The Ct Virtual Machine: Enabling High Performance Domain Specific Languages and Libraries

Anwar Ghuloum

Software & Services Group, Developer Products Division

Agenda

- Ct Primer
- The Ct VM

What is Intel Ct Technology?

- Ct adds parallel collection objects & methods to C++
  - Library interface and is fully ANSI/ISO-compliant (works with ICC, VC++, GCC)
- Ct abstracts away architectural details
  - Vector ISA width / Core count / Memory model / Cache sizes
  - Focus on what to do, not how to do it
  - Sequential semantics
- Ct forward-scales software written today
  - Ct is designed to be dynamically retargetable to SSE, AVX, LRB, …
- Ct is safe, by default
  - …but with expert controls to override for performance

Programmers think sequential, not parallel

Collection Objects

Vec are the basic type of parallel collection object

- a handle to a value
- managed by the runtime
- flat, multidimensional, or irregularly nested
- created and manipulated exclusively via the API
  - determinism and isolation
  - overrides and control for extra performance

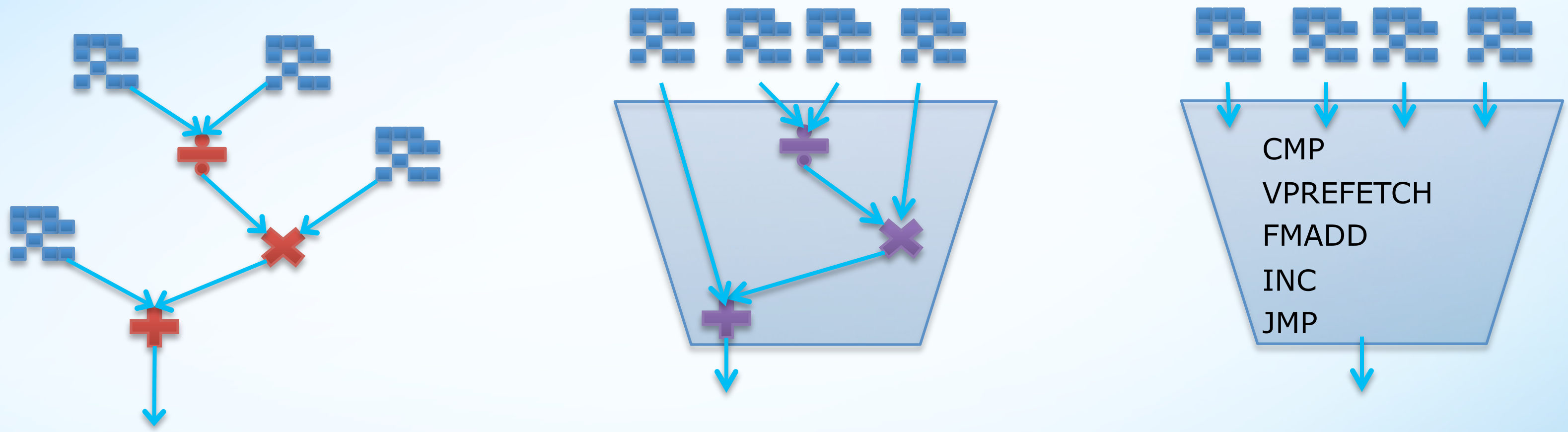
Provides Safety by Default

```
#include "ct.h"
using namespace Ct;
int main(int argc, char *argv[])
{
   // A Vec declaration must specify a base
   // type: Vec<basetype> aTypedVector;
            // For example, DoubleVec can refer to any
   // vector of doubles.
            Vec<F64> DoubleVec;

            // A regular 2 dimensional vector:
   Vec2D<I8> 2DMatrix ("{{0,1,2}, {3,4,5}, {6,7,8}}");

            //An irregularly nested vector:
            VecNested<I32> IrregularVector ("{{0,1,2},
   {3,4}, {5,6,7,8}}");

            return 0;
}
```

Parallel Operations on Ct Collections

The Ct Runtime Automates This Transformation

Vector Processing | Kernel Processing | Native/Intrinsic Coding



CMP
VPREFETCH
FMADD
INC
JMP

```
Vec<F32> A, B, C, D;
A += B/C * D;
```

```
Elt<F32> kernel(Elt<F32> a, b, c,
 d) {
  return a + (b/c)*d;
}
…
Vec<F32> A, B, C, D;
A = map(kernel)(A, B, C, D);
```

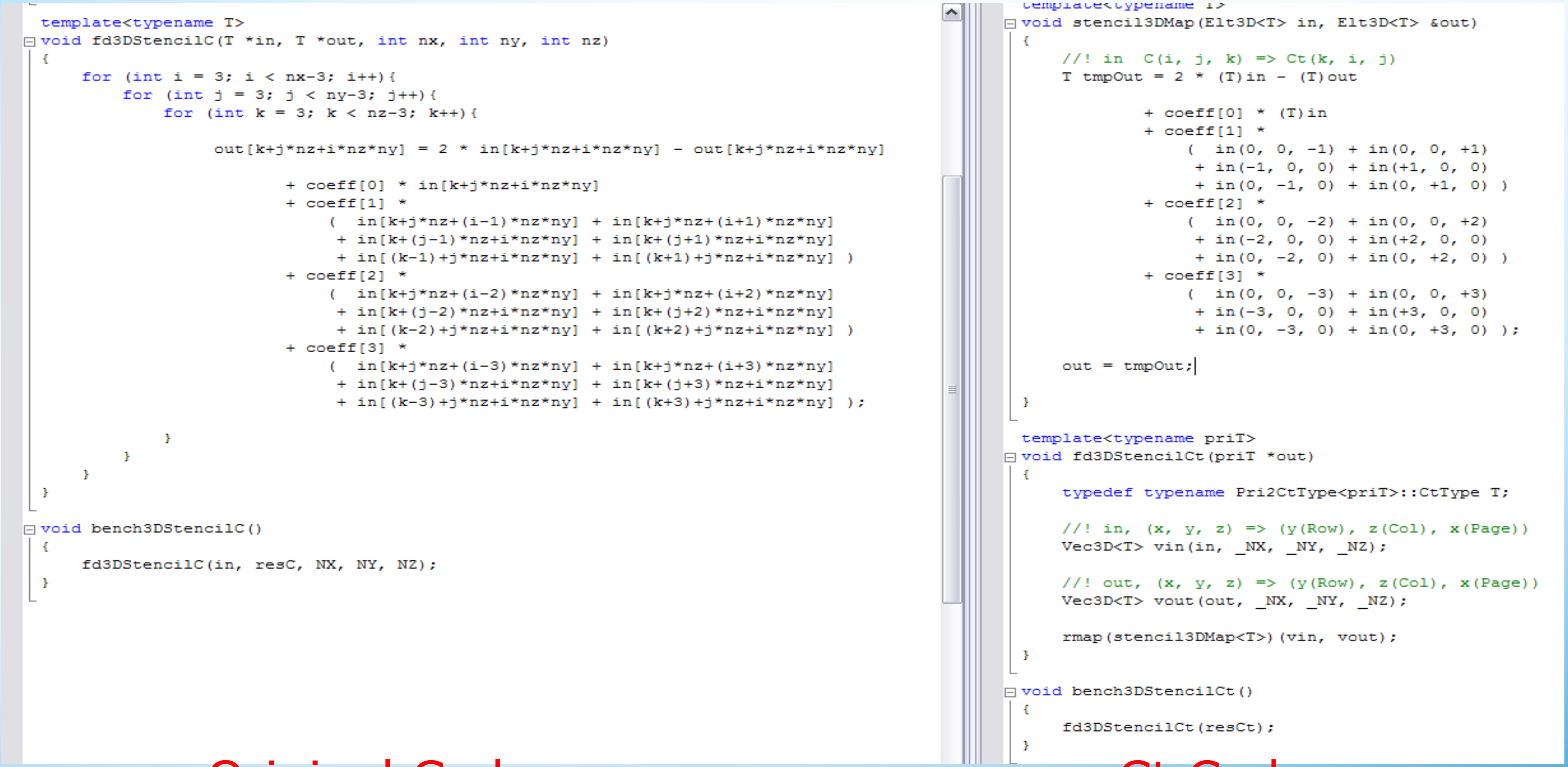
```
NVec<F32>native(NVec<F32> …) {
  __asm__ {
        …
  };
}
…
Vec<F32> A, B, C, D;
A = map(native)(A, B, C, D);
```

Or Programmers Can Choose Desired Level of Abstraction

Linear algebra, global data movement/communication

Embarrassingly parallel, shaders, image processing

3D order-6 stencil



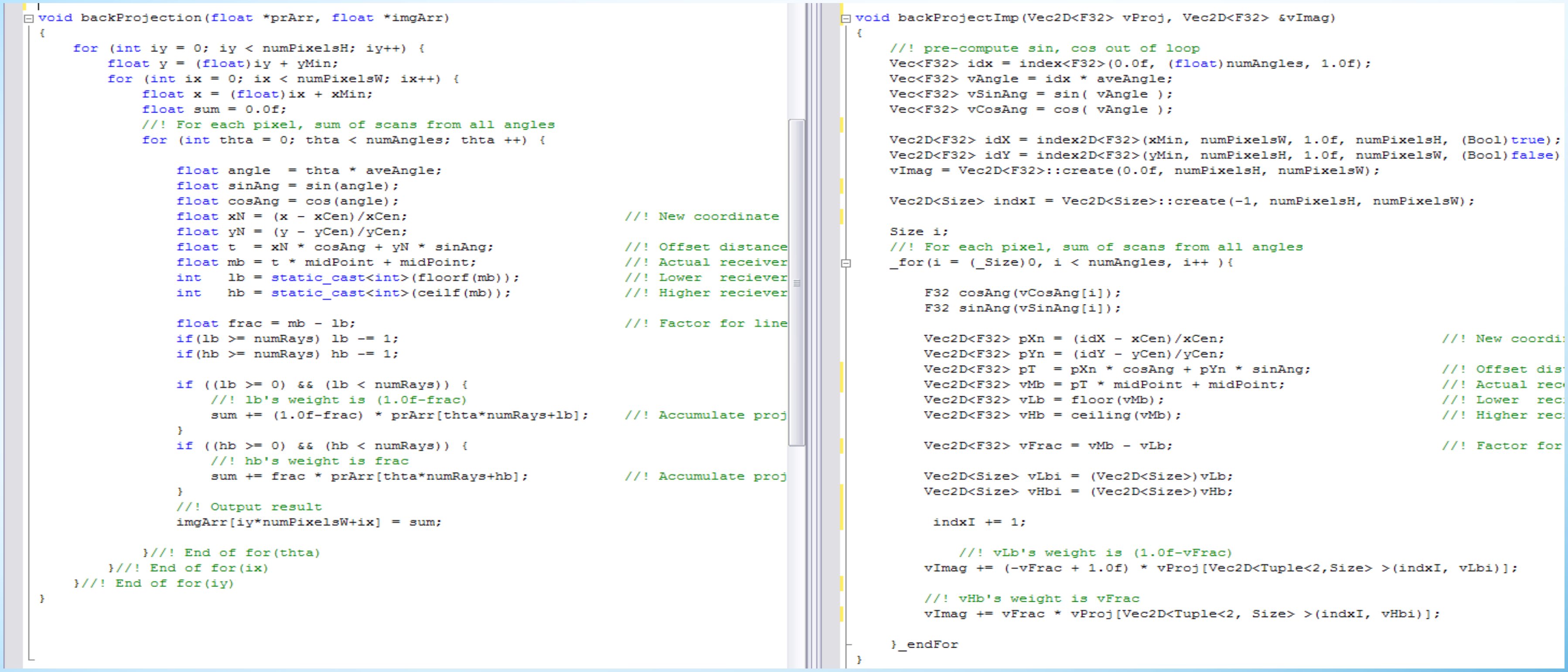
```cpp
template<typename T>
void fd3DStencilC(T *in, T *out, int nx, int ny, int nz)
{
    for (int i = 3; i < nx-3; i++){
        for (int j = 3; j < ny-3; j++){
            for (int k = 3; k < nz-3; k++){

                out[k+j*nz+i*nz*ny] = 2 * in[k+j*nz+i*nz*ny] - out[k+j*nz+i*nz*ny]

                        + coeff[0] * in[k+j*nz+i*nz*ny]
                        + coeff[1] *
                            (  in[k+j*nz+(i-1)*nz*ny] + in[k+j*nz+(i+1)*nz*ny]
                             + in[k+(j-1)*nz+i*nz*ny] + in[k+(j+1)*nz+i*nz*ny]
                             + in[(k-1)+j*nz+i*nz*ny] + in[(k+1)+j*nz+i*nz*ny] )
                        + coeff[2] *
                            (  in[k+j*nz+(i-2)*nz*ny] + in[k+j*nz+(i+2)*nz*ny]
                             + in[k+(j-2)*nz+i*nz*ny] + in[k+(j+2)*nz+i*nz*ny]
                             + in[(k-2)+j*nz+i*nz*ny] + in[(k+2)+j*nz+i*nz*ny] )
                        + coeff[3] *
                            (  in[k+j*nz+(i-3)*nz*ny] + in[k+j*nz+(i+3)*nz*ny]
                             + in[k+(j-3)*nz+i*nz*ny] + in[k+(j+3)*nz+i*nz*ny]
                             + in[(k-3)+j*nz+i*nz*ny] + in[(k+3)+j*nz+i*nz*ny] );

            }
        }
    }
}

void bench3DStencilC()
{
    fd3DStencilC(in, resC, NX, NY, NZ);
}
```

Original Code

```cpp
template<typename T>
void stencil3DMap(Elt3D<T> in, Elt3D<T> &out)
{
    //! in  C(i, j, k) => Ct(k, i, j)
    T tmpOut = 2 * (T)in - (T)out

            + coeff[0] * (T)in
            + coeff[1] *
                (  in(0, 0, -1) + in(0, 0, +1)
                 + in(-1, 0, 0) + in(+1, 0, 0)
                 + in(0, -1, 0) + in(0, +1, 0) )
            + coeff[2] *
                (  in(0, 0, -2) + in(0, 0, +2)
                 + in(-2, 0, 0) + in(+2, 0, 0)
                 + in(0, -2, 0) + in(0, +2, 0) )
            + coeff[3] *
                (  in(0, 0, -3) + in(0, 0, +3)
                 + in(-3, 0, 0) + in(+3, 0, 0)
                 + in(0, -3, 0) + in(0, +3, 0) );

    out = tmpOut;

}

template<typename priT>
void fd3DStencilCt(priT *out)
{
    typedef typename Pri2CtType<priT>::CtType T;

    //! in, (x, y, z) => (y(Row), z(Col), x(Page))
    Vec3D<T> vin(in, _NX, _NY, _NZ);

    //! out, (x, y, z) => (y(Row), z(Col), x(Page))
    Vec3D<T> vout(out, _NX, _NY, _NZ);

    rmap(stencil3DMap<T>)(vin, vout);
}

void bench3DStencilCt()
{
    fd3DStencilCt(resCt);
}
```

Ct Code

Back Projection



```
void backProjection(float *prArr, float *imgArr)
{
    for (int iy = 0; iy < numPixelsH; iy++) {
        float y = (float)iy + yMin;
        for (int ix = 0; ix < numPixelsW; ix++) {
            float x = (float)ix + xMin;
            float sum = 0.0f;
            //! For each pixel, sum of scans from all angles
            for (int thta = 0; thta < numAngles; thta ++) {

                float angle  = thta * aveAngle;
                float sinAng = sin(angle);
                float cosAng = cos(angle);
                float xN = (x - xCen)/xCen;                    //! New coordinate
                float yN = (y - yCen)/yCen;
                float t  = xN * cosAng + yN * sinAng;          //! Offset distance
                float mb = t * midPoint + midPoint;            //! Actual receiver
                int   lb = static_cast<int>(floorf(mb));       //! Lower  reciever
                int   hb = static_cast<int>(ceilf(mb));        //! Higher reciever

                float frac = mb - lb;                          //! Factor for line
                if(lb >= numRays) lb -= 1;
                if(hb >= numRays) hb -= 1;

                if ((lb >= 0) && (lb < numRays)) {
                    //! lb's weight is (1.0f-frac)
                    sum += (1.0f-frac) * prArr[thta*numRays+lb];    //! Accumulate proj
                }
                if ((hb >= 0) && (hb < numRays)) {
                    //! hb's weight is frac
                    sum += frac * prArr[thta*numRays+hb];          //! Accumulate proj
                }
                //! Output result
                imgArr[iy*numPixelsW+ix] = sum;

            }//! End of for(thta)
        }//! End of for(ix)
    }//! End of for(iy)
}
```

Original Code

```
void backProjectImp(Vec2D<F32> vProj, Vec2D<F32> &vImag)
{
    //! pre-compute sin, cos out of loop
    Vec<F32> idx = index<F32>(0.0f, (float)numAngles, 1.0f);
    Vec<F32> vAngle = idx * aveAngle;
    Vec<F32> vSinAng = sin( vAngle );
    Vec<F32> vCosAng = cos( vAngle );

    Vec2D<F32> idX = index2D<F32>(xMin, numPixelsW, 1.0f, numPixelsH, (Bool)true);
    Vec2D<F32> idY = index2D<F32>(yMin, numPixelsH, 1.0f, numPixelsW, (Bool)false)
    vImag = Vec2D<F32>::create(0.0f, numPixelsH, numPixelsW);

    Vec2D<Size> indxI = Vec2D<Size>::create(-1, numPixelsH, numPixelsW);

    Size i;
    //! For each pixel, sum of scans from all angles
    _for(i = (_Size)0, i < numAngles, i++ ){

        F32 cosAng(vCosAng[i]);
        F32 sinAng(vSinAng[i]);

        Vec2D<F32> pXn = (idX - xCen)/xCen;                            //! New coordi
        Vec2D<F32> pYn = (idY - yCen)/yCen;
        Vec2D<F32> pT  = pXn * cosAng + pYn * sinAng;                  //! Offset dis
        Vec2D<F32> vMb = pT * midPoint + midPoint;                     //! Actual rec
        Vec2D<F32> vLb = floor(vMb);                                   //! Lower  rec
        Vec2D<F32> vHb = ceiling(vMb);                                 //! Higher rec

        Vec2D<F32> vFrac = vMb - vLb;                                  //! Factor for

        Vec2D<Size> vLbi = (Vec2D<Size>)vLb;
        Vec2D<Size> vHbi = (Vec2D<Size>)vHb;

         indxI += 1;

            //! vLb's weight is (1.0f-vFrac)
        vImag += (-vFrac + 1.0f) * vProj[Vec2D<Tuple<2,Size> >(indxI, vLbi)];

        //! vHb's weight is vFrac
        vImag += vFrac * vProj[Vec2D<Tuple<2, Size> >(indxI, vHbi)];

    }_endFor
}
```

Ct Code

How Does it Really Work?

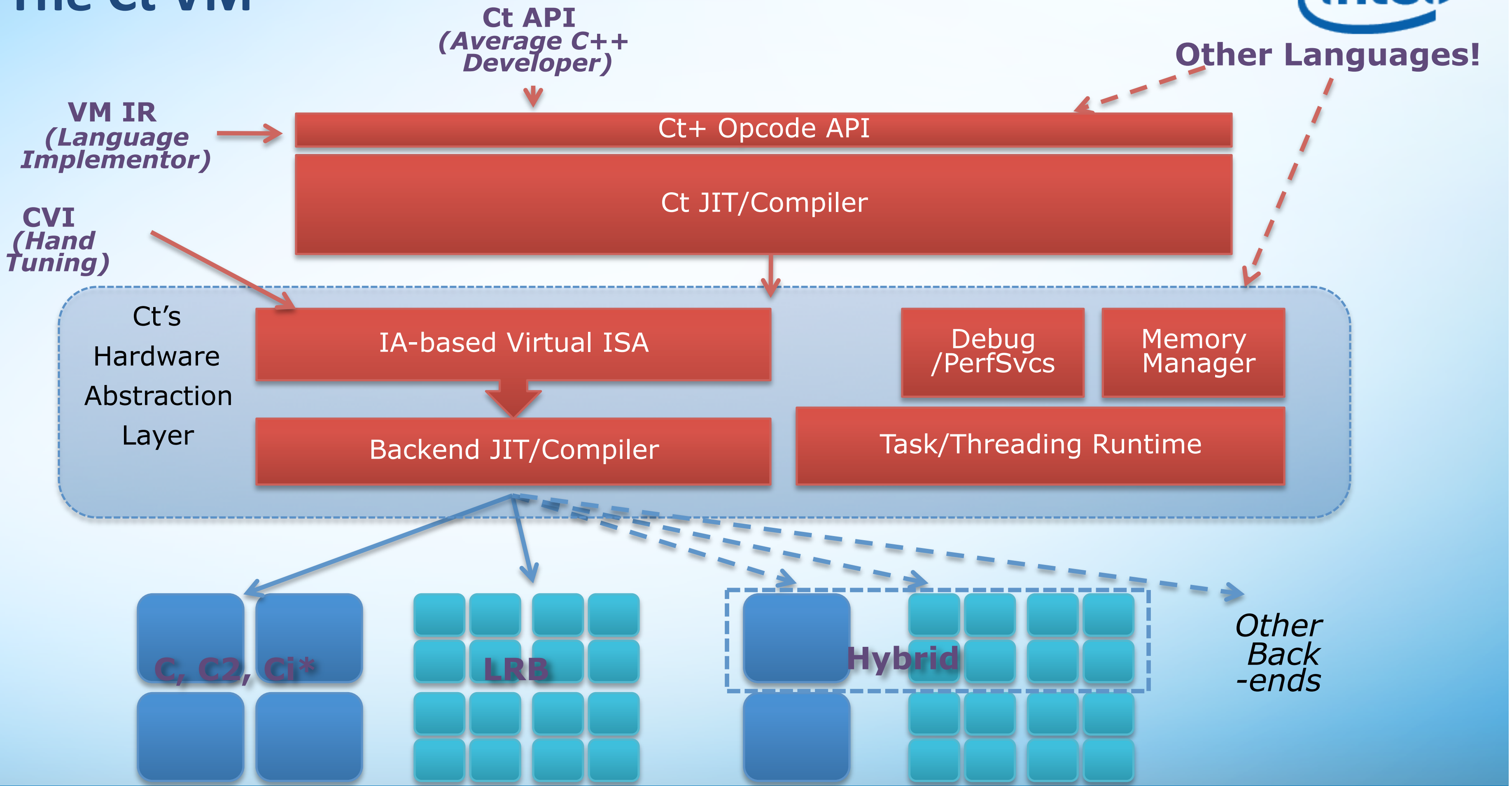
Ct is really a high-level APIs…

…that streams opcodes to an optimizing virtual machine

The source (front-end) can be anything:

- A new language
- A bytecode parser
  - Experiments with Python, HLSL
- An application-specific library
- A compiler front-end

The Ct VM



Ct API
(Average C++ Developer)

Other Languages!

VM IR
(Language Implementor)

CVI
(Hand Tuning)

Ct+ Opcode API

Ct JIT/Compiler

Ct's Hardware Abstraction Layer

IA-based Virtual ISA

Backend JIT/Compiler

Debug /PerfSvcs

Memory Manager

Task/Threading Runtime

C, C2, Ci*

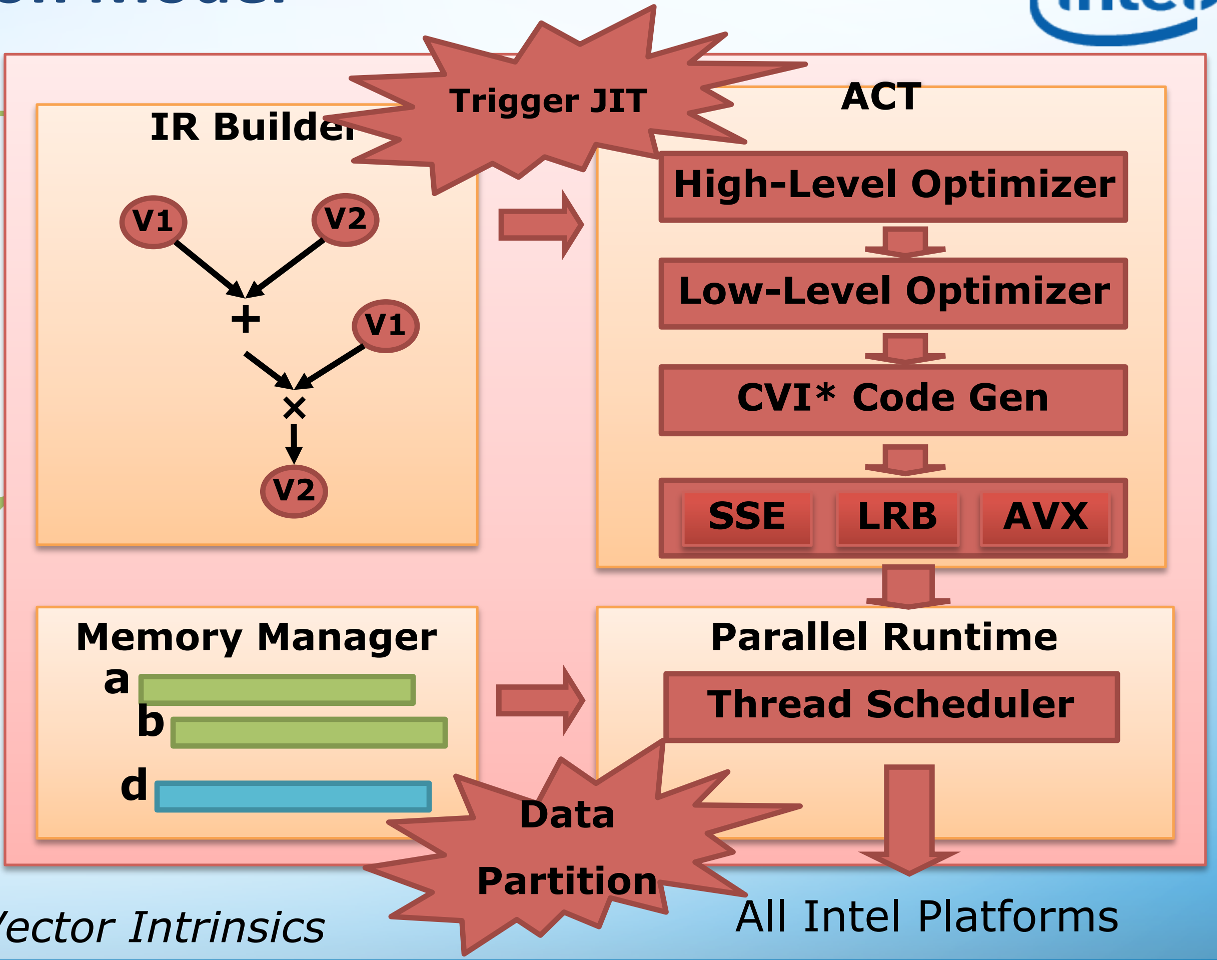
LRB

Hybrid

Other Back -ends

Runtime Evaluation Model

```
float src1[], src2[], dest[];

Vec<F32>a(src1,N), b(src2,N);
rcall(foo)(a, b)
…
foo(Vec<F32> a, Vec<F32> b) {
  Vec<F32> c = a + b;
  Vec<F32> d = c * a;
  return;
}
```



Trigger JIT

IR Builder

ACT

High-Level Optimizer

Low-Level Optimizer

CVI* Code Gen

SSE LRB AVX

Memory Manager

Parallel Runtime

Thread Scheduler

Data Partition

All Intel Platforms

Ct Dynamic Engine

* *CVI = Converged Vector Intrinsics*

Why Does this Matter for C/C++ Developers?

It's not just a single kernel…

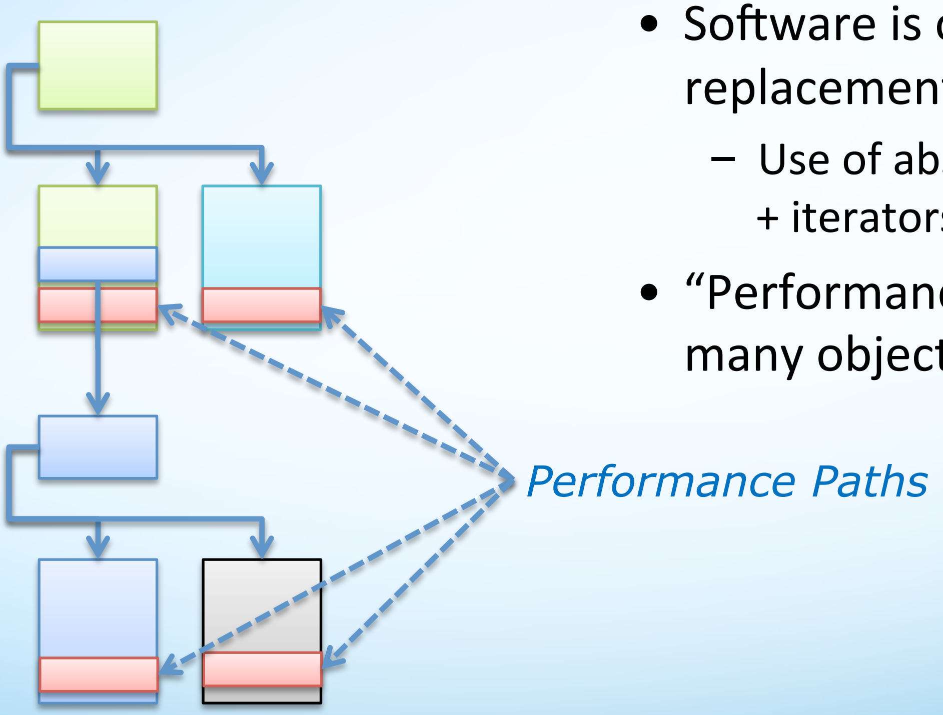
- Productivity craters when many kernels have to be tuned
  - Focusing energy on 1 algorithm makes sense, if it is the dominant algorithm

…in one place

- Widely used libraries often give up performance for well designed generic interfaces

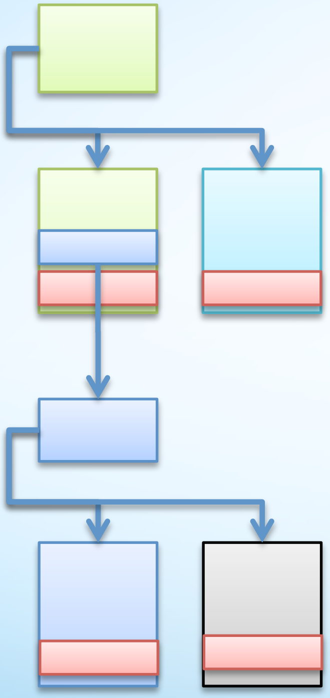
→ Inherently spreads compute across methods

Performance Without De-architecting Software

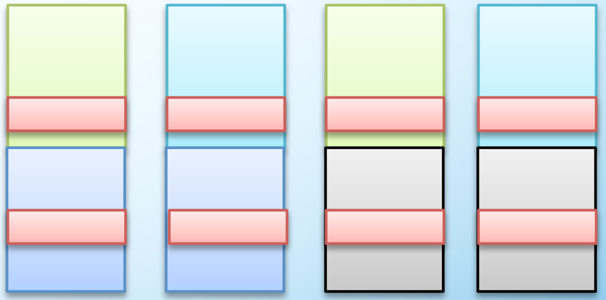


- Software is often architected for reuse, replacement, extension:
  - Use of abstract classes, virtual function calls, C++ iterators, indirection is the norm…
- "Performance paths" are often spread across many objects and files

Performance Without De-architecting Software

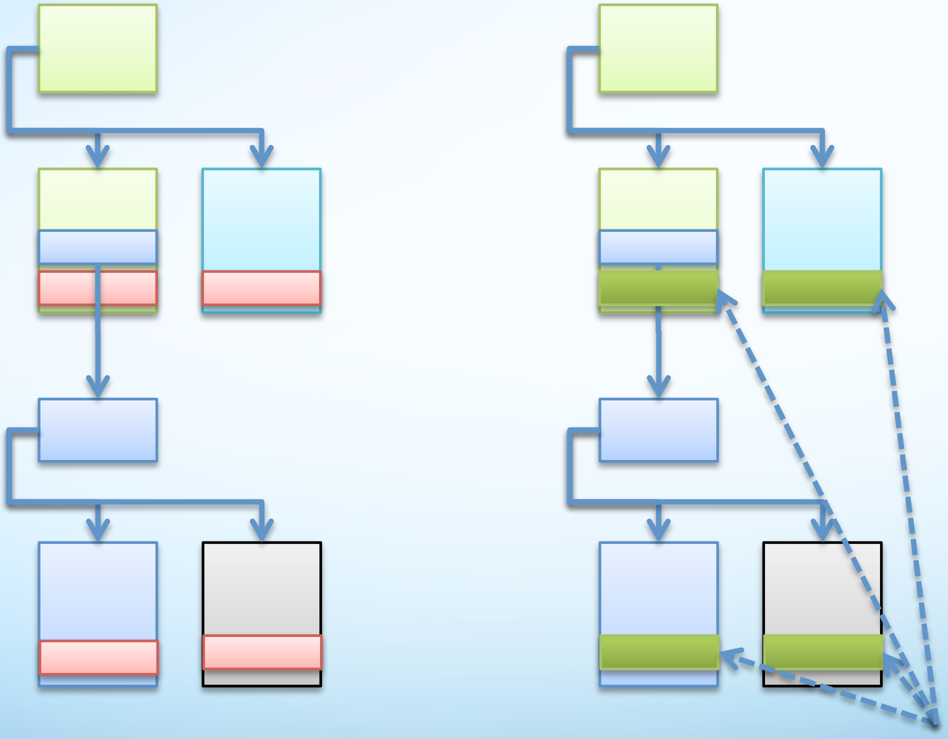


- Performance tools typically want to see *everything!*
- You look at all possible/likely paths
  - Brittle
  - Difficult to maintain
  - Difficult to extend
  - Difficult to program



De-architecting for performance

Performance Without De-architecting Software



Ct in your Classes

- Combine good software practices and performance with Ct:
  - Pepper your models/ classes with Ct
  - Ct's VM takes care of dynamically gathering the performance paths

Concluding Remarks

- Managed/dynamic runtimes are no longer synonymous with poor performance

  →You don't have to sacrifice productivity for performance

- The pace of credible language emergence will be sustained
  - A new language every 18 months
  - It may even *grow*, driven by architectural/application innovation and specialization

Fini

Software & Services Group, Developer Products Division