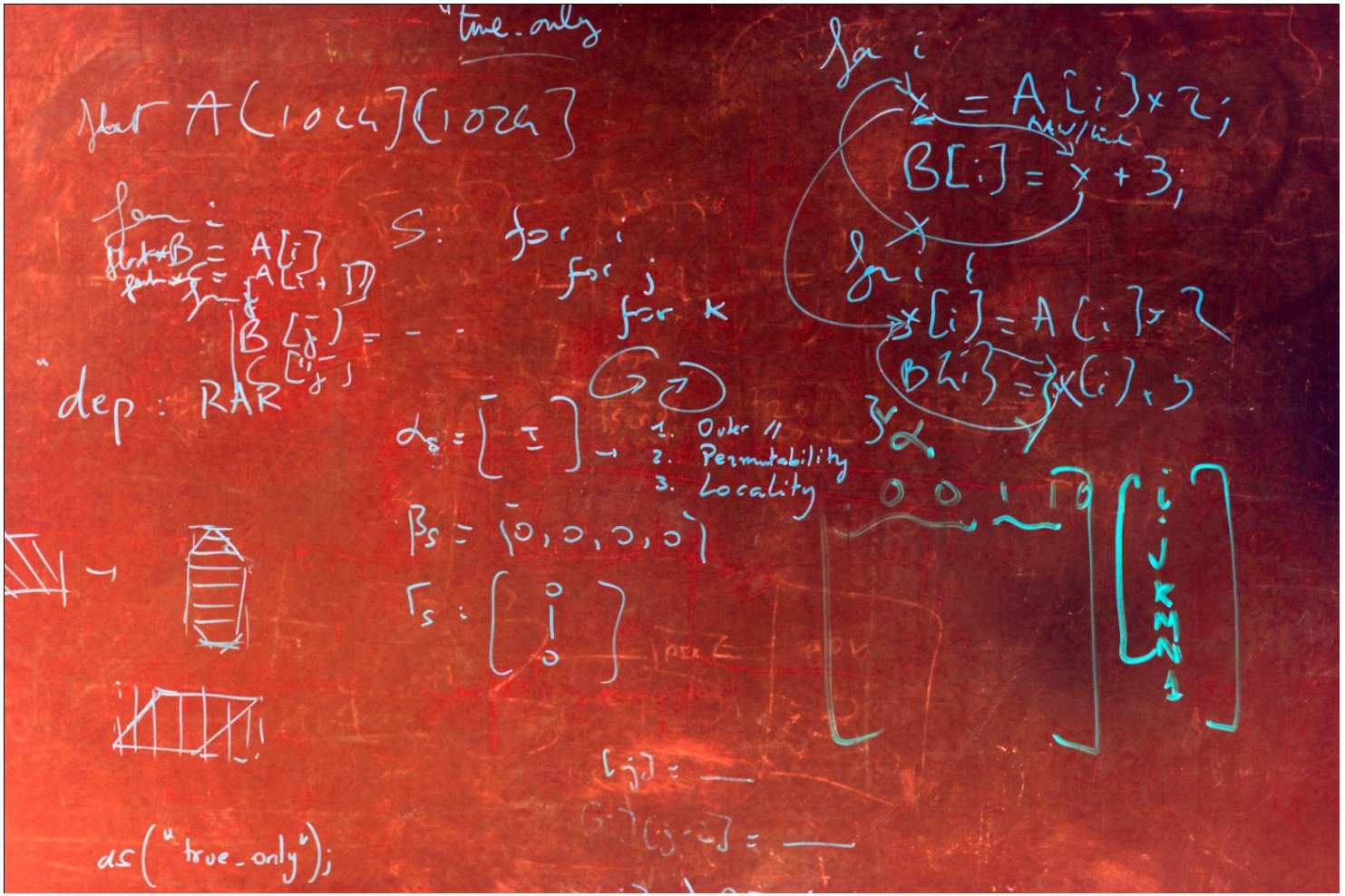
SANE: an Array Language for Sensor Applications

B. Meister, D. Wohlford, J. Springer, T. Benson, M. Baskaran, N. Vasilache, D. Campbell, R. Lethin



Outline

Context

Approach & Results

Conclusion & Future plans

CONTEXT

Exascale / Extreme Scale Constraints

Extreme flops/Watt needed in sensor applications

- Higher resolutions, more dimensions, higher rates
- Embedded: limited power (airborne)
- Best GPUs ~ 15-20 Gflops/Watt *if used at peak*
- Extreme Scale ~100 Gflops/Watt

Achieving 100 Gflops/W [ExascaleStudyGroup08]

Near-Threshold Voltage (NTV)

Greater concurrency

Lower power

Explicitly managed memories and data transfers

New dynamic adaptive execution models

Locality: always been condition for speed & power

- Competes with more and more concurrency
- Best solutions often lie in a tradeoff

Sensor applications

Image formation, change detection, adaptive imaging

- e.g., SAR, CCD, MTI, STAP

```
Foreach pixel
  Accumulator = 0
  Foreach pulse P in X
    Compute r = the range from the transmitter position
    for P to the pixel location
      Compute Roundtrip range R = 2r
      Compute range bin: bin = (R - R0) / ΔR
      w = bin – floor(bin);
      sample = (1-w) * X(P,floor(bin)) + w * X(P,floor(bin)+1)
      ideal = i*2*k*R   // i = sqrt(-1)
      Accumulator += e^ideal * sample
    end P
  end pulse
end pixel
```

Productivity in exascale programming

Developer must know about app (physics, required precision) and algorithms, AND target HW, parallelization and optimization techniques

Often, two-step development:

- App specialist prototypes using high-productivity language to validate algorithm
- Code monkey re-implements in low-level language and parallelizes (C, CUDA, etc.)

Error-prone

Porting: from scratch (Unproductive), from existing C version (Low Maintainability) ?

Performance portability is mostly a myth.

Objective of SANE

Reduce requirements for success in writing code

- Developer should only know about app: physics, error tolerance.
  - Program in array language, independent of HW

Increase short-term and long-term productivity

- High-level array code
  - Re-optimized for each HW target
    - New target = new machine model (Human-readable XML file)
    - Automatic algorithmic selection, locality and parallelization
  - Readable by other developers (long-term maintainability)
- Keep up with HW evolution

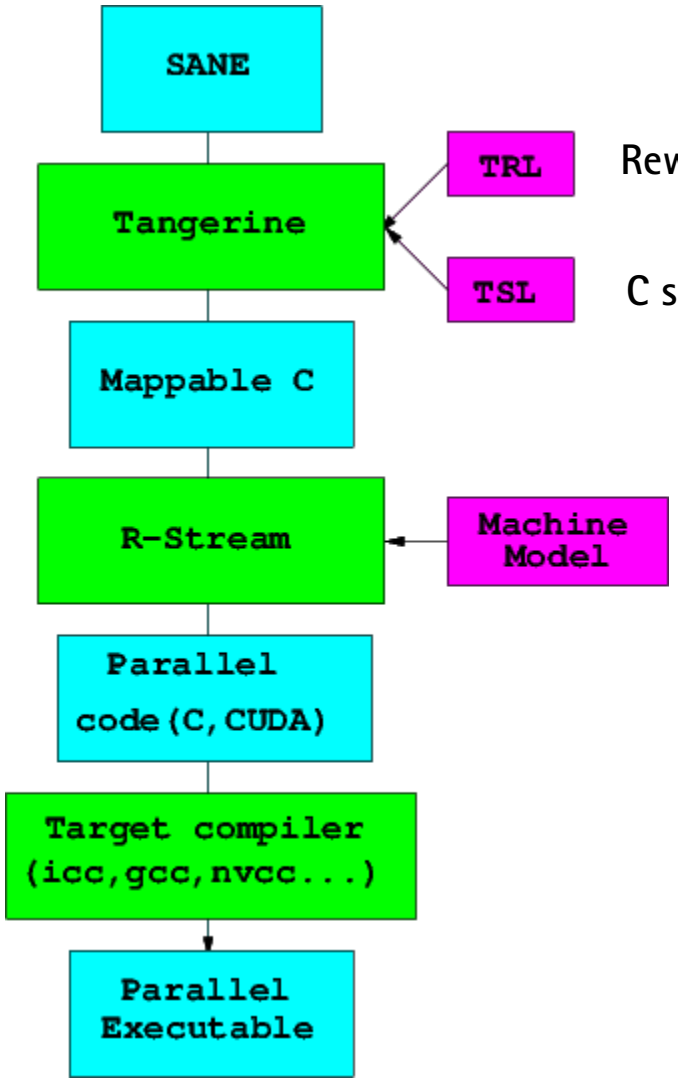
Validation: SAR backprojection loop

APPROACH

Automatic mapping from high-level language

Algorithmic rewriting/selection

Automatic parallelization & optimization



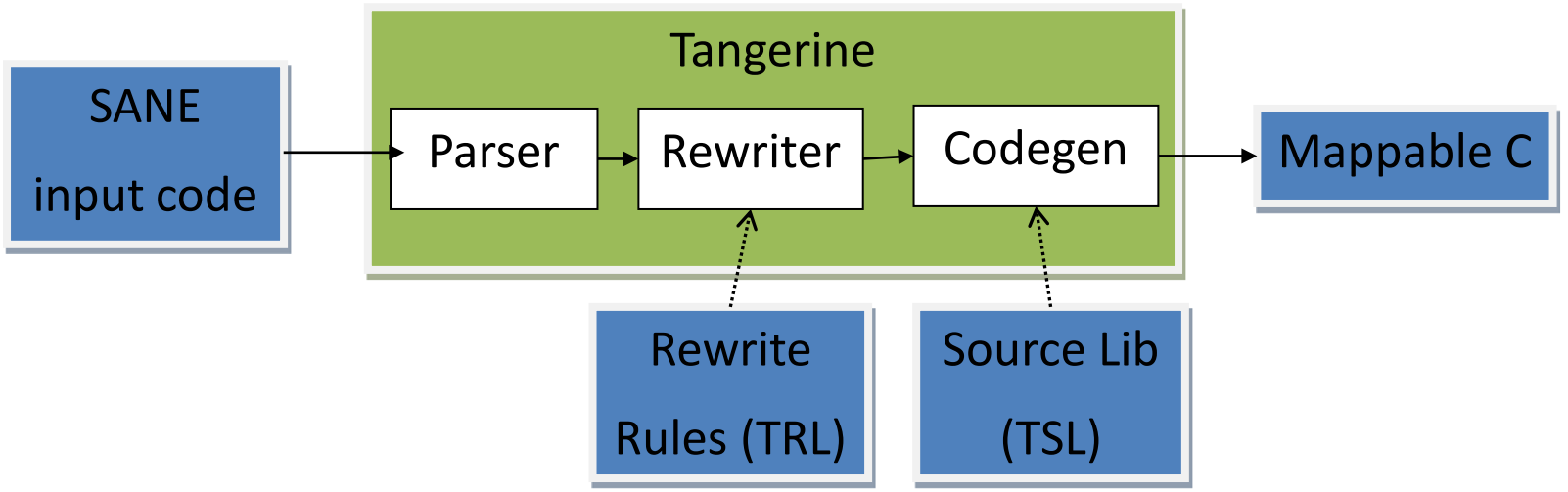
Tangerine rewriting array language front-end

Input program written in SANE source syntax

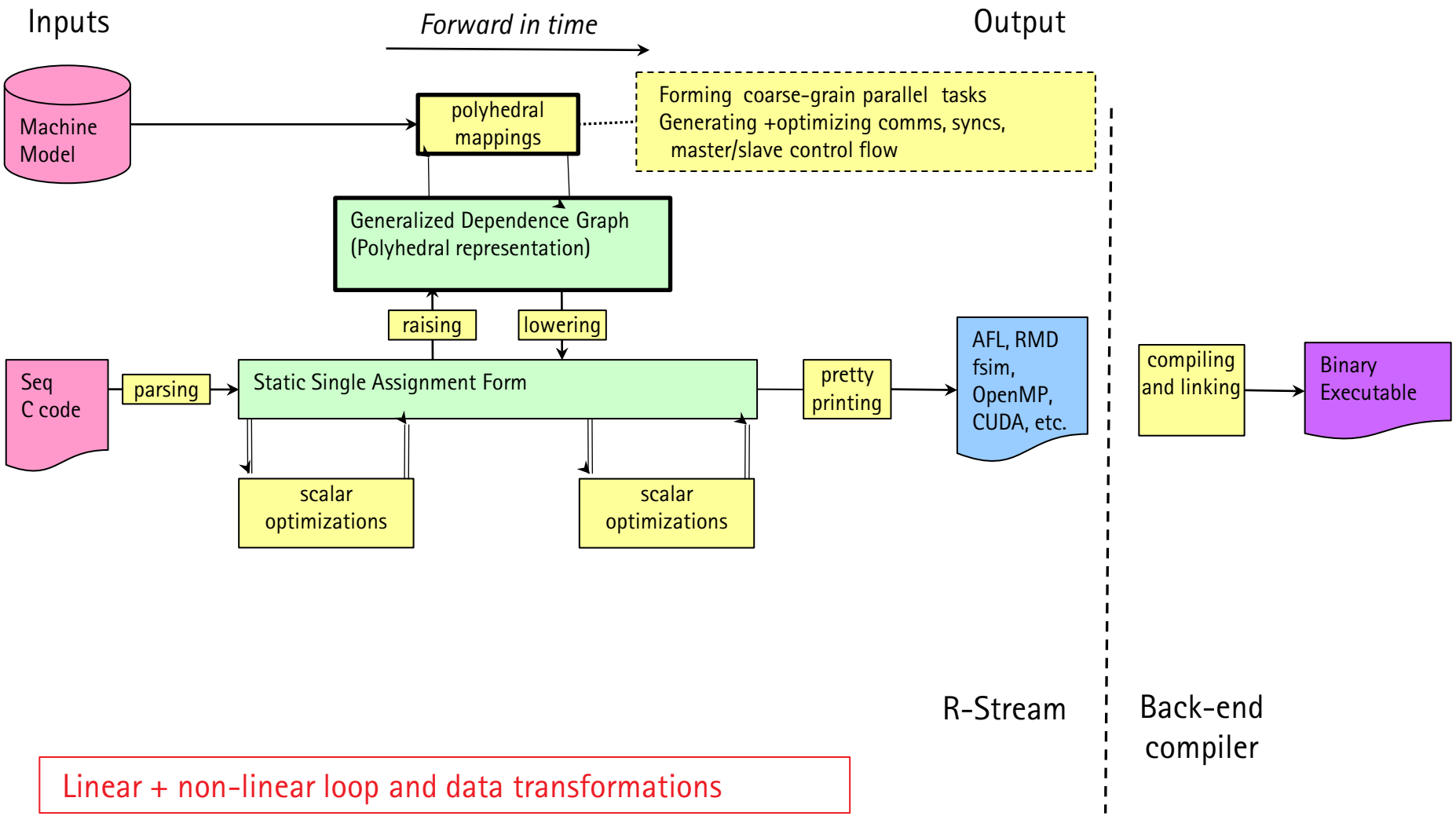
Source programs are optimized by ***rewriting*** according to specified ***rules.***

- *Using same language + rule keywords and variables*

Output is C, suitable for subsequent input to R-Stream



R-Stream auto-parallelizer



Linear + non-linear loop and data transformations

SANE language & front-end features

Succinct array operations (A = B*C, etc.)

Static typing, automatic type inference

- Performance – productivity tradeoff
- Specify type when specific precision is needed
- SAR: sin/cos argument must be double, result can be single

Complex, tuples support

Layout transformations

- AoS-to-SoA

SANE language - sample

```
function backproj(image:complexf(IMAGE_SIZE, IMAGE_SIZE),
                  phaseHistory:complexf(IMAGE_SIZE, IMAGE_SIZE),
                  platformPosition:single*3(IMAGE_SIZE),
                  xycoord:single*2(IMAGE_SIZE),
                  zcoord:single(IMAGE_SIZE, IMAGE_SIZE),
                  carrierF:double, dr:double, R0:double)

   pi4   = 4.0*PI;
   w     = pi4 * carrierF;
   ku2   = w / SPEED_OF_LIGHT;
   drInv = 1.0 / dr;
   for i = 1 : IMAGE_SIZE
      coord:single*3;
      ty:single*2;
      ty = xycoord(i);
      for j = 1 : IMAGE_SIZE
         tx:single*2;
         tx = xycoord(j);
         coord{1} = tx{1};
         coord{2} = ty{2};
         coord{3} = zcoord(i,j);
         accumMatchedFilter(image(i,j), phaseHistory, coord, platformPosition, ku2, R0, drInv);
      end
   end
```

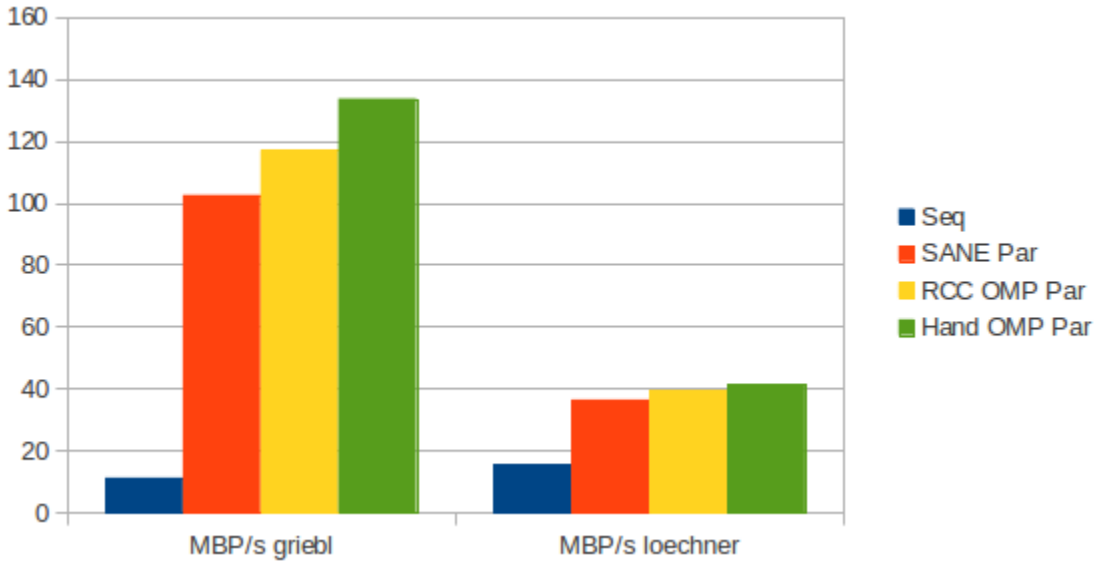
Mappable C code from SANE front-end (sample)

```
#pragma rstream map
void backproj(float  _real_image[1000][1000], float  _imag_image[1000][1000],
  float  _real_phaseHistory[1000][1000], float  _imag_phaseHistory[1000][1000]
  , float  _tpl1_platformPosition[1000], float  _tpl2_platformPosition[1000],
  float  _tpl3_platformPosition[1000], float  _tpl1_xycoord[1000], float
  _tpl2_xycoord[1000], float  zcoord[1000][1000], double *carrierF, double *dr
  , double *R0)
{
  double  pi4;
  double  w;
  double  ku2;
  double  drInv;
  int32_t  i;
  pi4 = 4.0 * 3.141592653589793;
  w = pi4 * *carrierF;
  ku2 = w / 3.0E8;
  drInv = 1.0 / *dr;
  for ( i = 1; i <= 1000; i += 1) {
    float  _tpl1_coord;
    float  _tpl2_coord;
    float  _tpl3_coord;
    float  _tpl1_ty;
    float  _tpl2_ty;
    int32_t  j;
    _tpl1_ty = _tpl1_xycoord[(int)(i)-1];
    _tpl2_ty = _tpl2_xycoord[(int)(i)-1];
    for ( j = 1; j <= 1000; j += 1) {
      float  _tpl1_tx;
      float  _tpl2_tx;
      _tpl1_tx = _tpl1_xycoord[(int)(j)-1];
      _tpl2_tx = _tpl2_xycoord[(int)(j)-1];
      _tpl1_coord = _tpl1_tx;
      _tpl2_coord = _tpl2_ty;
      _tpl3_coord = zcoord[(int)(i)-1][(int)(j)-1];
      accumMatchedFilter(& _real_image[(int)(i)-1][(int)(j)-1],  &
        _imag_image[(int)(i)-1][(int)(j)-1],  _real_phaseHistory ,
        _imag_phaseHistory ,  & _tpl1_coord ,  & _tpl2_coord ,  &
        _tpl3_coord ,  _tpl1_platformPosition ,  _tpl2_platformPosition ,
        _tpl3_platformPosition ,  & ku2 ,  &*R0 ,  & drInv);
    }
  }
}
```

Preliminary results

griebl: Intel Xeon E5520 @ 2.27GHz with four cores, 2x hyper-threading and 8MB L2 caches

loechner: Intel Core(TM) i5-2520M CPU @ 2.50GHz with two cores, 2x hyper-threading and 3MB L2 caches



Recent algorithmic optimizations for SAR [Benson12, Park12] not taken into account

Conclusion & future work

WRAPPING UP

SANE is an early prototype

Good enough for proof of concept

Language

- More complete (while loops, etc)

Rewriter

- Real rule inference driver
- Learning, connect with R-Stream's Auto-tuner (ARCC)

General robustness / flexibility

- Better integration of source library w/ type inference

Main advantages

Really high productivity

- One-step mapping.
- SAR: No human intervention in the SANE->executable path

Good for algorithm research

- Short cycle (seconds vs. weeks/days in best case)
- Rewriter automates some of the work

Performance portability

- Entire code is recompiled for each machine
- Explicit data movement (scratchpads, DMA), synchronization, thread creation is automatically generated and optimized

Control of precision using types

References

[Benson12] T. M. Benson, D. P. Campbell, and D. A. Cook, "Gigapixel spotlight synthetic aperture radar backprojection using clusters of gpus and CUDA," in Proceedings of the IEEE Radar Conference, 2012, pp. 853–858.

[ExascaleStudyGroup08] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, *"Exascale Computing Study: Technology Challenges in Achieving Exascale Systems."* DARPA IPTO report, 2008.

[Park2012] J. Park et al. Approximate Strength Reduction: A Method for Accuracy-Performance Trade-offs in Radar and Medical Imaging. Tech report, submitted to International Conference on Supercomputing 2012.