

Stillwater OpenKL Run-time

Separation of Concerns to improve productivity,
portability, and performance of Computational
Science Applications

Would you like to write this

```
matrixMul_kernel.cu                                     Page 1

/*
 * Copyright 1993-2010 NVIDIA Corporation. All rights reserved.
 *
 * Please refer to the NVIDIA end user license agreement (EULA) associated
 * with this source code for terms and conditions that govern your use of
 * this software. Any use, reproduction, disclosure, or distribution of
 * this software and related documentation outside the terms of the EULA
 * is strictly prohibited.
 *
 */

/* Matrix multiplication: C = A * B.
 * Device code.
 */

#ifdef _MATRIXMUL_KERNEL_H_
#define _MATRIXMUL_KERNEL_H_

#include <stdio.h>

#define CHECK_BANK_CONFLICTS 0
#if CHECK_BANK_CONFLICTS
#define AS(1, 3) cutilBankChecker((float*) &As[0][0]), (BLOCK_SIZE * 1 + 3))
#define BS(1, 3) cutilBankChecker((float*) &Bs[0][0]), (BLOCK_SIZE * 1 + 3))
#else
#define AS(1, 3) As[1][3]
#define BS(1, 3) Bs[1][3]
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//! Matrix multiplication on the device: C = A * B
//! wA is A's width and wB is B's width
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template <int BLOCK_SIZE> __global__ void
matrixMul(float* C, float* A, float* B, int wA, int wB)
{
    // Block index
    int bx = blockIdx.x;
    int by = blockIdx.y;

    // Thread index
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Index of the first sub-matrix of A processed by the block
    int aBegin = wA * BLOCK_SIZE * by;

    // Index of the last sub-matrix of A processed by the block
    int aEnd = aBegin + wA - 1;

    // Step size used to iterate through the sub-matrices of A
    int aStep = BLOCK_SIZE;

    // Index of the first sub-matrix of B processed by the block
    int bBegin = BLOCK_SIZE * bx;

    // Step size used to iterate through the sub-matrices of B
    int bStep = BLOCK_SIZE * wB;

    // Csub is used to store the element of the block sub-matrix
    // that is computed by the thread
    float Csub = 0;

    // Loop over all the sub-matrices of A and B
```

```
matrixMul_kernel.cu                                     Page 2

    // required to compute the block sub-matrix
    for (int a = aBegin, b = bBegin;
         a <= aEnd;
         a += aStep, b += bStep) {

        // Declaration of the shared memory array As used to
        // store the sub-matrix of A
        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

        // Declaration of the shared memory array Bs used to
        // store the sub-matrix of B
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        // Load the matrices from device memory
        // to shared memory; each thread loads
        // one element of each matrix
        AS(ty, tx) = A[a + wA * ty + tx];
        BS(ty, tx) = B[b + wB * ty + tx];

        // Synchronize to make sure the matrices are loaded
        __syncthreads();

        // Multiply the two matrices together;
        // each thread computes one element
        // of the block sub-matrix
#pragma unroll
        for (int k = 0; k < BLOCK_SIZE; ++k)
            Csub += AS(ty, k) * BS(k, tx);

        // Synchronize to make sure that the preceding
        // computation is done before loading two new
        // sub-matrices of A and B in the next iteration
        __syncthreads();

        // Write the block sub-matrix to device memory;
        // each thread writes one element
        int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
        C[c + wB * ty + tx] = Csub;
    }

#endif // #ifdef _MATRIXMUL_KERNEL_H_
```

If you could write this?

$C = A * B;$

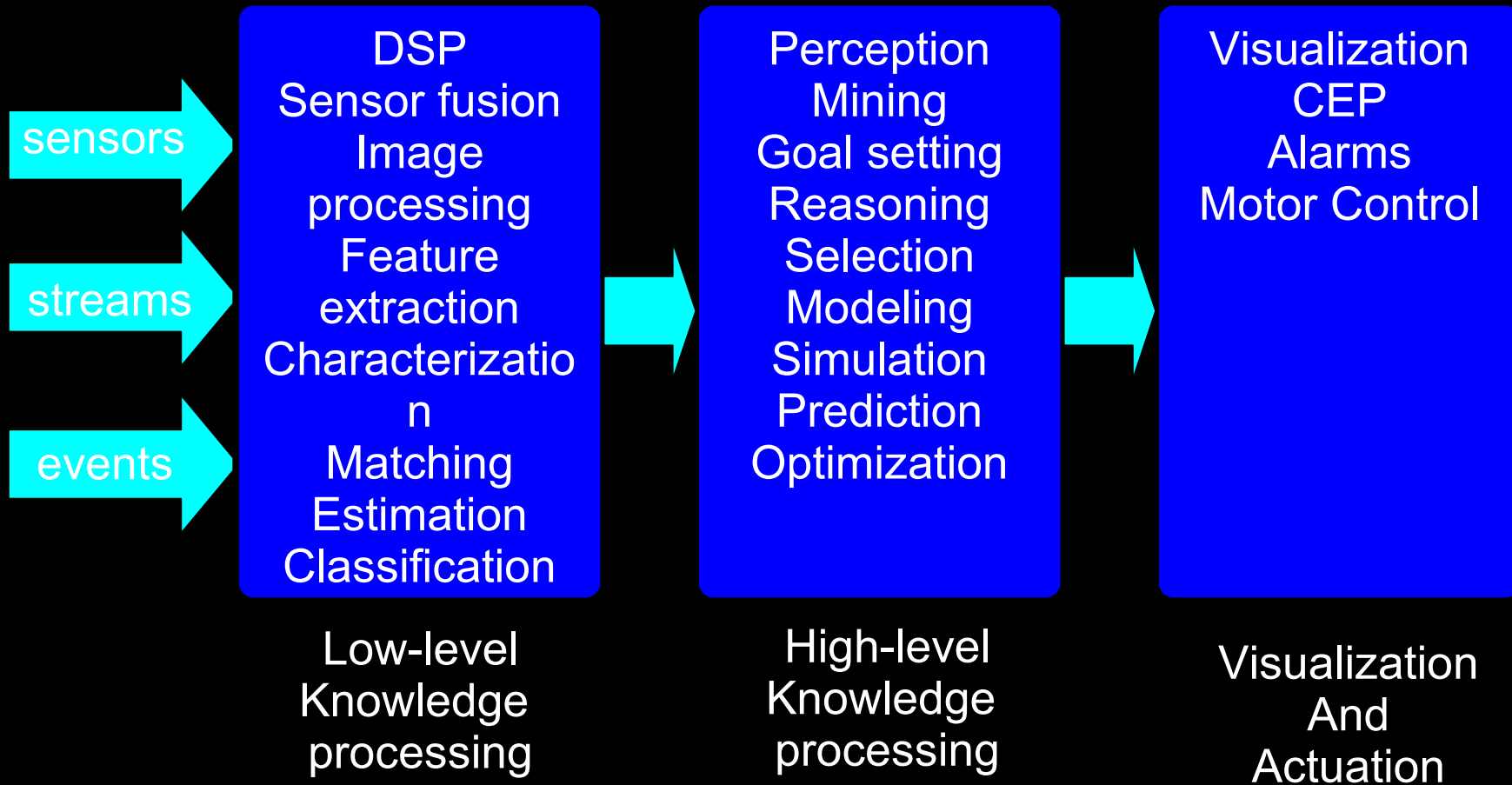
Goals for OpenKL

- Create high-productivity run-time for computational science and engineering
- Optimize performance for underlying platform
- Enable applications to move without penalty



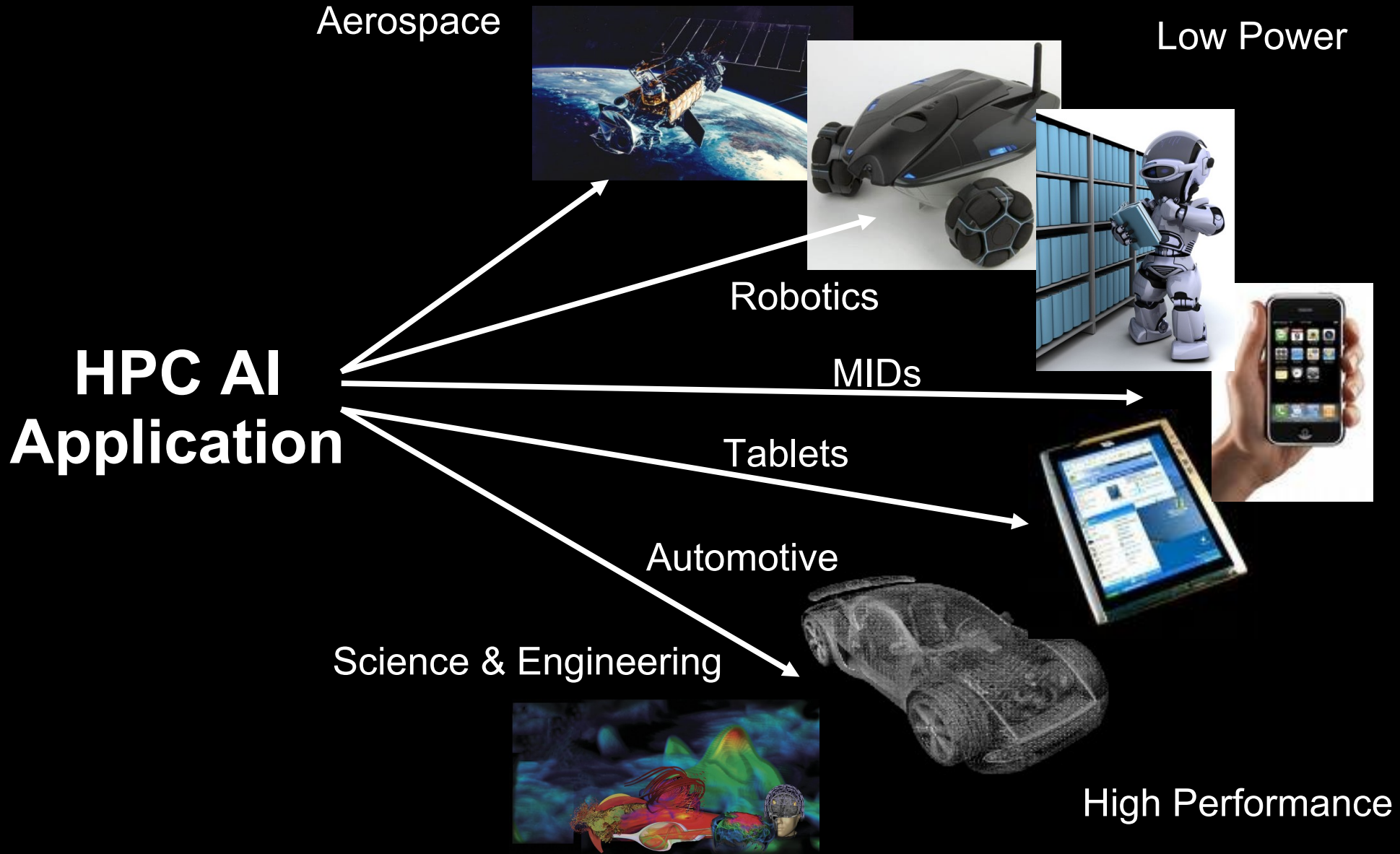
Using HPC to create intelligence

The Knowledge Processing Pipeline



Stillwater OpenKL is tailored to knowledge processing workloads

Problem: Deliver Intelligence to proliferating hardware environments



Cloud Computing....



A Very Successful Solution for Graphics Applications

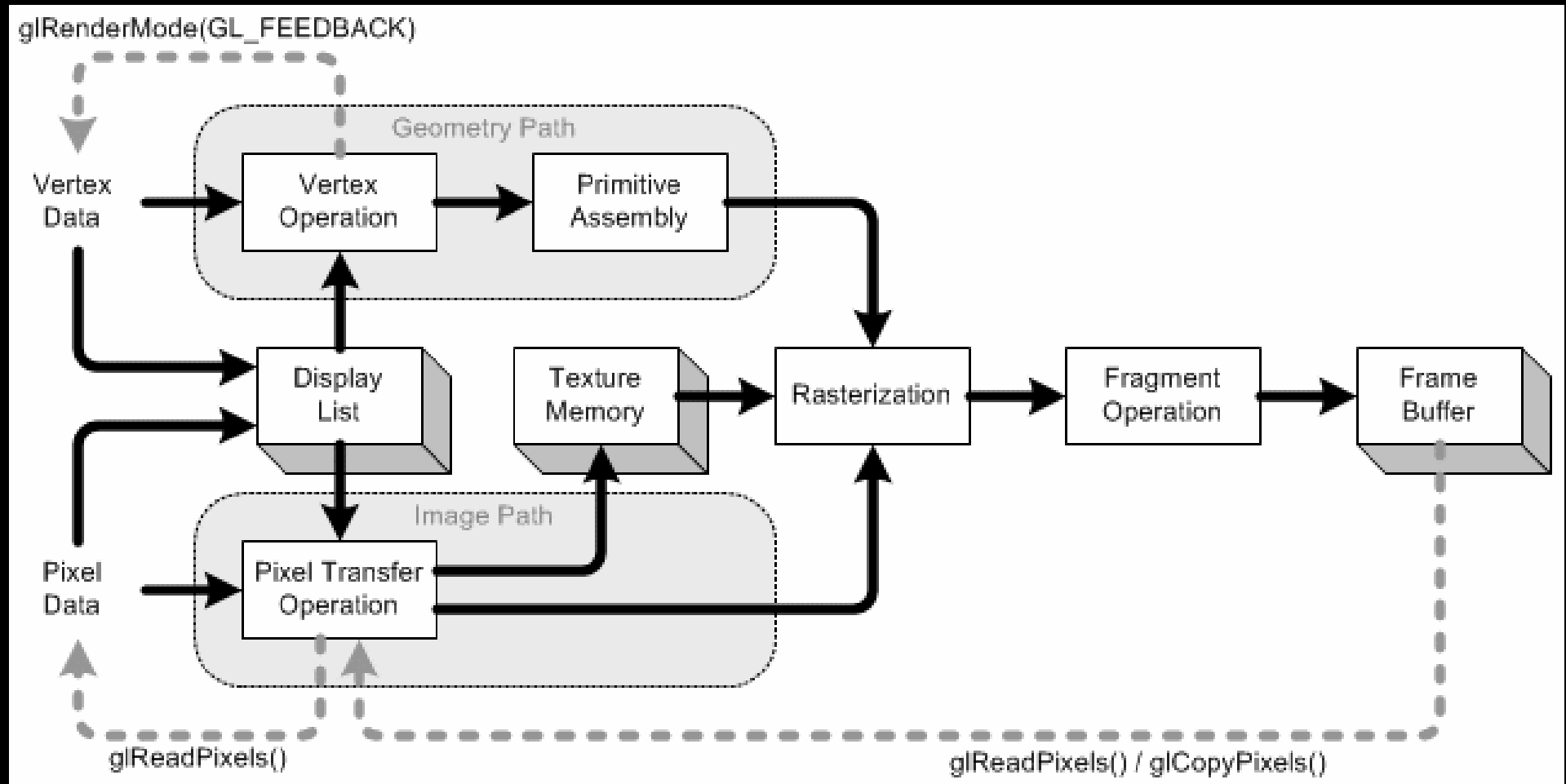
Graphics
Application
n



Separation of Concerns

- To render a synthetic image you need
 - A rasterizer to generate colored pixels
 - A lighting model to make color visible
 - A transformation model to compose a scene
 - Geometry to place in the scene
- In the OpenGL Virtual Machine
 - Transform and Lighting models are abstract
 - Rasterization algorithm is implicit
 - Data structures are managed

OpenGL Virtual Machine for graphics applications



Graphics applications are abstracted away from hardware

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("blender");
    glutDisplayFunc(display);
    glutVisibilityFunc(visible);

    glNewList(1, GL_COMPILE); /* create ico display list */
    glutSolidIcosahedron();
    glEndList();

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
    glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
    glLightfv(GL_LIGHT2, GL_DIFFUSE, light2_diffuse);
    glLightfv(GL_LIGHT2, GL_POSITION, light2_position);

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_LINE_SMOOTH);
    glLineWidth(2.0);

    glMatrixMode(GL_PROJECTION);
    gluPerspective( /* field of view in degree */ 40.0,
    /* aspect ratio */ 1.0, /* Z near */ 1.0, /* Z far */ 10.0);
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(0.0, 0.0, 5.0, /* eye is at (0,0,5) */
    0.0, 0.0, 0.0, /* center is at (0,0,0) */
    0.0, 1.0, 0.); /* up is in positive Y direction */
    glTranslatef(0.0, 0.6, -1.0);

    glutMainLoop();
    return 0;
}
```

Setup rendering pipeline

Create geometry
and handle

Setup lighting model

Setup raster model

Setup initial transformation

Interactive rendering loop

Graphics Applications Performance

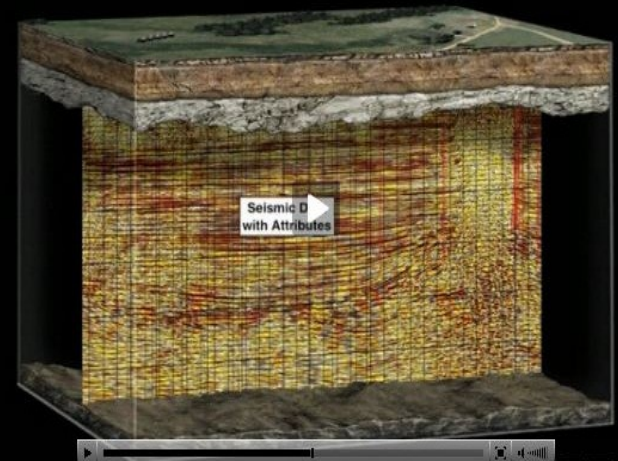
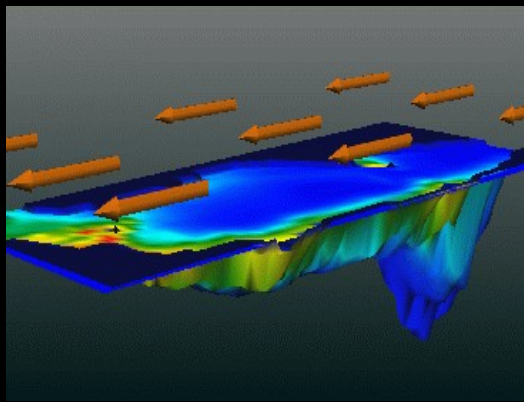
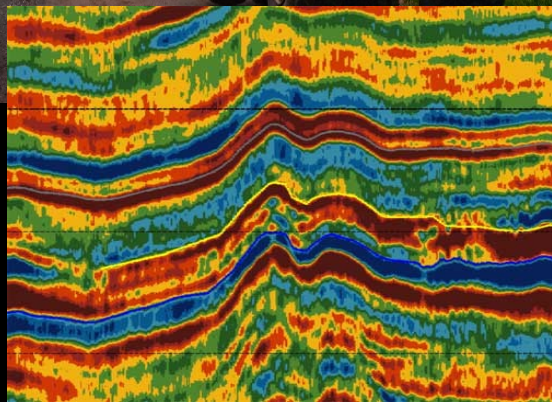
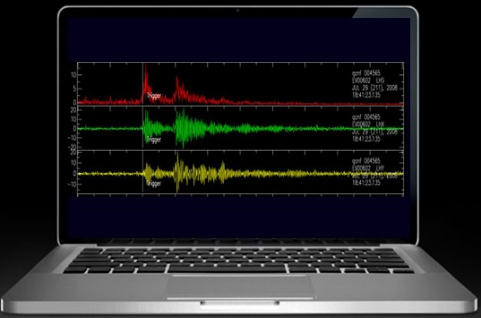
- Bottlenecks shift between transformation and rasterization
 - Transform one million one-pixel triangles
 - Draw a million pixel triangle
- The data flow dynamics differ from system to system
 - Unproductive to program explicitly
 - The OpenGL Virtual Machine articulates the separation of concerns
 - The OpenGL run-time dynamically maps these abstractions onto underlying hardware

Result

- Graphics Applications that write to the OpenGL virtual machine
 - Can leverage higher expressiveness which leads to higher software development and maintenance productivity
 - Are decoupled from underlying hardware
 - Can take advantage of hardware acceleration and cluster parallelization without having to be rewritten or even recompiled

Stillwater OpenKL provides these same benefits to computational science and engineering applications

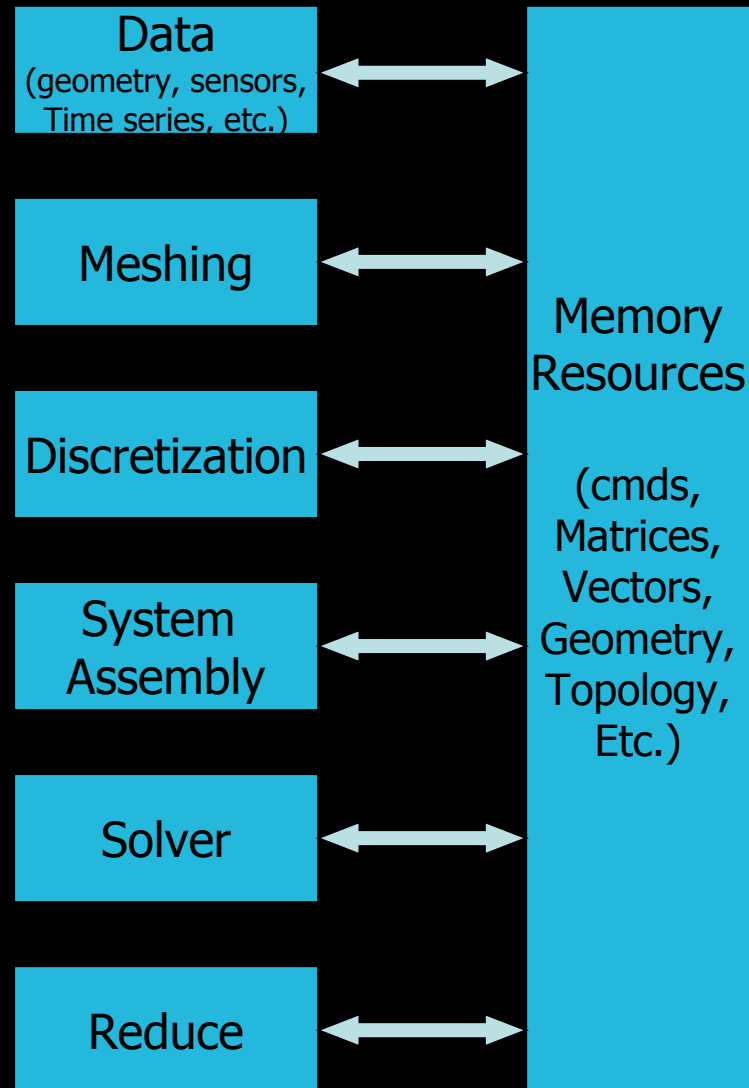
Computational Science: Seismic Surveying



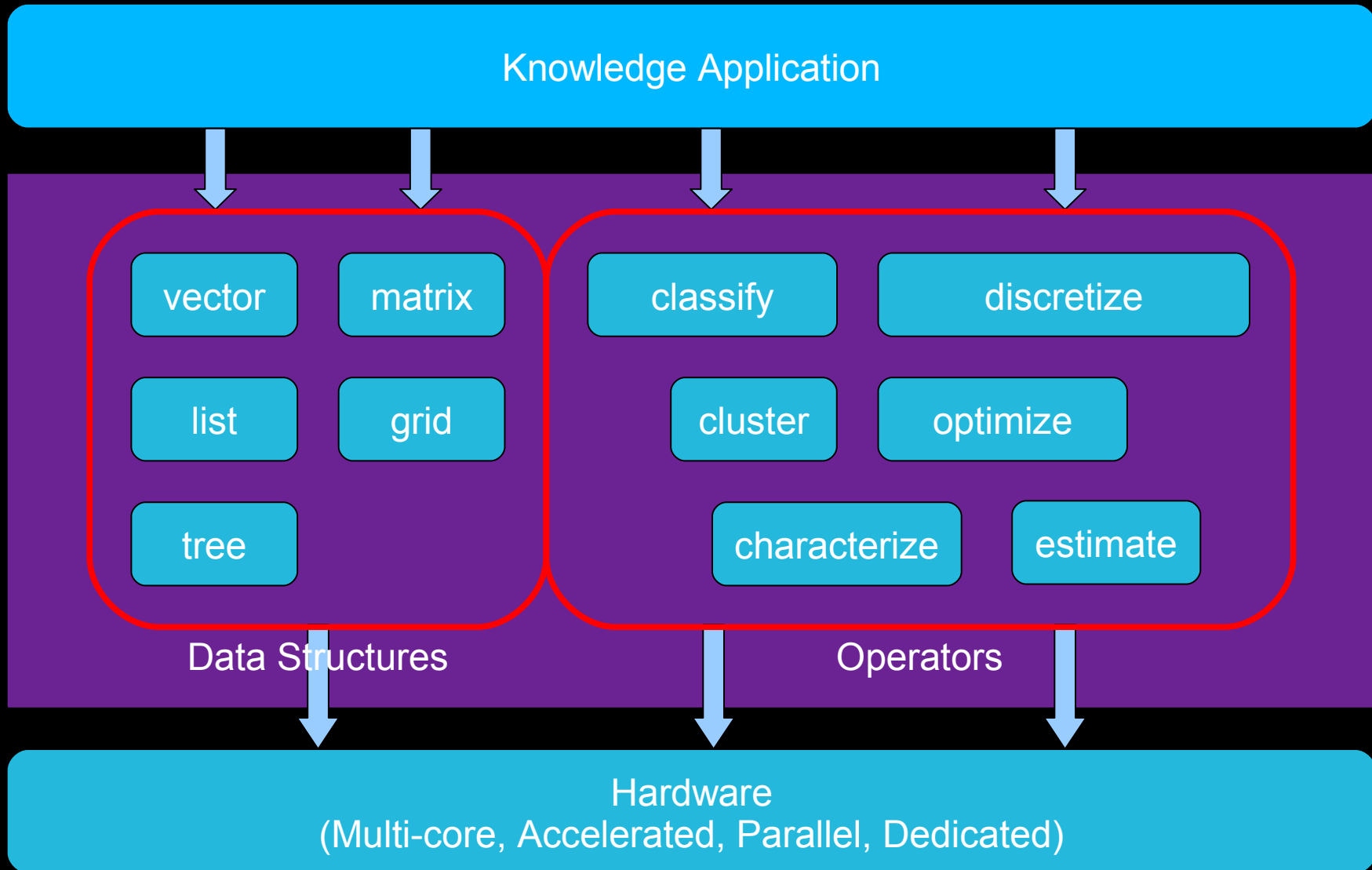
Separation of Concerns

- To solve a computational science problem you need
 - Discretization model and ordering to drive system assembly
 - Constraint solver
 - Reduction operator to map solution
- In the OpenKL Virtual Machine
 - Discretization model is abstracted to yield a constraint set
 - Constraint solver algorithm is implicit
 - Data structures are managed

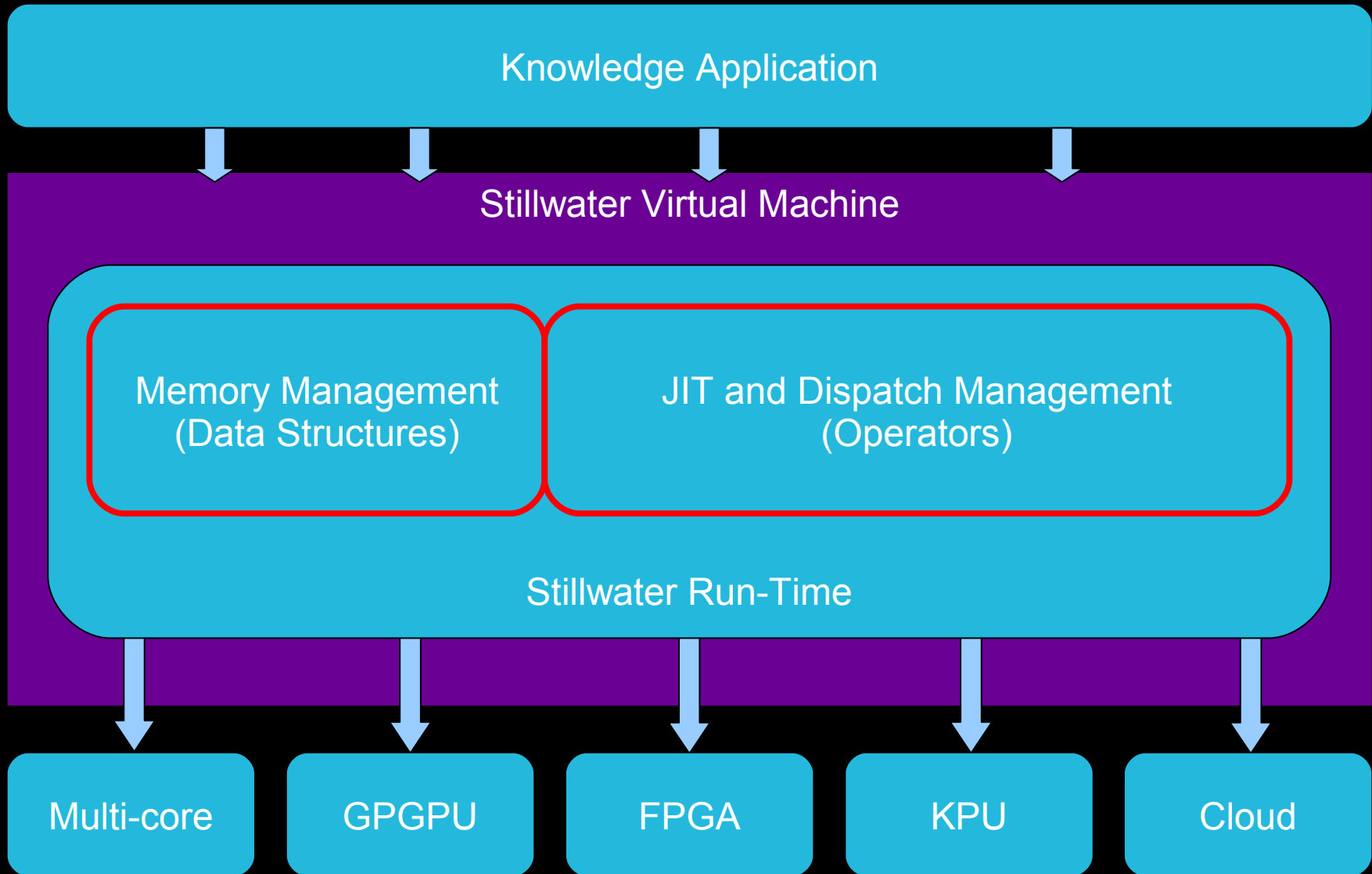
OpenKL Virtual Machine



OpenKL Types and Operators



Seamless Application Deployment



Stillwater Virtual Machine Abstractions

- Isolates domain-specific operations
 - Natural functional pipeline
 - Service Oriented Architecture
- Clarifies data structure ownership
 - Enables parallel algorithm performance
 - Simplifies automated optimizations
- Runs as a mini-OS
 - Resource management
 - Virtualizes parallel hardware so application can scale from multi-core to cloud without modification

Knowledge Application

Knowledge Engine

Cloud Virtualization

OpenKL Run-Time

Multi-core
(cpu, gpu, kpu, fpga)

OpenKL/MTL4

Knowledge Application

FEM
Knowledge Engine

FVM
Knowledge Engine

GN&C
Knowledge Engine

Statistics
Knowledge Engine

OpenKL Run-Time

MTL4

CST

FFTW, OSKI

MKL, TBB,
ArBB, CC

ACML,
APPML

CUDA

SciEngine

Stillwater

DMM

Intel

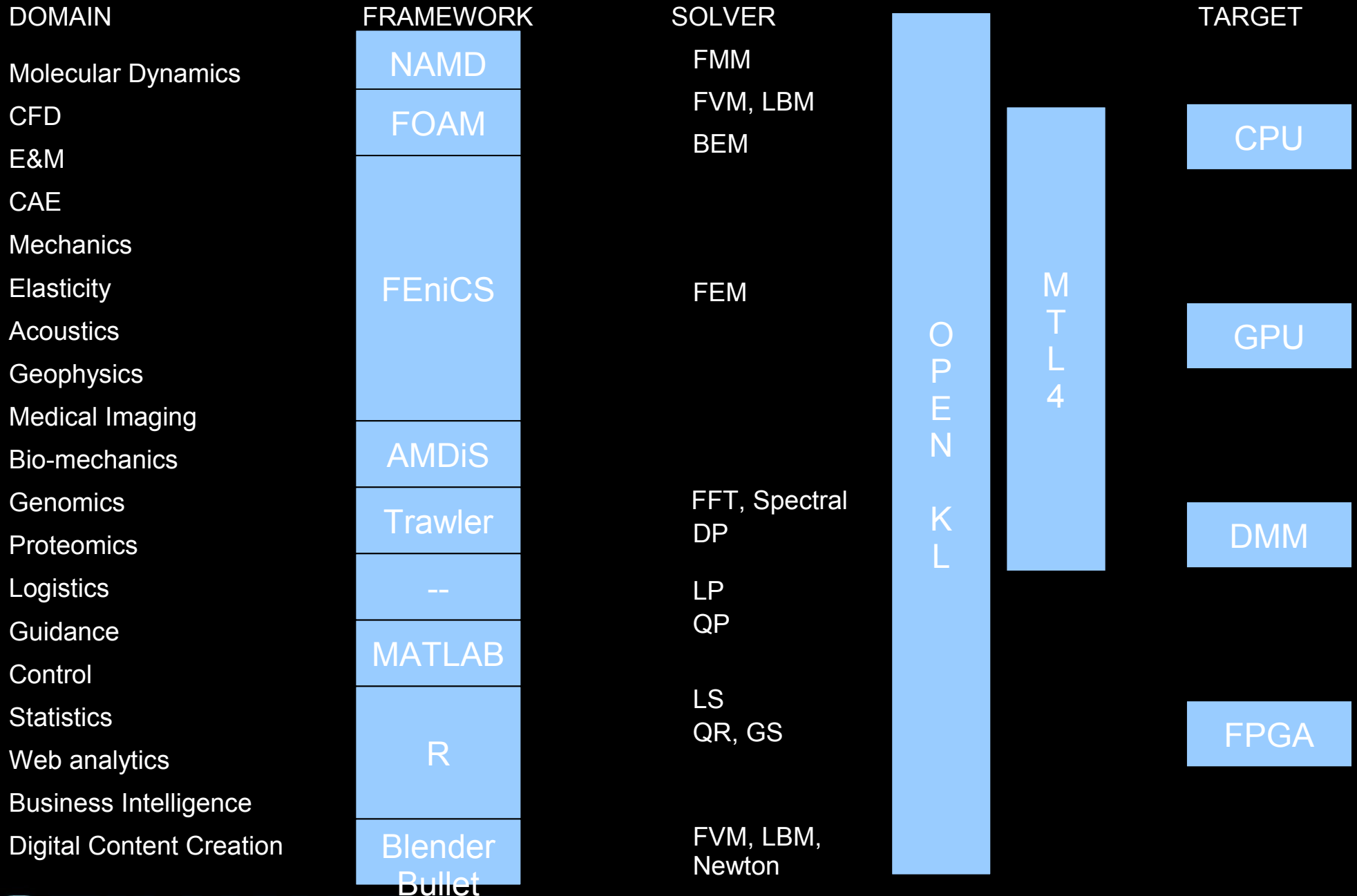
AMD

NVIDIA

FPGA

KPU

OpenKL/MTL4 Projects



For more information contact

Theodore Omtzigt

theo@stillwater-sc.com