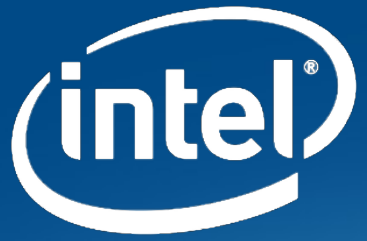


Fuel your Insight

Code Modernization and Software Defined Visualization

Jefferson Amstutz, Software Engineer

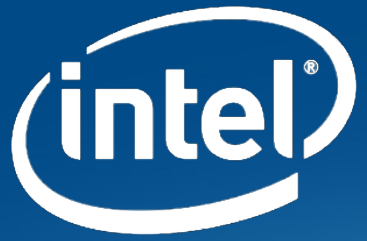
Legal Notices and Disclaimers

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Agenda

- Software Defined Visualization (SDVis)
  - Motivation
  - Intel SDVis Projects
  - SDVis Examples
- Delivering High Performance CPU Libraries
  - Multithreading
  - Vectorization

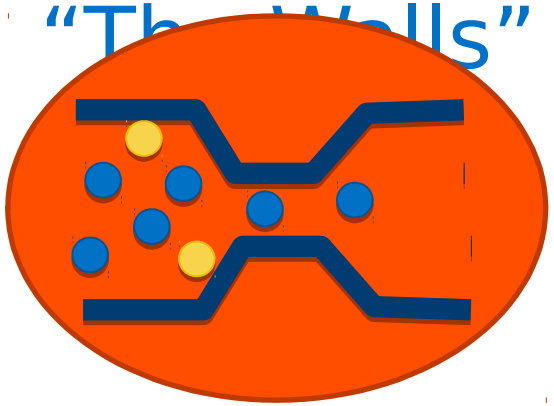


Software Defined Visualization

Growing Challenges in HPC

System Bottlenecks

"The Walls"



Memory | I/O | Storage
Energy Efficient Performance
Space | Resiliency | Unoptimized Software

Divergent Workloads

Machine Learning

HPC

Big Data

Visualization

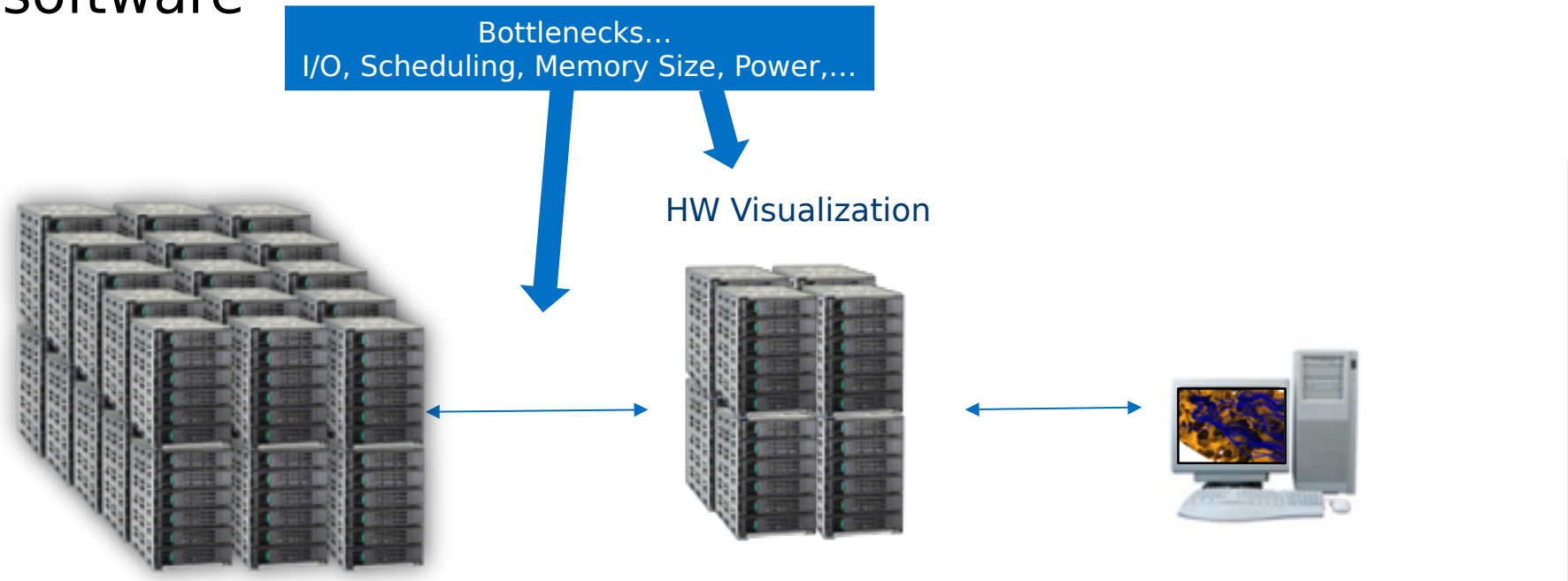
Resources Split Among Modeling and Simulation | Big Data Analytics | Machine Learning | Visualization

Barriers to Extending Usage

Democratization at Every Scale | Cloud Access | Exploration of New Parallel Programming Models

The Challenge Moving to ExaScale
Ex: *Visualization Analysis*

2008->2016: using dedicated hardware and specialized software



How?
Intel-Supported Software Defined Visualization (SDVis)!

http://software.intel.com/sdvis www.sdvis.org

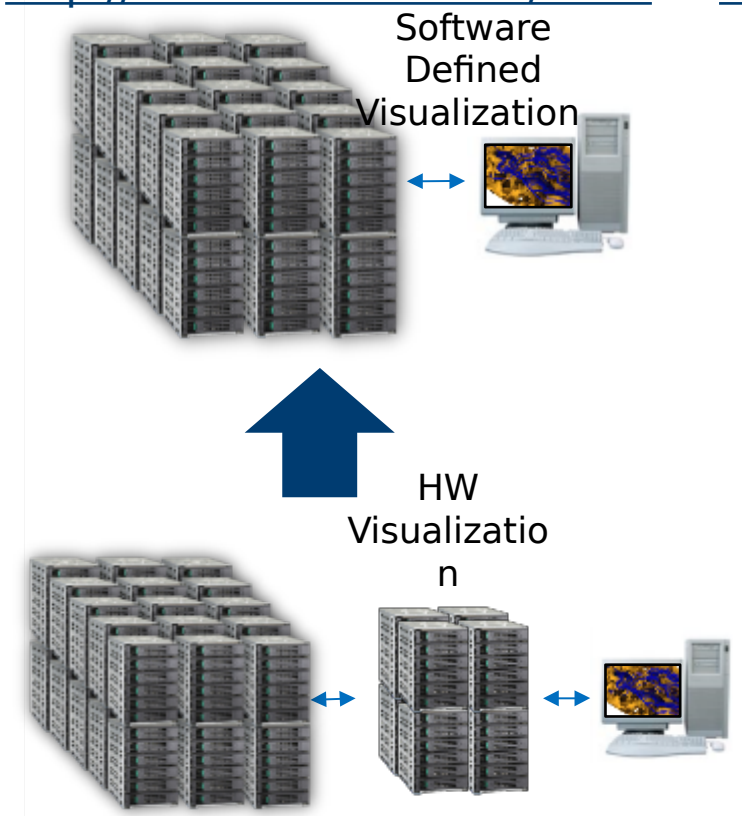
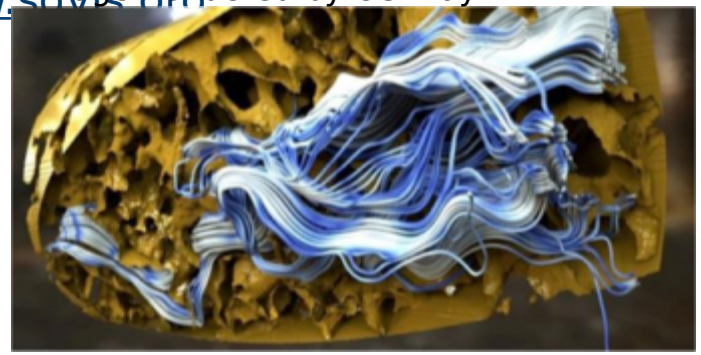
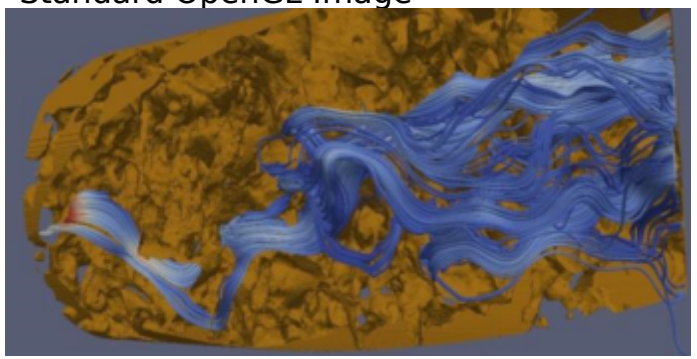


Image Rendered by OSPRay



Standard OpenGL Image



Embree

- CPU Optimized Ray Tracing Algorithms
- 'Tool kit' for Building Ray Tracings Apps
- Broadly Adopted by 3rd Party ISVs
- More at http://embree.github.io

OSPRay

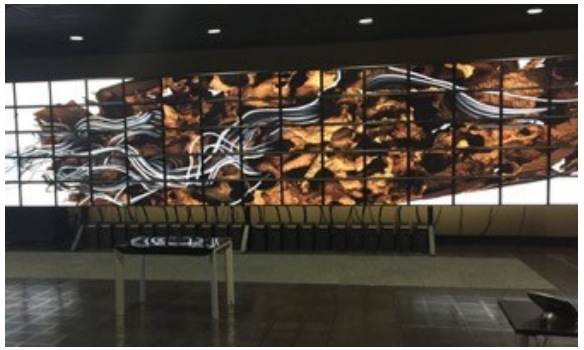
- Rendering Engine Based on Embree
- API Designed to Ease Creation of Visualization Software
- More at http://ospray.org

OpenSWR

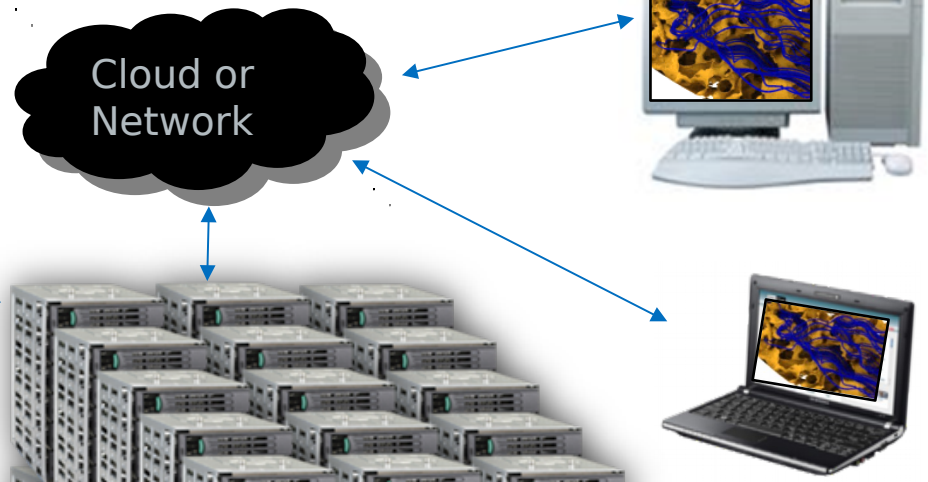
- High Performance CPU Vis Rasterization
- Fully Integrated into MESA v12.0+
- Supports ParaView, Visit, VTK, EnSight, VL3
- More at http://mesa3d.org ; www.openswr.org

Our Vision: Scalable, Flexible Vis Rendering that Runs Anywhere!

Standalone Laptops or Workstations



Big Memory Nodes or Rendering Focused Clusters



Large Compute+Vis clusters with Local or Remote Clients

Goal: Address Large-scale, High Performance, and High Fidelity Visualization via "Modern" Software

Gain deeper understanding of data impacting science & discovery

High fidelity, more realistic images even as data sets become increasingly larger,

and more complex; no need to compromise data resolution

Solve computing + modeling problem together (in-situ vis)

Essential SW development suite that makes concurrent simulation and visualization efficient – users can work interactively and get results quicker

One system

Use same system for both simulation and visualization, avoid data transfer

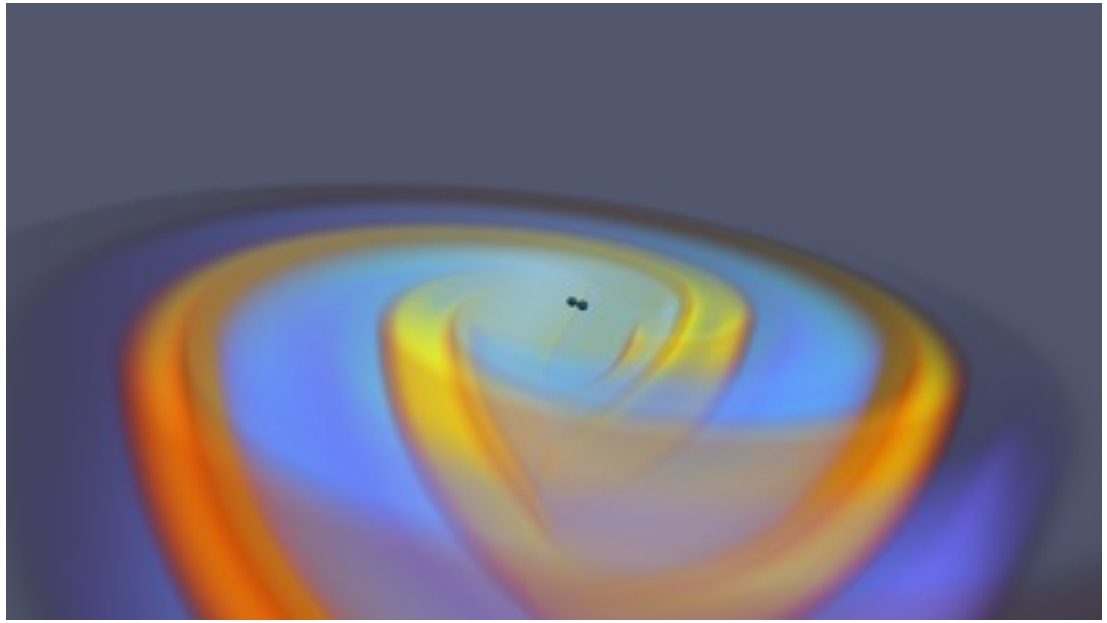
dela g toughest

proble

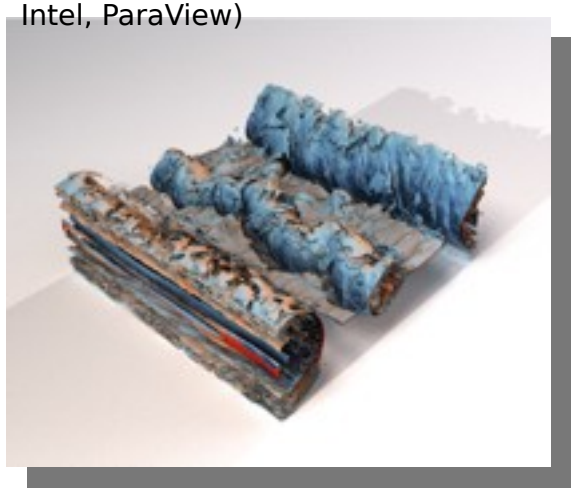
No GPU Needed!

Benefits of SDVis

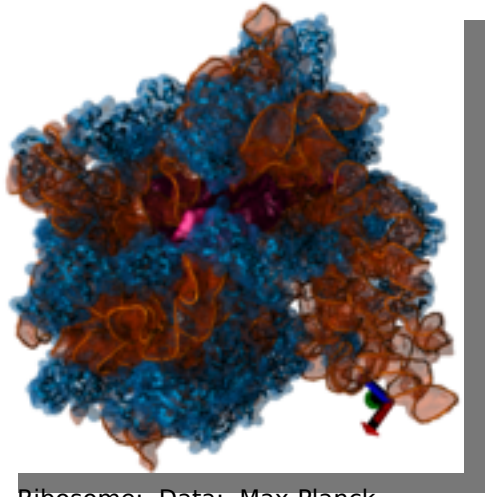
- Open-sourced technology delivering vivid visualization of complex, enormous data sets
- Innovative software libraries for visualizing results with *high performance* by unlocking the parallelism already in your system
- *High-fidelity* images for gaining deeper insights in science and industry, faster
- Software Only solution *lowers costs* – no card cost, no card maintenance, lower power bills



Gravational Waves : GR-Chombo AMR Data, Stephen Hawking CTC, UCambridge; Queens College, London; visualization, Carson Brownlee, Intel, ParaView)



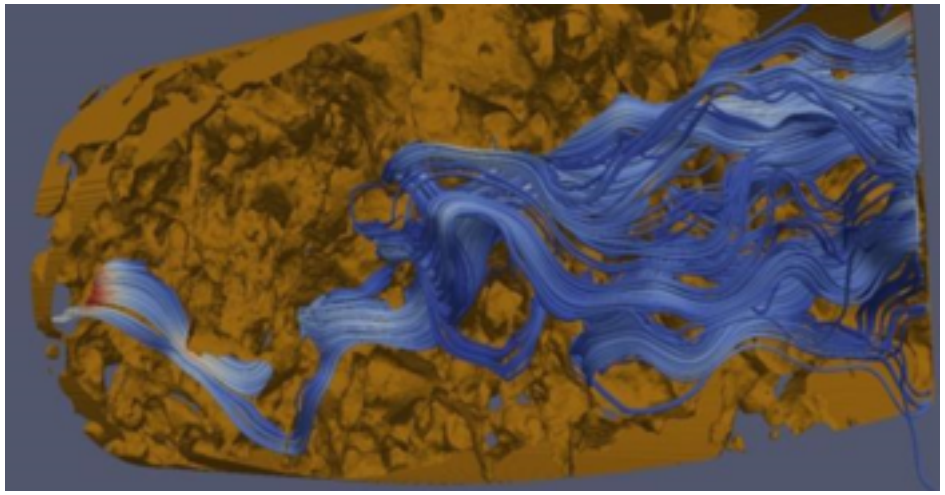
Magnetic Reconnection Model, Courtesy Bill Duaghton(LANL) and Berc Geveci(Kitware)

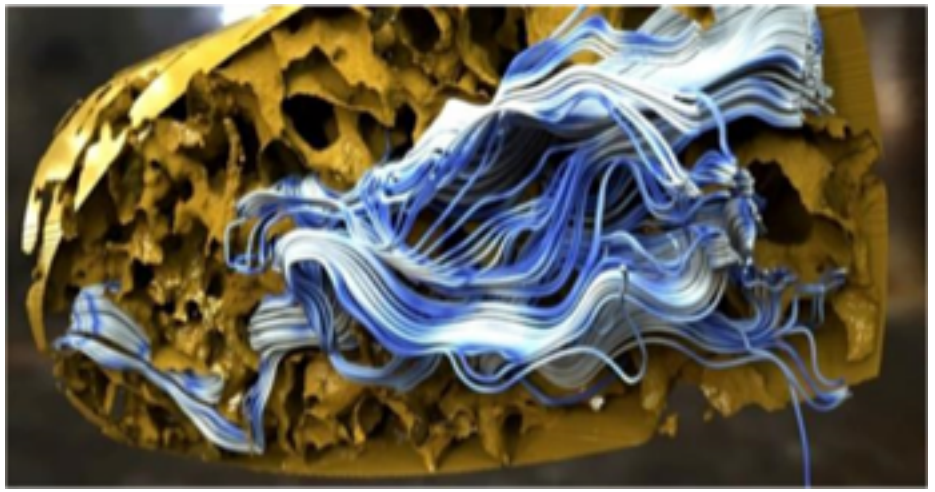


Ribosome: Data: Max-Planck Institute for Biophysical Chemistry

Hi-Fidelity Visualization with…

- … scalable image quality
- … scalable data model size
- … scalable rendering cost



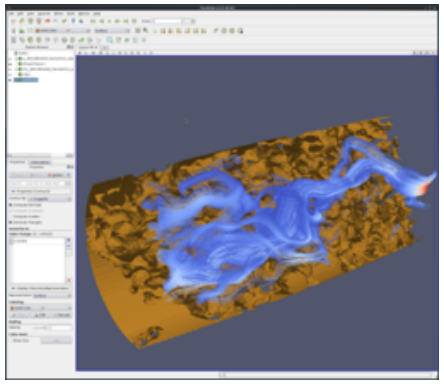


Data set provided by Florida International University

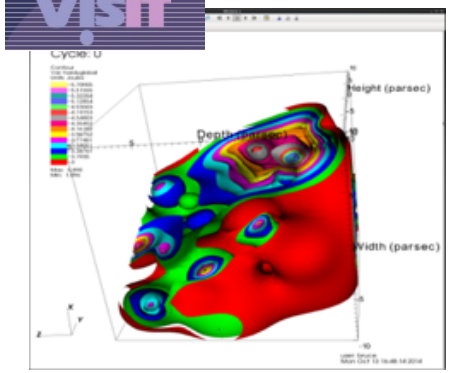
OpenSWR Software Rasterizer (www.mesa3d.org www.openswr.org)

- High performance open source software implementation of OpenGL* rasterizer
  - Fully multi-threaded and vectorized for Intel® processors
  - Can access full system memory - highest resolution data
  - Leverages community development effort (MESA)
- Drop in replacement for OpenGL library
- Available since July'16 in Mesa v12.0+ targeting features and performance for leading SciVis Apps

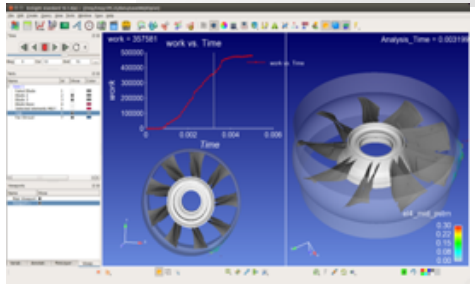
ParaView



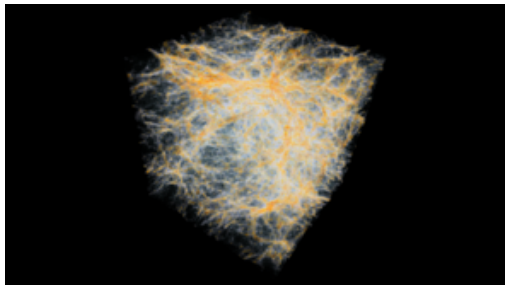
VisIt



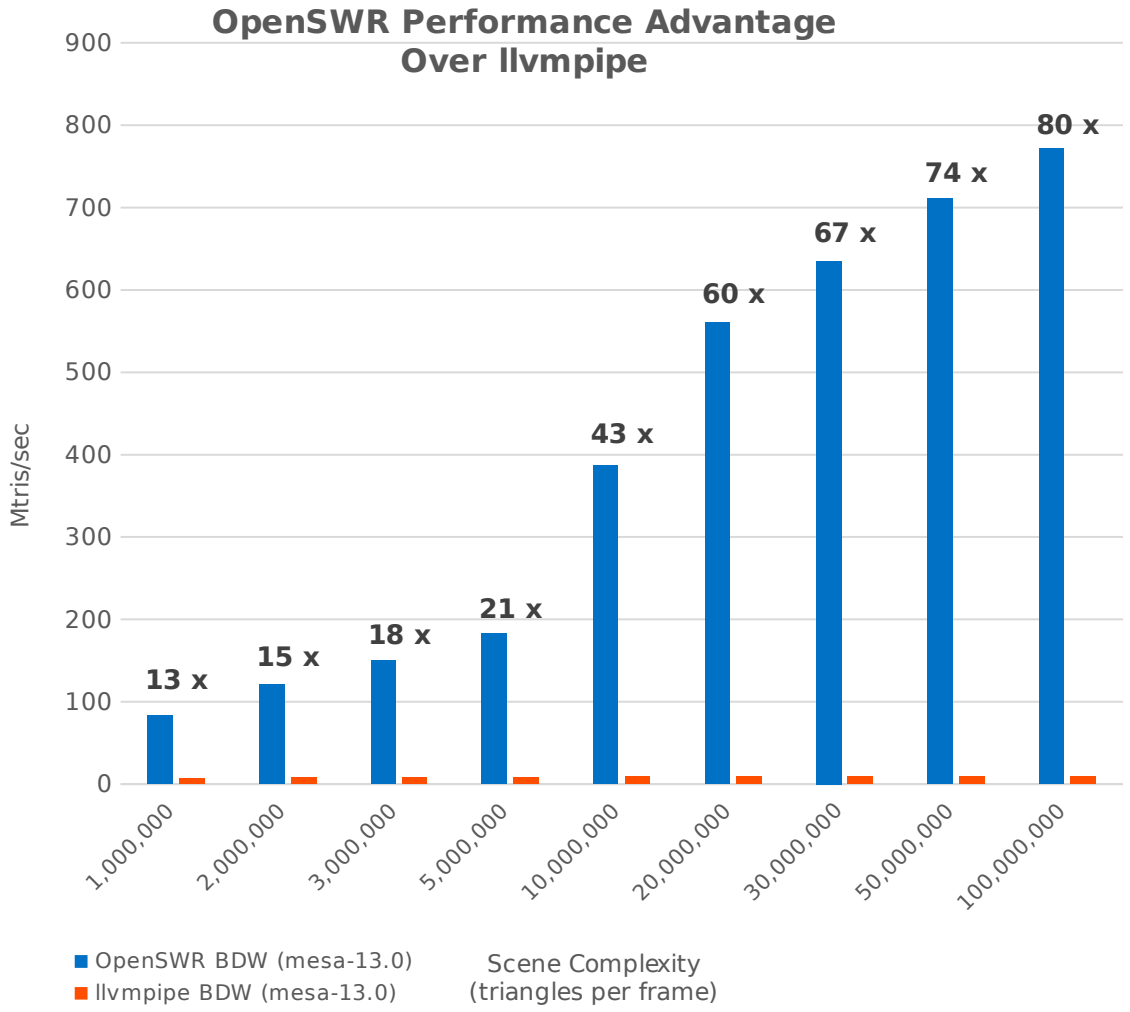
CEI EnSight Extreme Visualization Software



Argonne NATIONAL LABORATORY VL3

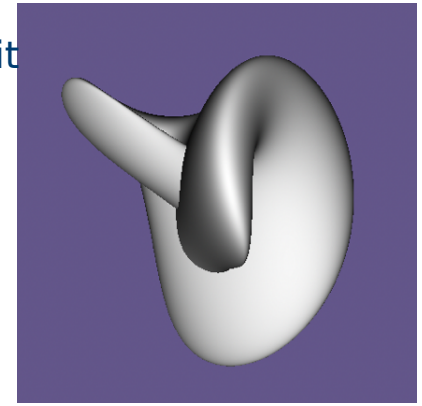


VTK Benchmark + OpenSWR



Application: VTK GLBenchmarking

Description: Scientific visualization toolkit



Availability:

- **Code:** *http://openswr.org*
- **Recipe:** *http://openswr.org*

Usage Model:

- Native on Xeon, with full core utilization
- AVX/AVX2 Intrinsics

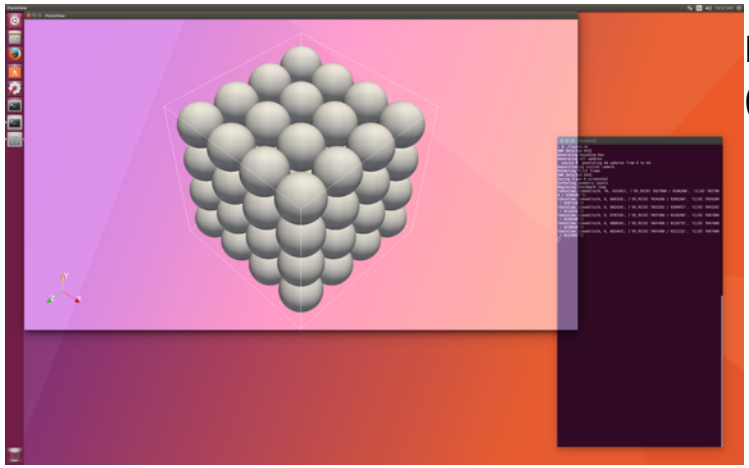
Highlights:

- OpenSWR enables simple OpenGL driver replacement, no app change
- VTK is the scientific visualization toolkit behind ParaView and VisIt, and a good platform for showing comparative performance across typical vis rendering workloads
- End-User benefits: Ability to achieve competitive performance and the flexibility of IA for rendering / visualization applications

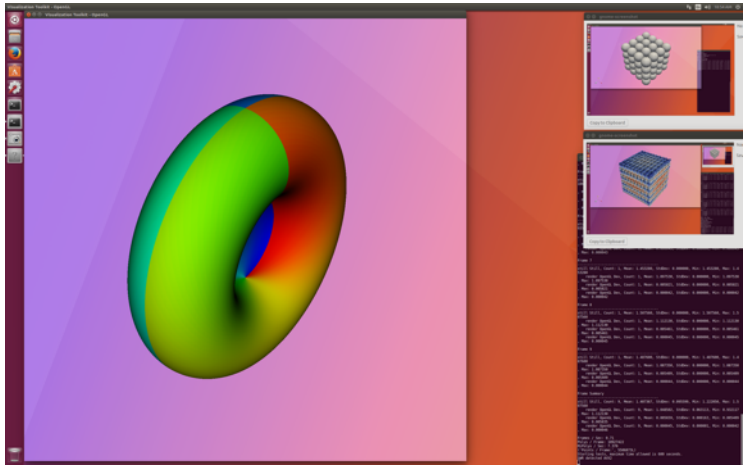
Results:

- OpenSWR provides interactive rendering across tested workloads and dramatic performance improvements over mesa llvmpipe
- OpenSWR on Xeon exhibits strong core scaling to optimally utilize available resources. Further KNL optimizations are in progress.
- Significant performance improvements over Alpha 3.2 release a year ago. Many additional performance areas to explore.

Mesa/OpenSWR Benchmarks

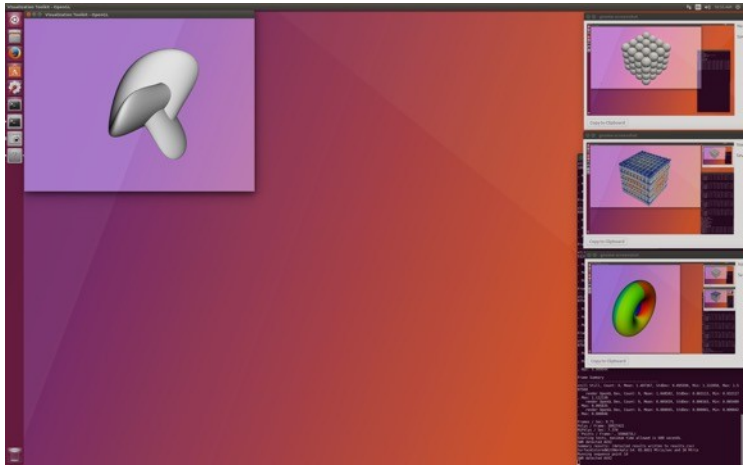


manyspheres.py
67 MiPolys

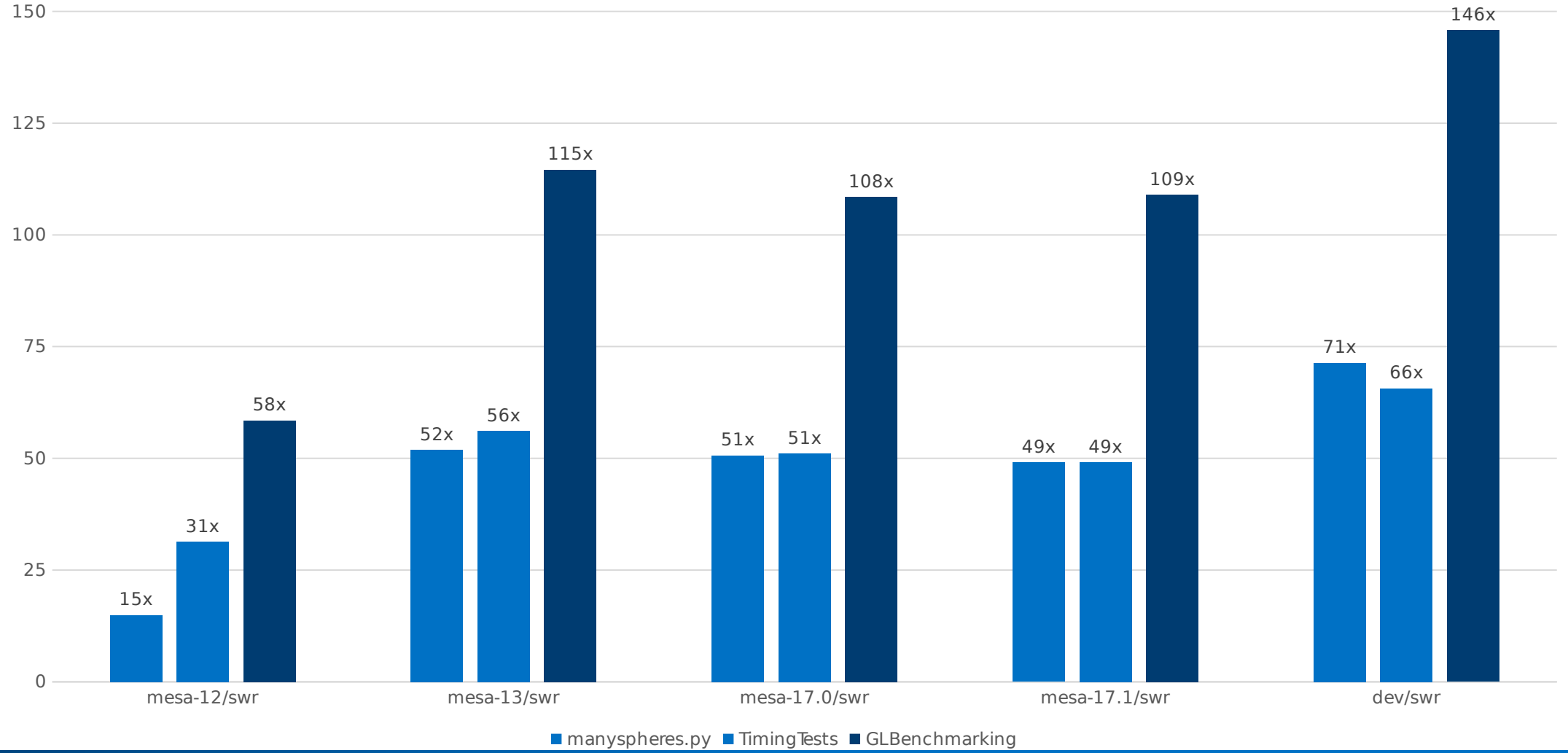


TimingTests
30 MiTris

GLBenchmarking
30MiTris



Intel® Xeon Phi™ 7210 Processor
SWR vs. Mesa LLVMPIPE PERFORMANCE RATIO OVER TIME



Ray Tracing Foundation: Embree Ray Tracing Kernel Library

Provides highly optimized and scalable ray tracing kernels

- Acceleration structure build and ray traversal
- Single Ray, Ray Packets(4,8,16), Ray Streams(N)

Targets up to photorealistic professional and scientific rendering applications

Highest ray tracing performance on CPUs

- 1.5–6× speedup reported by users

Support for latest CPUs / ISAs

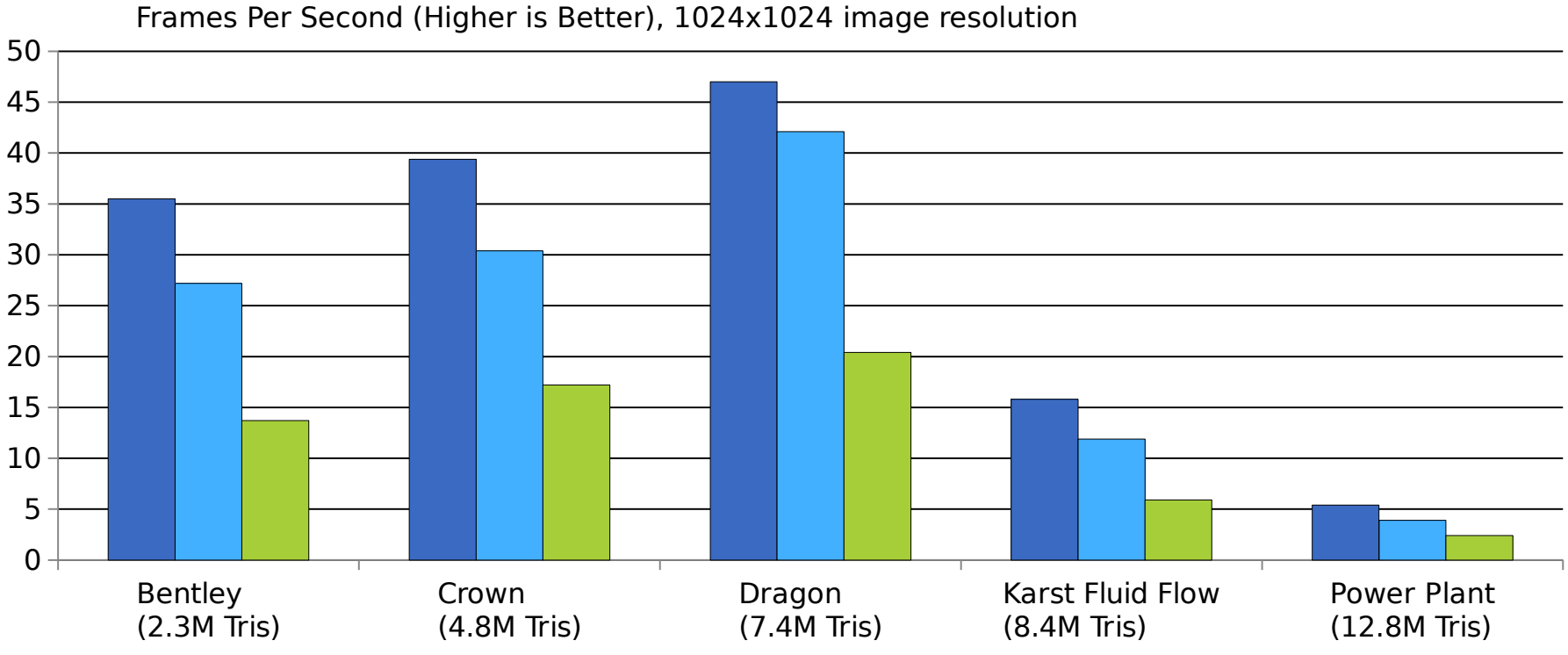
- Intel® Xeon Phi™ Processor (codenamed *Knights Landing*) – AVX-512

API for easy integration into applications

Free and open source under Apache 2.0 license

- http://embree.github.com

Performance: Embree vs. NVIDIA* OptiX*



Embree 2.13.0, ICC 2016 Update 1, Intel® SPMD Program Compiler (Intel® ISPC) 1.9.1

NVIDIA* OptiX* 4.0.1, CUDA* 8.0.44

Source: Intel

Embree Adoption*

DreamWorks

Ubisoft Motion Pictures

Autodesk

SURVICE Engineering Company

EasternGraphics visualize your business

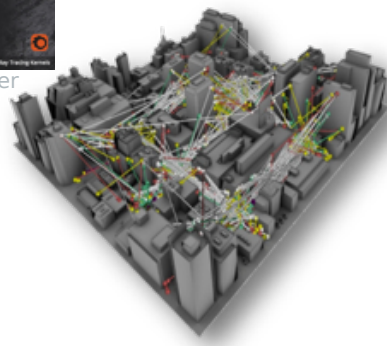
FluidRay RT Physically Based Renderer

corona

Simlab Soft Bringing Art closer

3DEXCITE

Courtesy of Jeff Patton, Rendered with Corona Renderer



Rendered with StingRay, SURVICE Engineering

Image rendered with FluidRay RT

pCon.planner rendered courtesty EasternGraphics

*Many other announced users incl.: Pixar, Weta Digital, Activision, Chaos V-Ray, Ready At Dawn, FrostBite, EpicGames UnReal, High Moon, Blue Sky, UBISoft MP, Framestore, ...

OSPRay: A Ray-Tracing based Rendering Engine for *High-Fidelity* Visualization

- Build on top of Embree; Launched June 2016
- Scalable Visualization targeted features
  - Surfaces (both polygonal and non-polygonal)
  - Volumes, and volume rendering
  - *High-Fidelity* rendering/shading methods
  - Scalable Cluster Wide Rendering
- Packed it up in an ‘easy-to-use’ rendering library for visualization
  - Same "spirit" as OpenGL, but different API

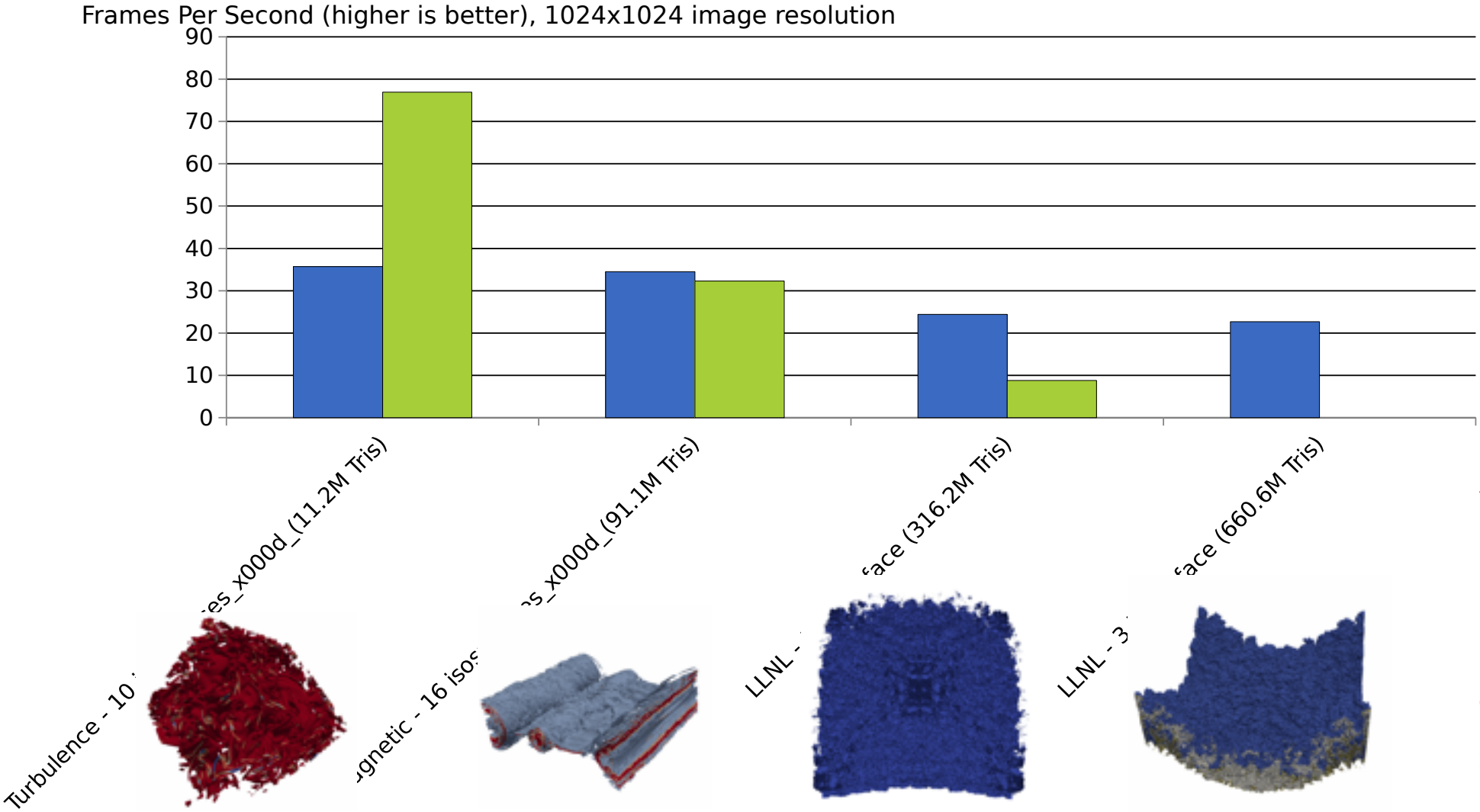
Brayns

PowerCT

NASA

VL3

Performance: OSPRay vs. GPU



Intel® Xeon® Processor E5-2699 v3
2 x 18 cores, 2.3 GHz

ParaView 5.2.0 RC2 w/ OSPRay 1.1.0

NVIDIA* Driver 370.28, default configuration

Source: Intel

Software Defined Visualization (SDVis)

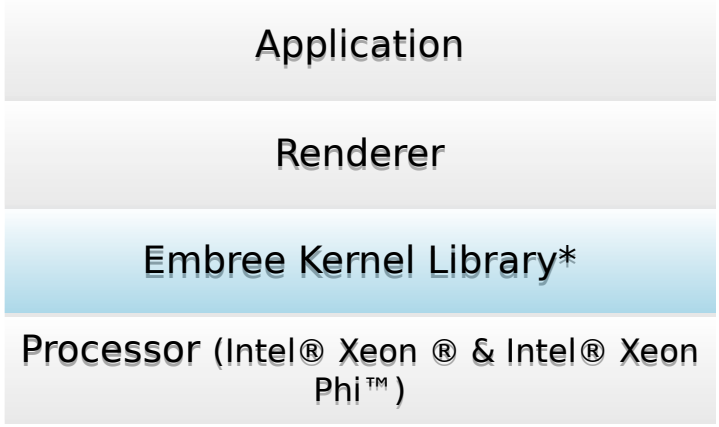
What is it?

Open-Source Libraries Developed by Intel
Used in leading applications for visualization
Optimized for parallel processing architectures

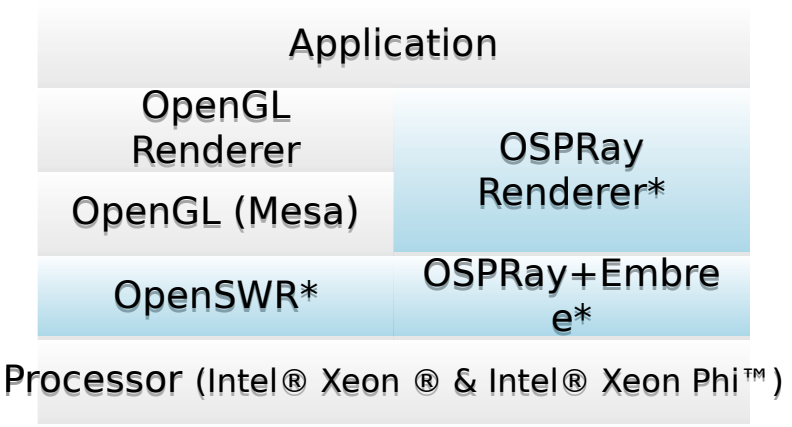
Provides

Access to large memory space for large data sets
Improved visual fidelity
Cost efficient, interactive visualization performance

Professional Rendering



Scientific Visualization

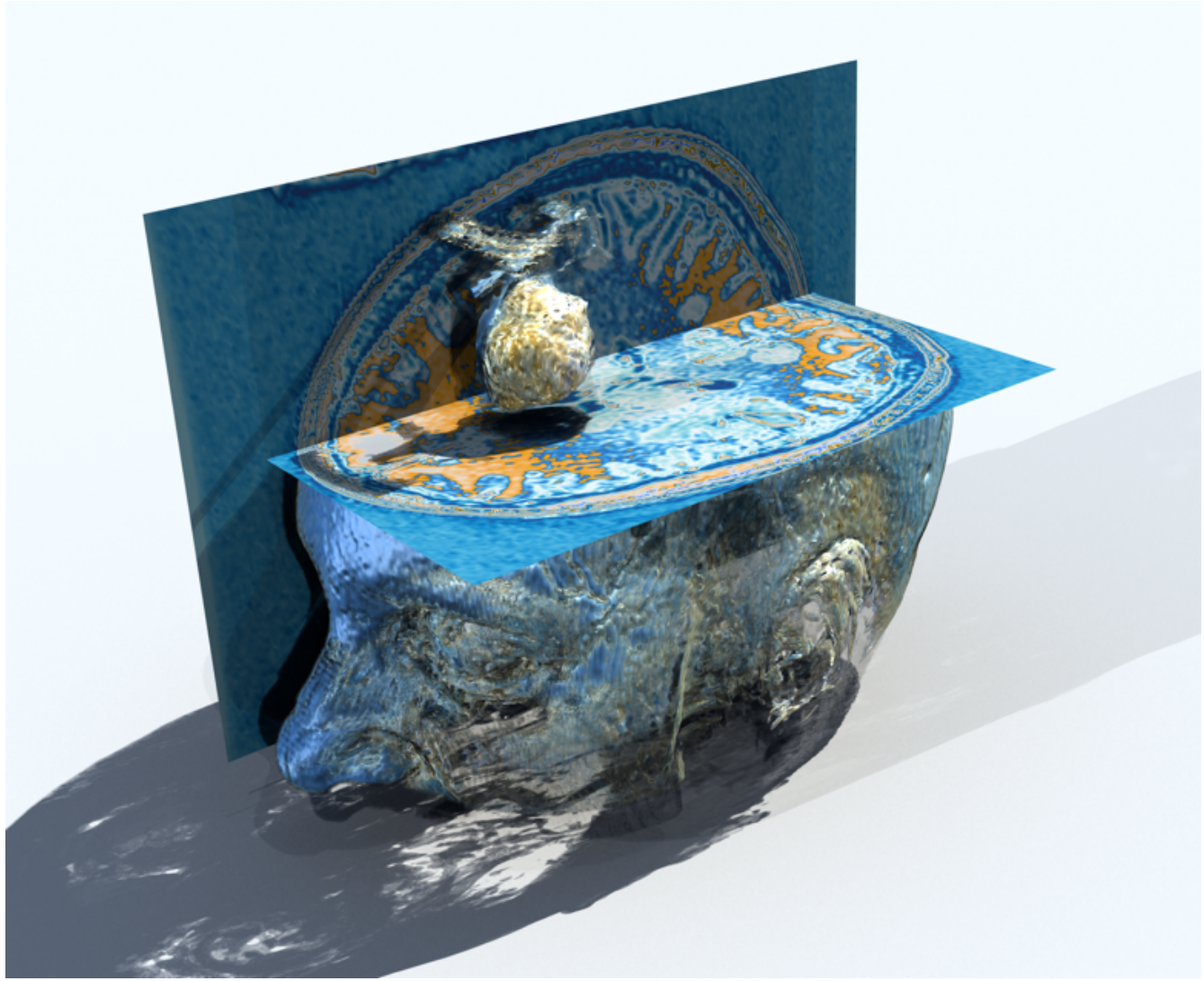


* Intel® Developed Software

Software defined visualization is the use of optimized libraries to enable high-performance, high-fidelity visualization applications on Intel® platforms

ParaView v5.x with integrated OSPRay and OpenSWR

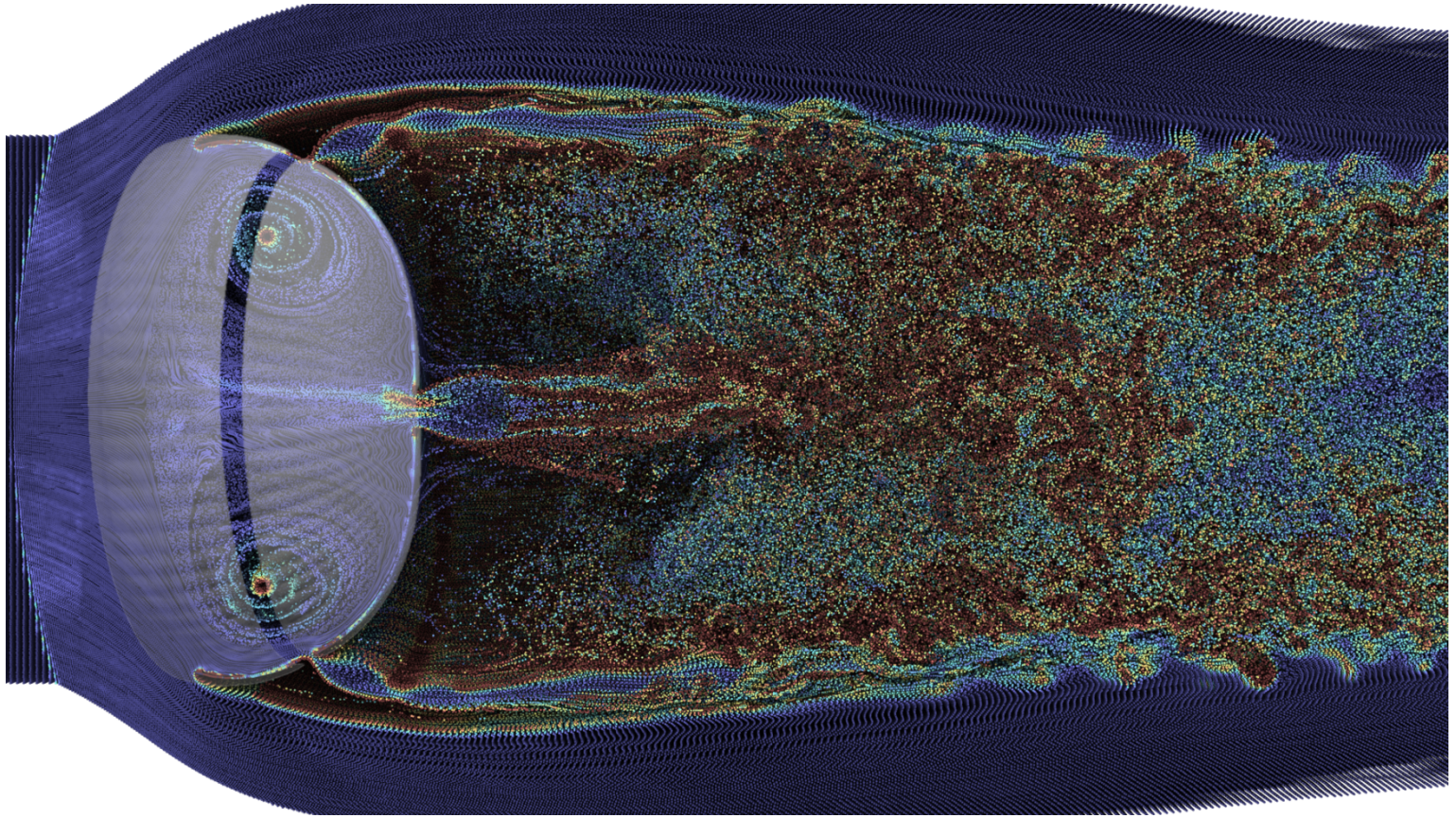
- Brain Tumor monitoring and treatment
- 3D interactive @ 10-20fps
- Intel® Xeon Phi™ processor cluster
- Ambient occlusion plus shadows
- Stop by the Intel SC’16 booth to see it live!



- Data courtesy Kitware. Visualization, Carson Brownlee, Intel

NASA – Custom OSPRay App

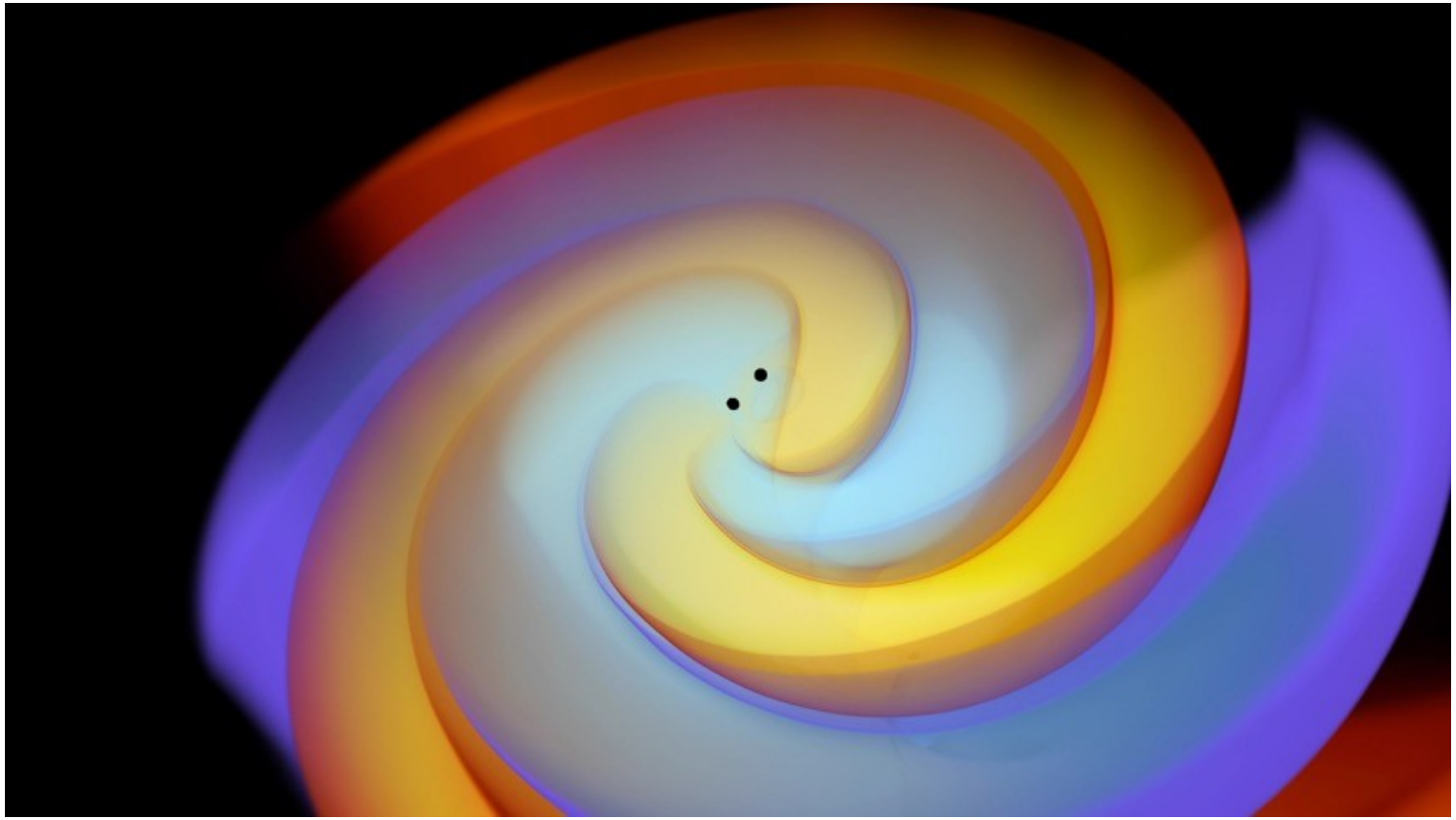
- Rendered on Pleiades supercomputer attached Vis wall cluster



dataset: parachute; simulation: Dr M. Barad, NASA Ames; visualization: Tim Sandstrom, NASA Ames

Stephen Hawking Centre for Theoretical Cosmology – ParaView / VTK with OSPRay

- 600 GB Memory Footprint
- 36 TB Simulation Data Set
- 4 Intel® Xeon Phi™ 7230 Processors
- 1 Intel® Xeon® E5 v4 Dual Socket node
- Intel® Omni-Path Fabric
- ~10 fps



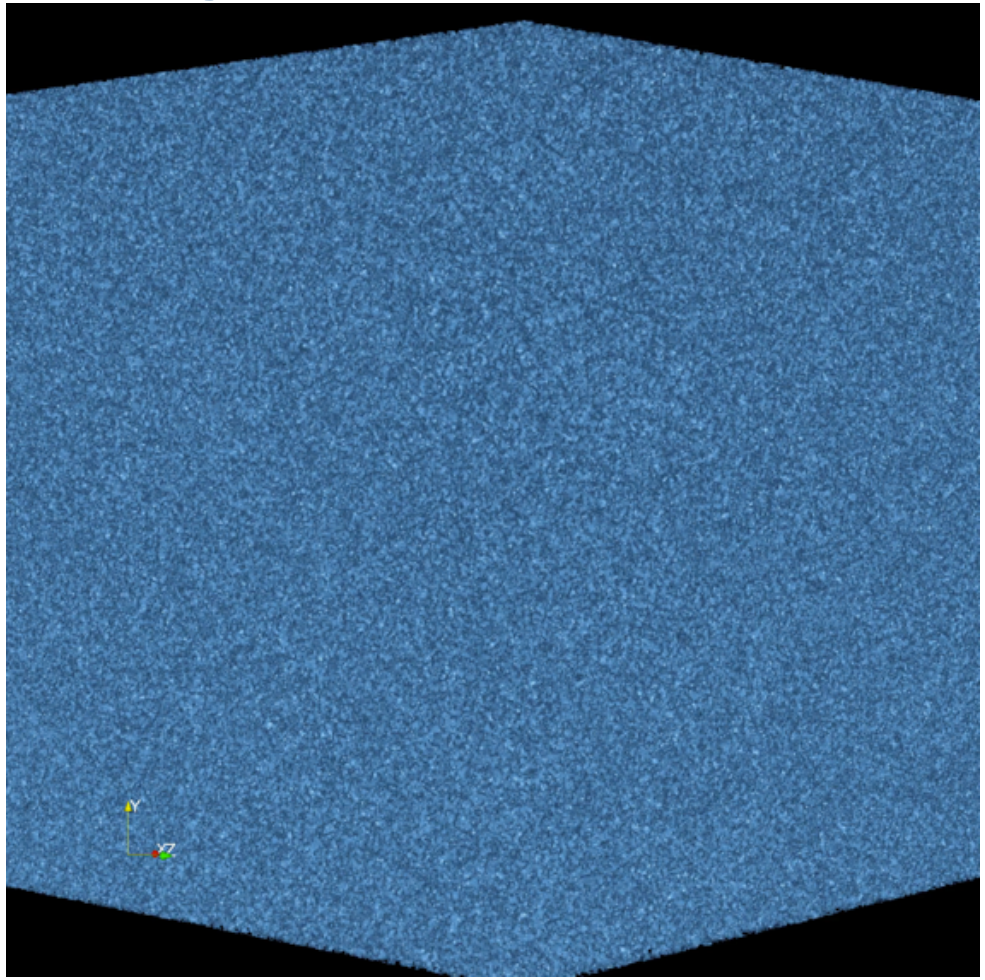
Gravational Waves : GR-Chombo AMR Data, Stephen Hawking CTC, UCambridge; Queens College, London; visualization, Carson Brownlee, Intel)

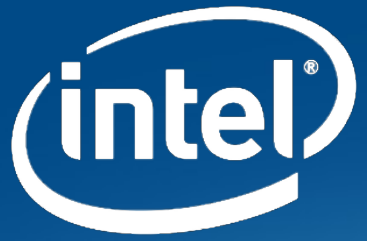
Stephen Hawking Centre for Theoretical Cosmology – ‘Walls’ in situ with OSPRay Rendering

- 10 TB Memory Footprint
- SGI UV-300 16TB SMP
- >1000 Shared memory Intel® Xeon® E5 v3 processors
- ~15 fps

- Domain Wall formation in the universe from Big Bang to today (13.8 billion years)

- Simulation code by Shellard et al, Visualizaiton by Johannes Gunther (Intel)

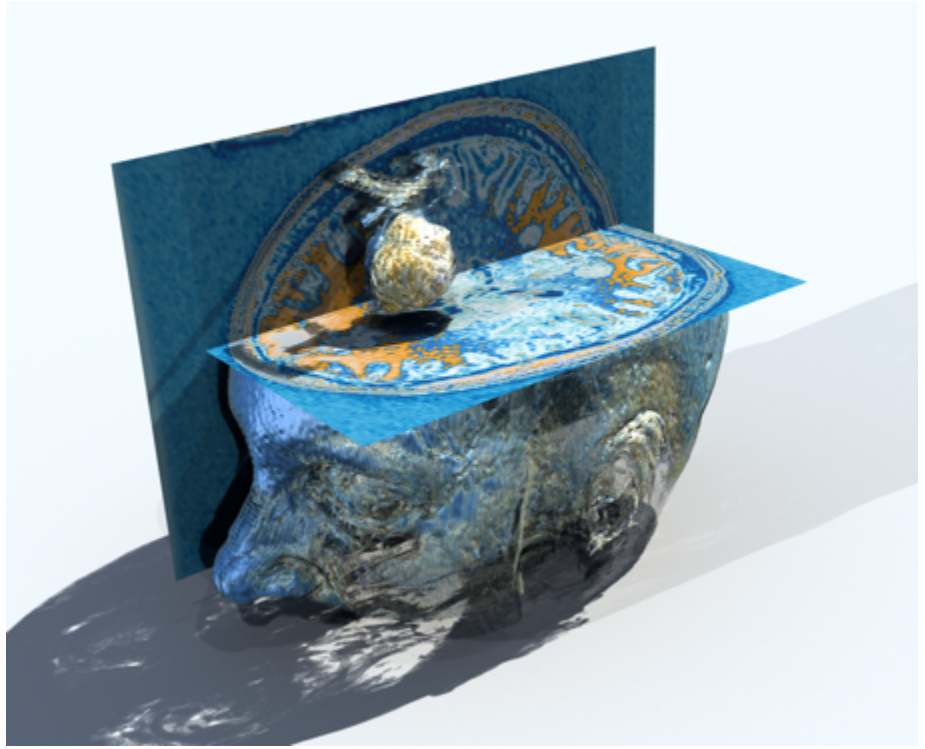




Delivering High Performance CPU Libraries

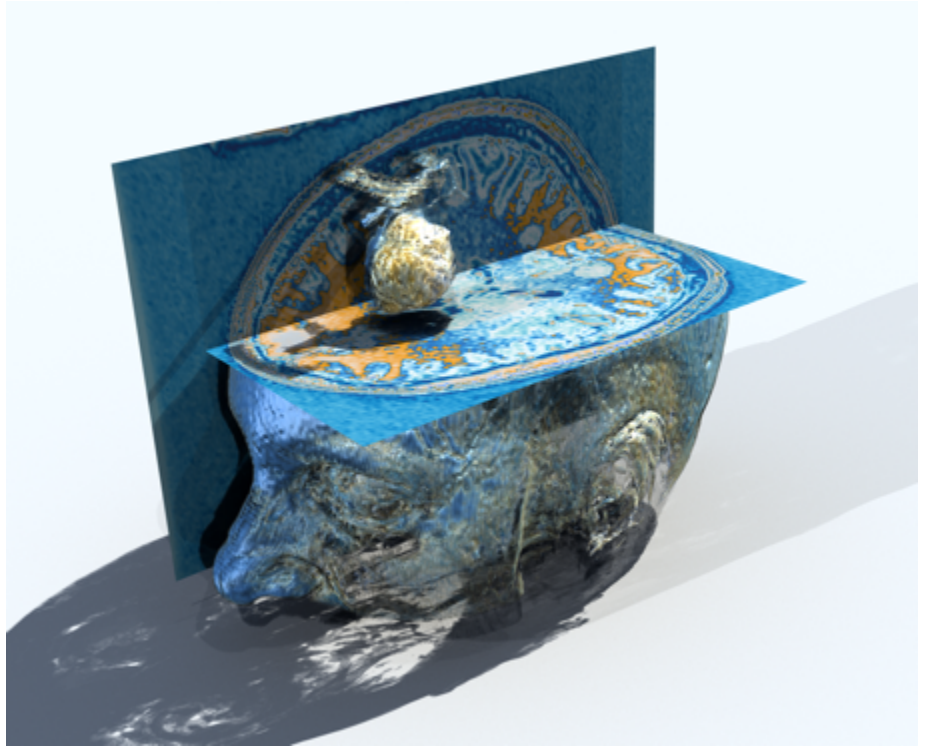
Exploiting Parallelism

- Performance is a big driver for HPC as it guides…
  - …hardware procurement decisions,
  - …software infrastructure and design decisions,
  - …resulting in scientific discoveries!
- Achieving great performance can be difficult, but the formula is surprisingly consistent
  - Parallelism, parallelism, and more



Exploiting Parallelism

- Performance is a big driver for HPC as it guides…
  - …hardware procurement decisions
  - …software infrastructure and design decisions
  - …resulting in scientific discoveries!
- Achieving great performance can be difficult, but the formula is surprisingly consistent
  - Parallelism, parallelism, and more



Exploiting Parallelism - Methods

- Parallelism is NOT the same as concurrency
  - Composable constructs, both are useful for getting good performance
- Processor clock frequencies and power requirements have led HPC hardware to scale up with parallel execution
- 3 major methods of scaling computation:
  - Vector instructions (ex: AVX®, AVX2®, AVX512®)
  - Multicore (ex: Intel Thread Building Blocks®, libiomp, Cilk+)
  - Multi Node (ex: Intel MPI®)

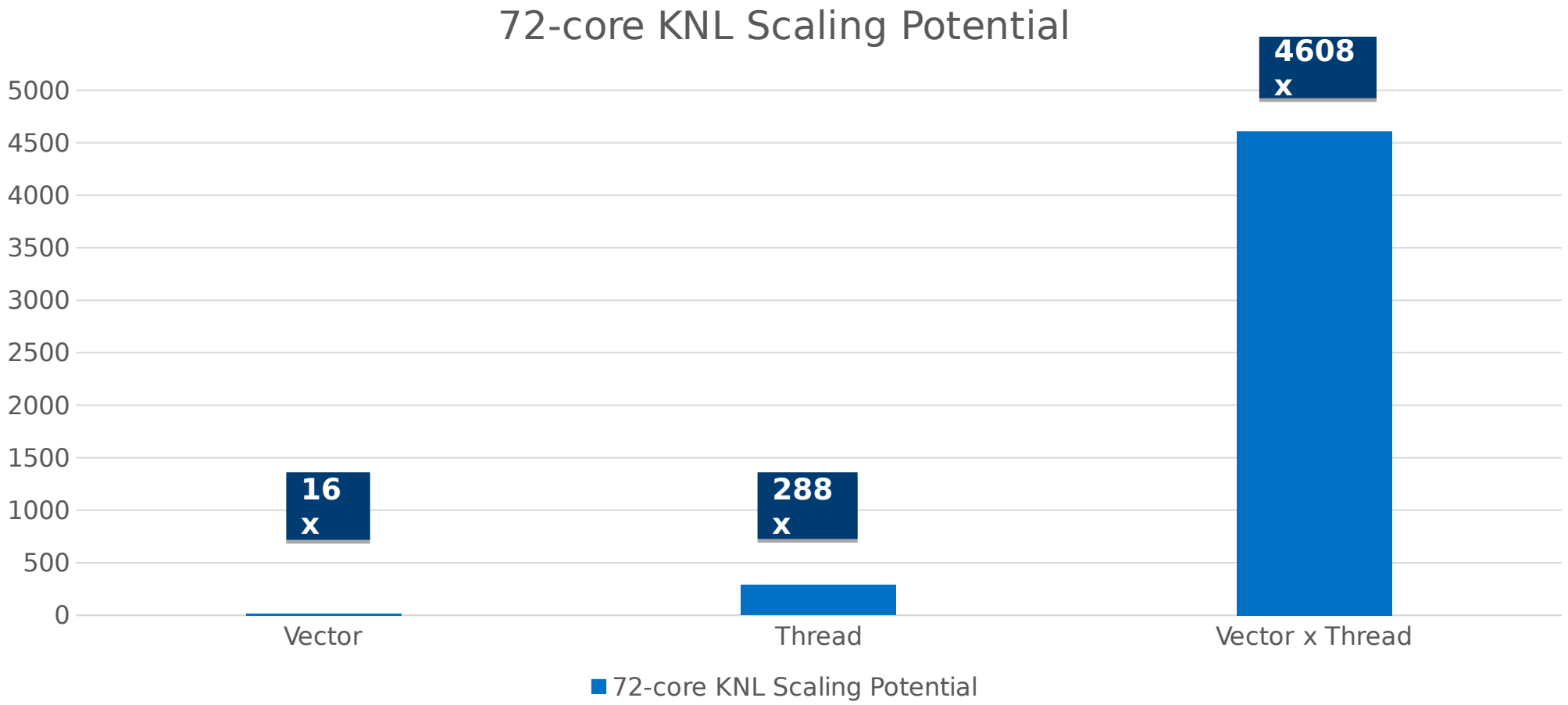
Exploiting Parallelism - Methods

- Parallelism is NOT the same as concurrency
  - Composable constructs, both are useful for getting good performance
- Processor clock frequencies and power requirements have led HPC hardware to scale up with parallel execution
- 3 major methods of scaling computation:
  - Vector instructions (ex: AVX®, AVX2®, AVX512®)
  - Multicore (ex: Intel Thread Building Blocks®, libiomp, Cilk+)
  - Multi Node (ex: Intel MPI®)

Exploiting Parallelism - Methods

- Parallelism is NOT the same as concurrency
  - Composable constructs, both are useful for getting good performance
- Processor clock frequencies and power requirements have led HPC hardware to scale up with parallel execution
- 3 major methods of scaling computation:
  - Vector instructions (ex: AVX®, AVX2®, AVX512®)
  - Multicore (ex: Intel Thread Building Blocks®, libiomp, Cilk+)
  - Multi Node (ex: Intel MPI®)

Motivation– “Unlocking” FLOPS for Performance!



Exploiting Parallelism – Multicore

- Multicore CPUs have been ubiquitous for nearly 2 decades!
  - In that time, a lot of work has been done to make multicore programming easier
    - Compiler extensions, tasking libraries, new languages, etc…
- 2 ways to use more than one core
  - Multiprocessing (less effective, legacy programming model)
  - Multithreading (more effective, modern programming model)

Exploiting Parallelism – Multicore

- Multicore CPUs have been ubiquitous for nearly 2 decades!
  - In that time, a lot of work has been done to make multicore programming easier
    - Compiler extensions, tasking libraries, new languages, etc…
- 2 ways to use more than one core
  - Multiprocessing (less effective, legacy programming model)
  - Multithreading (more effective, modern programming model)

Multithreading – Programming Models

- Easy to be overwhelmed by various parallel programming models
  - Plain threads
    - (not flexible, difficult to port)
  - Futures, async(), and callbacks
    - better for concurrency than parallelism
  - Structured parallelism + generic parallel algorithms
    - Common in HPC, can reuse existing expertise
    - Easy to keep code portable

Multithreading – Programming Models

- Easy to be overwhelmed by various parallel programming models
  - Plain threads ×
    - (not flexible, difficult to port)
  - Futures, async(), and callbacks
    - better for concurrency than parallelism
  - Structured parallelism + generic parallel algorithms
    - Common in HPC, can reuse existing expertise
    - Easy to keep code portable

Multithreading – Programming Models

- Easy to be overwhelmed by various parallel programming models
  - Plain threads ×
    - (not flexible, difficult to port)
  - Futures, async(), and callbacks ×
    - better for concurrency than parallelism
  - Structured parallelism + generic parallel algorithms
    - Common in HPC, can reuse existing expertise
    - Easy to keep code portable

Multithreading – Programming Models

- Easy to be overwhelmed by various parallel programming models
  - Plain threads ×
    - (not flexible, difficult to port)
  - Futures, async(), and callbacks ×
    - better for concurrency than parallelism
  - Structured parallelism + generic parallel algorithms ✓
    - Common in HPC, can reuse existing expertise
    - Easier to keep code portable

Multithreading – Programming Models

- Structured parallelism comes in two flavors:
  - Compiler directives:

```
void foo(float *a, float *b, float *c) {
#pragma omp parallel for
  for (int i = 0; i < ARRAY_SIZE; ++i) {
    c[i] = a[i] + b[i];
  }
}
```

  - Tasking libraries:

```
void foo(float *a, float *b, float *c) {
  tbb::parallel_for(ARRAY_SIZE, [&](int i) {
    c[i] = a[i] + b[i];
  });
}
```

Multithreading – Programming Models

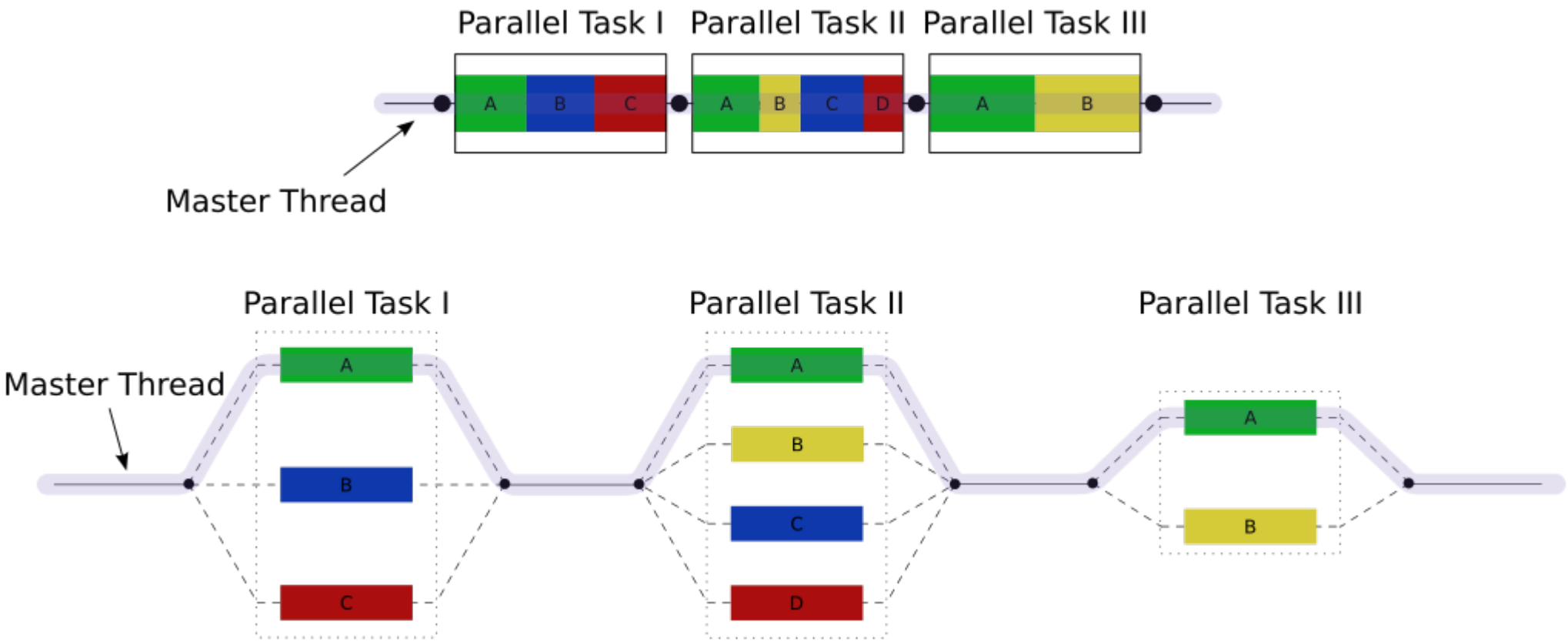
- Structured parallelism comes in two flavors:
  - Compiler directives:

```
void foo(float *a, float *b, float *c) {
#pragma omp parallel for
  for (int i = 0; i < ARRAY_SIZE; ++i) {
    c[i] = a[i] + b[i];
  }
}
```

  - Tasking libraries:

```
void foo(float *a, float *b, float *c) {
  tbb::parallel_for(ARRAY_SIZE, [&](int i) {
    c[i] = a[i] + b[i];
  });
}
```

Multithreading – “Fork-Join” Parallelism



Multithreading – Tasking Systems

- Thread: a unique stack of execution which may run in parallel
- Task: a unit of work (typically a function) to be run on a thread

Tasks are scheduled on a pool of threads to optimize machine utilization

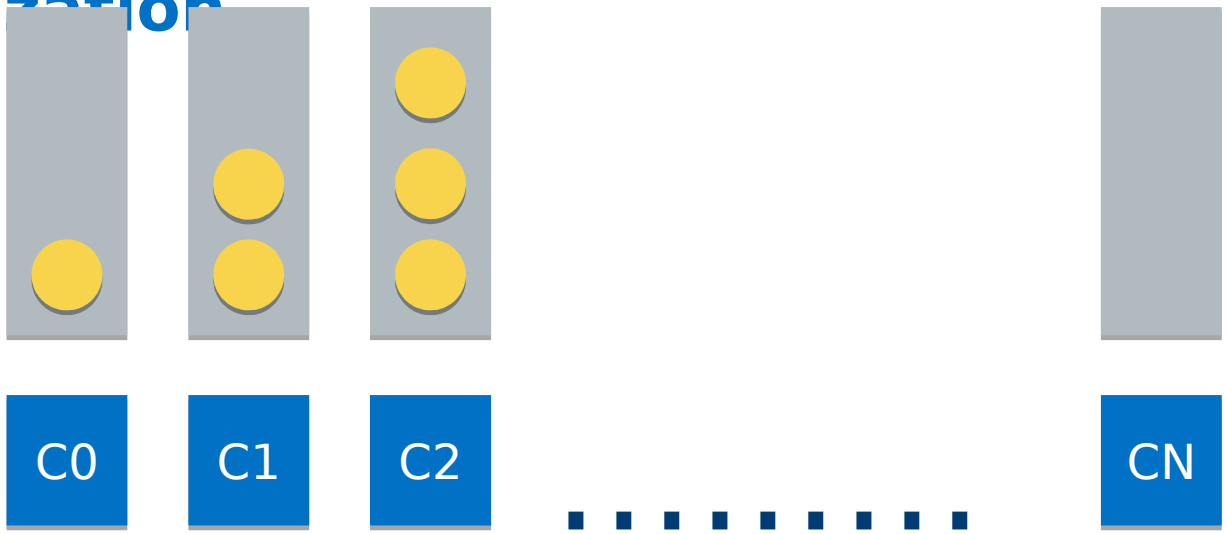
Multithreading – Tasking Systems

	Compiler Extension	Composable	Language Support	Offloading	Vectorization Support
OpenMP	✓	✗	C, C++, Fortran	✓	✓
Cilk+	✓	✓	C, C++	✗	✓
TBB	✗	✓	C++	✓*	✗*

Multithreading – Tasking Systems

- Thread: a unique stack of execution which may run in parallel
- Task: a unit of work (typically a function) to be run on a thread

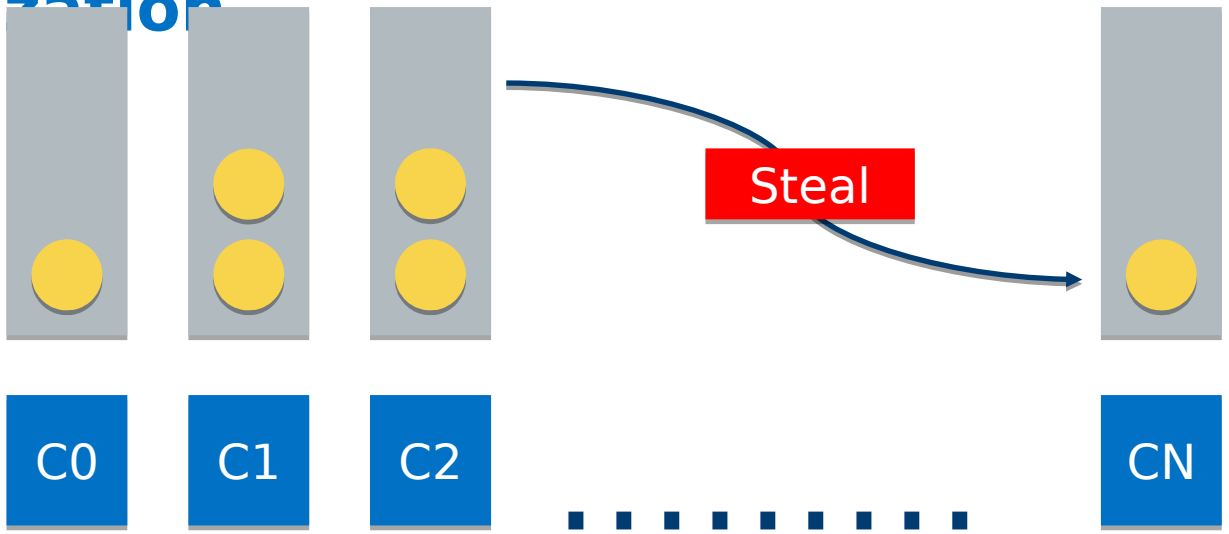
Tasks are scheduled on a pool of threads to optimize machine utilization



Multithreading – Tasking Systems

- Thread: a unique stack of execution which may run in parallel
- Task: a unit of work (typically a function) to be run on a thread

Tasks are scheduled on a pool of threads to optimize machine utilization



Multithreading – Abstracting “parallel_for()”

```
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

Multithreading – Abstracting “parallel_for()”

Original

```
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

OpenMP

```
#pragma omp parallel for
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

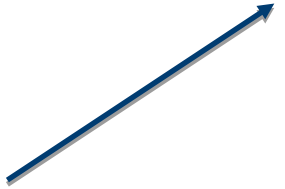
Multithreading – Abstracting “parallel_for()”

Original

```
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

OpenMP

```
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```



...to get default behavior like Cilk+ and TBB

Multithreading – Abstracting “parallel_for()”

Original

```
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

Cilk+

```
cilk_for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

Multithreading – Abstracting “parallel_for()”

Original

```
for (int i = 0; i < SIZE; ++i)
  computeIteration(i);
```

TBB

```
tbb::parallel_for(0, SIZE, [](int i) {
  computeIteration(i);
}
```

Multithreading – Abstracting “parallel_for()”

```
template <typename TASK_T>
inline void parallel_for (int nTasks, TASK_T&& fcn)
{
  // call the tasking system you want to use
}
```

- Templating over TASK_T just means it can be…
  - … a C-style function
  - … a C++ class which implements ‘operator()’
  - … a C++11 lambda function

Multithreading – Abstracting “parallel_for()”

```
template <typename TASK_T>
inline void parallel_for (int nTasks, TASK_T&& fcn)
{
  // call the tasking system you want to use
}
```

- Templating over TASK_T just means it can be…
  - … a C-style function
  - … a C++ class which implements ‘operator()’
  - … a C++11 lambda function

Multithreading – Abstracting “parallel_for()”

```
template <typename TASK_T>
inline void parallel_for (int nTasks, TASK_T&& fcn)
{
  // call the tasking system you want to use
}
```

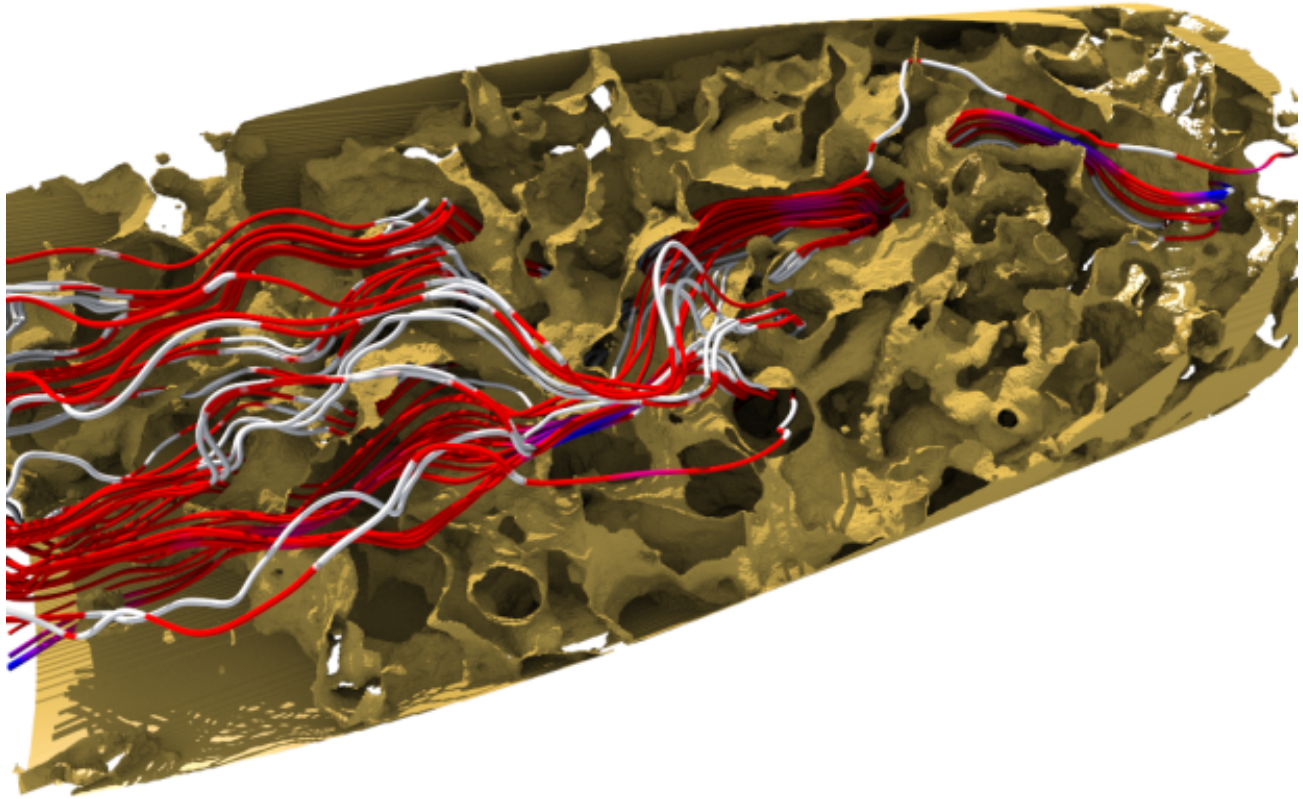
- Templating over TASK_T just means it can be…
  - … a C-style function
  - … a C++ class which implements ‘operator(
  - … a C++11 lambda function

More info on my blog:

http://jeffamstutz.io

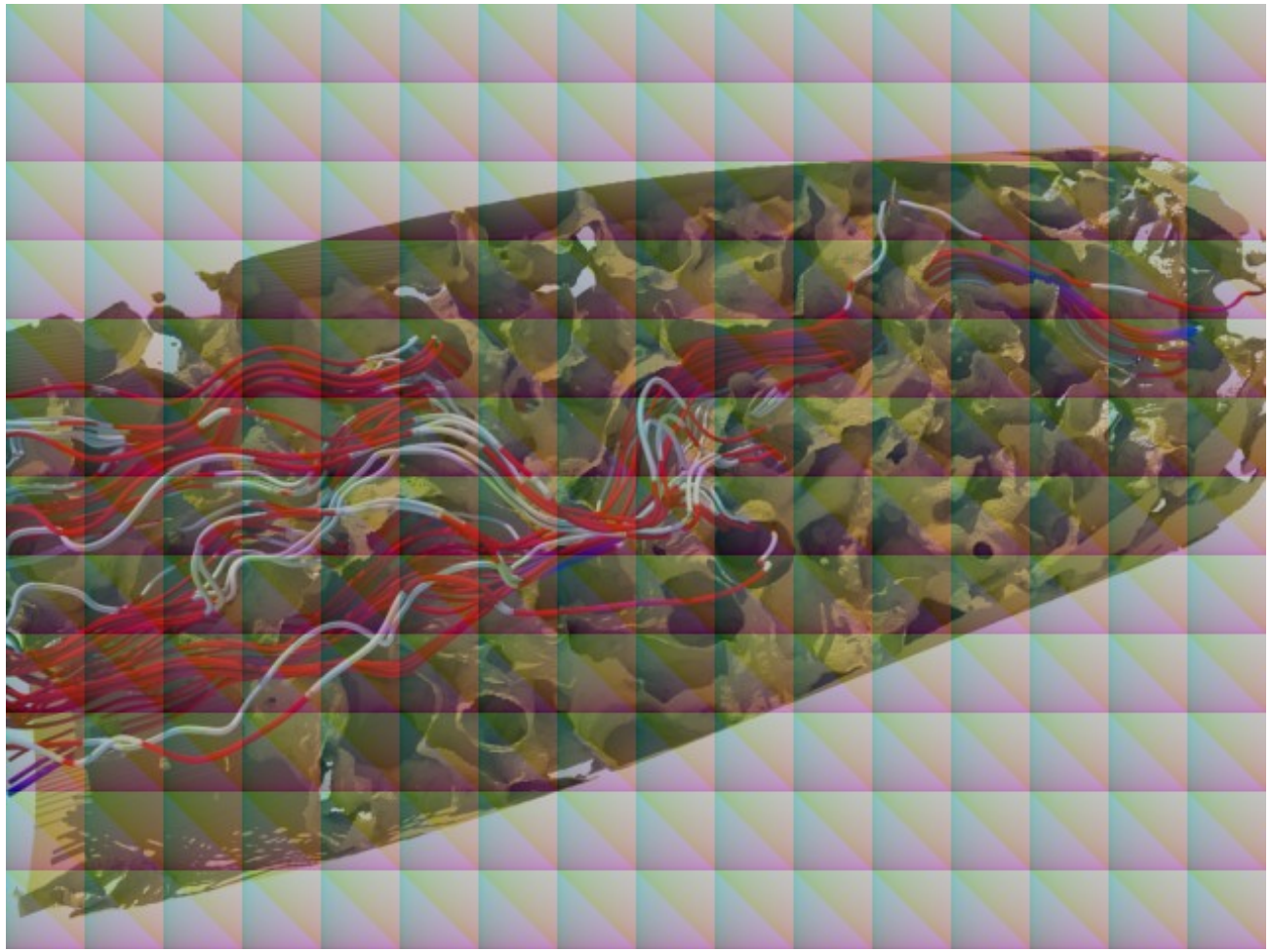
Multithreading – Using “parallel_for()”

- Each frame rendered as a collection of square tiles
- Each tile can be further subdivided into “packets” of pixels
- Nesting parallel_for() key to performance
  - Lots of tasks make it easier to balance work



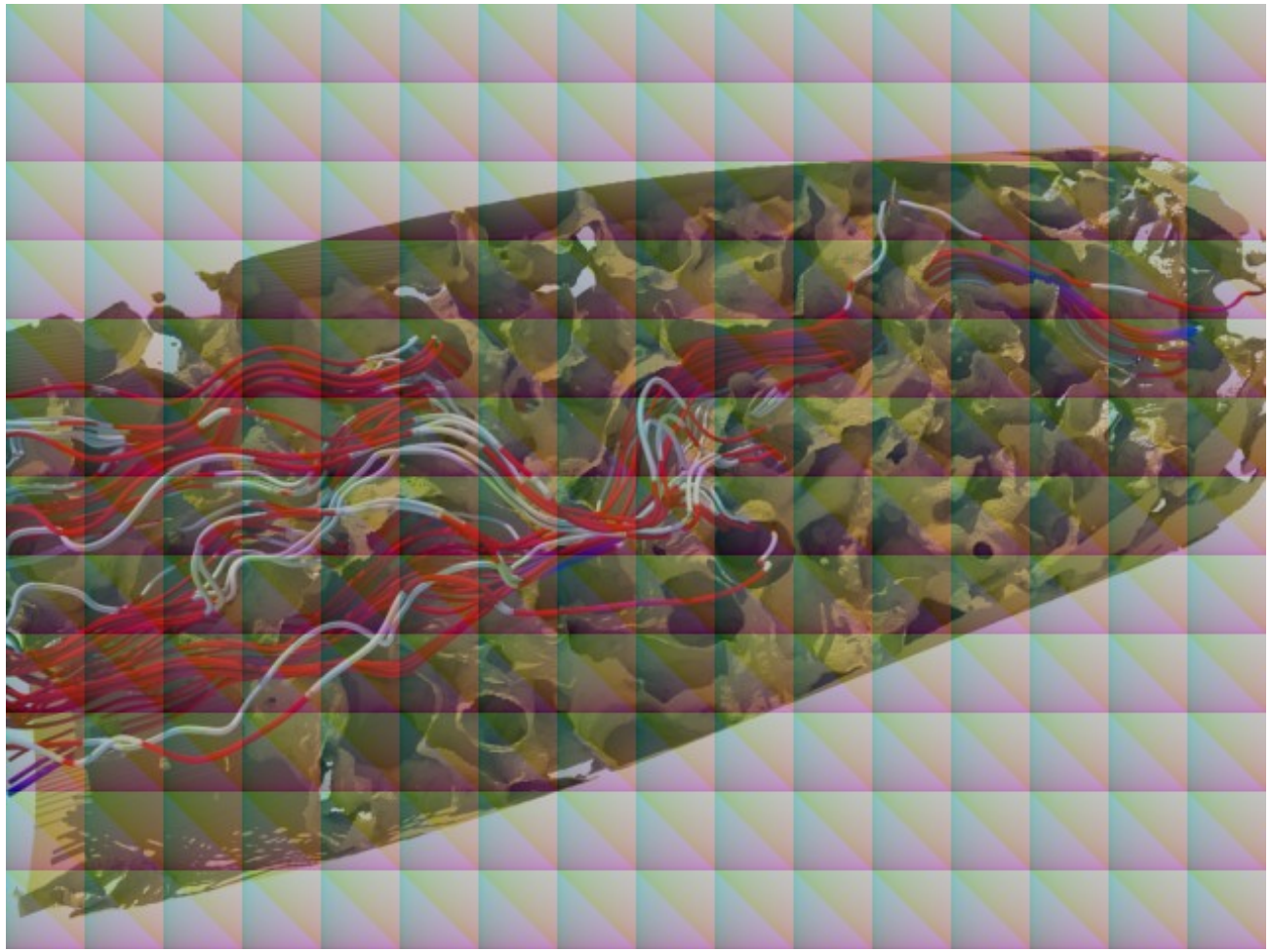
Multithreading – Using “parallel_for()”

- Each frame rendered as a collection of square tiles
- Each tile can be further subdivided into “packets” of pixels
- Nesting parallel_for() key to performance
  - Lots of tasks make it easier to balance work



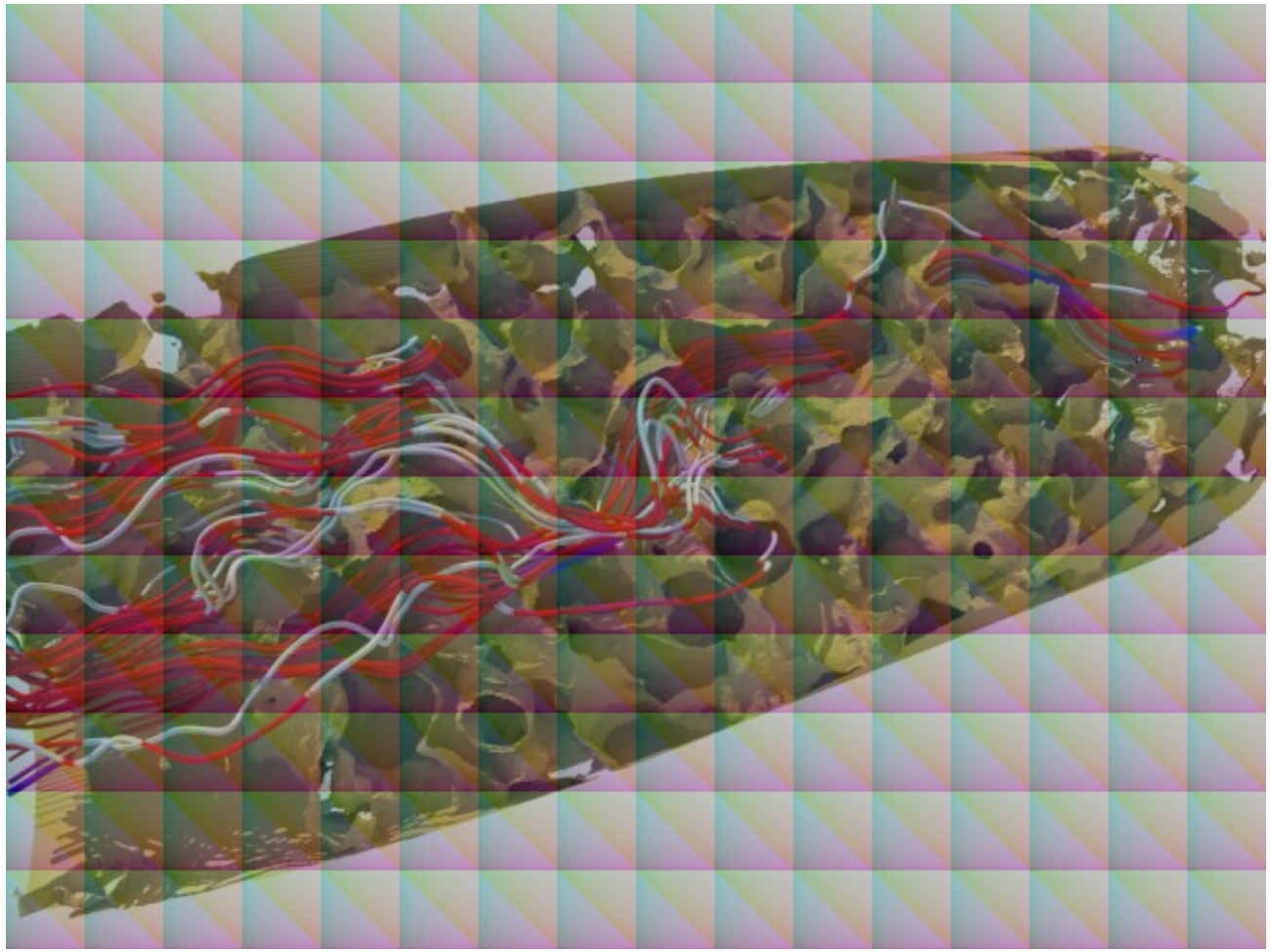
Multithreading – Using “parallel_for()”

- Each frame rendered as a collection of square tiles
- Each tile can be further subdivided into “packets” of pixels
- Nesting parallel_for() key to performance
  - Lots of tasks make it easier to balance work



Multithreading – Using “parallel_for()”

- Each frame rendered as a collection of square tiles
- Each tile can be further subdivided into “packets” of pixels
- Nesting parallel_for() key to performance
  - Lots of tasks make it easier to balance work



Multithreading – Using “parallel_for()”

```
void renderFrame(Renderer* r, FrameBuffer* fb)
{
  parallel_for(fb->numTiles(), [&](int tileID) {
    Tile tile;
    setupTile(tile, tileID);
    parallel_for(tile.numJobs(), [&](int jobID) {
      renderer->renderTile(tile, jobID);
    });
    fb->write(tile);
  });
}
```

Multithreading – Using “parallel_for()”

```
void renderFrame(Renderer* r, FrameBuffer* fb)
{
  parallel_for(fb->numTiles(), [&](int tileID) {
    Tile tile;
    setupTile(tile, tileID);
    parallel_for(tile.numJobs(), [
      renderer->renderTile(tile, j
    });
    fb->write(tile);
  });
}
```

For each tile in the framebuffer…

Multithreading – Using “parallel_for()”

```
void renderFrame(Renderer* r, FrameBuffer* fb)
{
  parallel_for(fb->numTiles(), [&](int tileID) {
    Tile tile;
    setupTile(tile, tileID);
    parallel_for(tile.numJobs(), [
      renderer->renderTile(tile, j
    });
    fb->write(tile);
  });
}
```

...create a Tile on the stack and initialize

Multithreading – Using “parallel_for()”

```
void renderFrame(Renderer* r, F
{
  parallel_for(fb->numTiles(),                    {
    Tile tile;
    setupTile(tile, tileID);
    parallel_for(tile.numJobs(), [&](int jobID) {
      renderer->renderTile(tile, jobID);
    });
    fb->write(tile);
  });
}
```

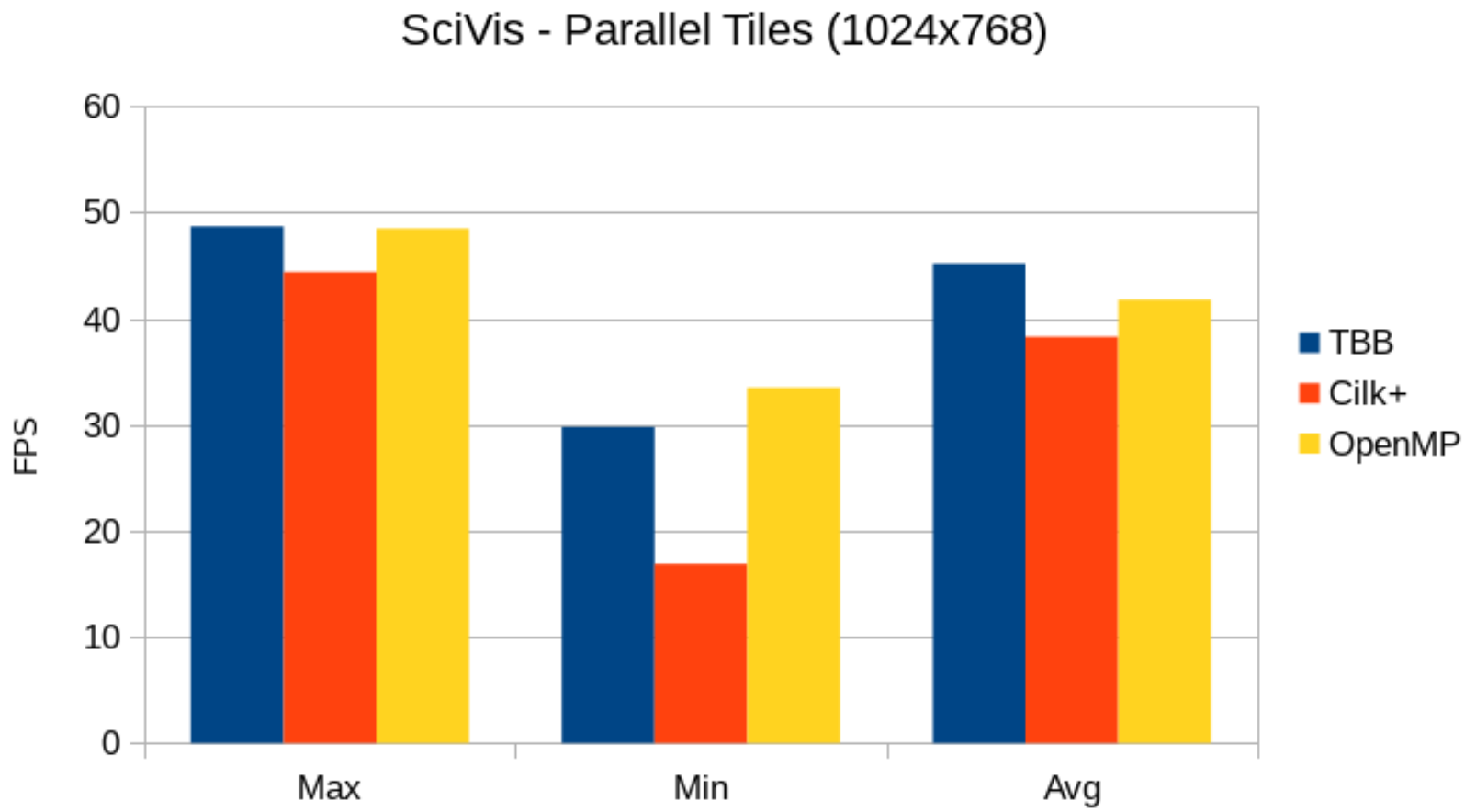
...render each subsection of that tile in parallel

Multithreading – Using “parallel_for()”

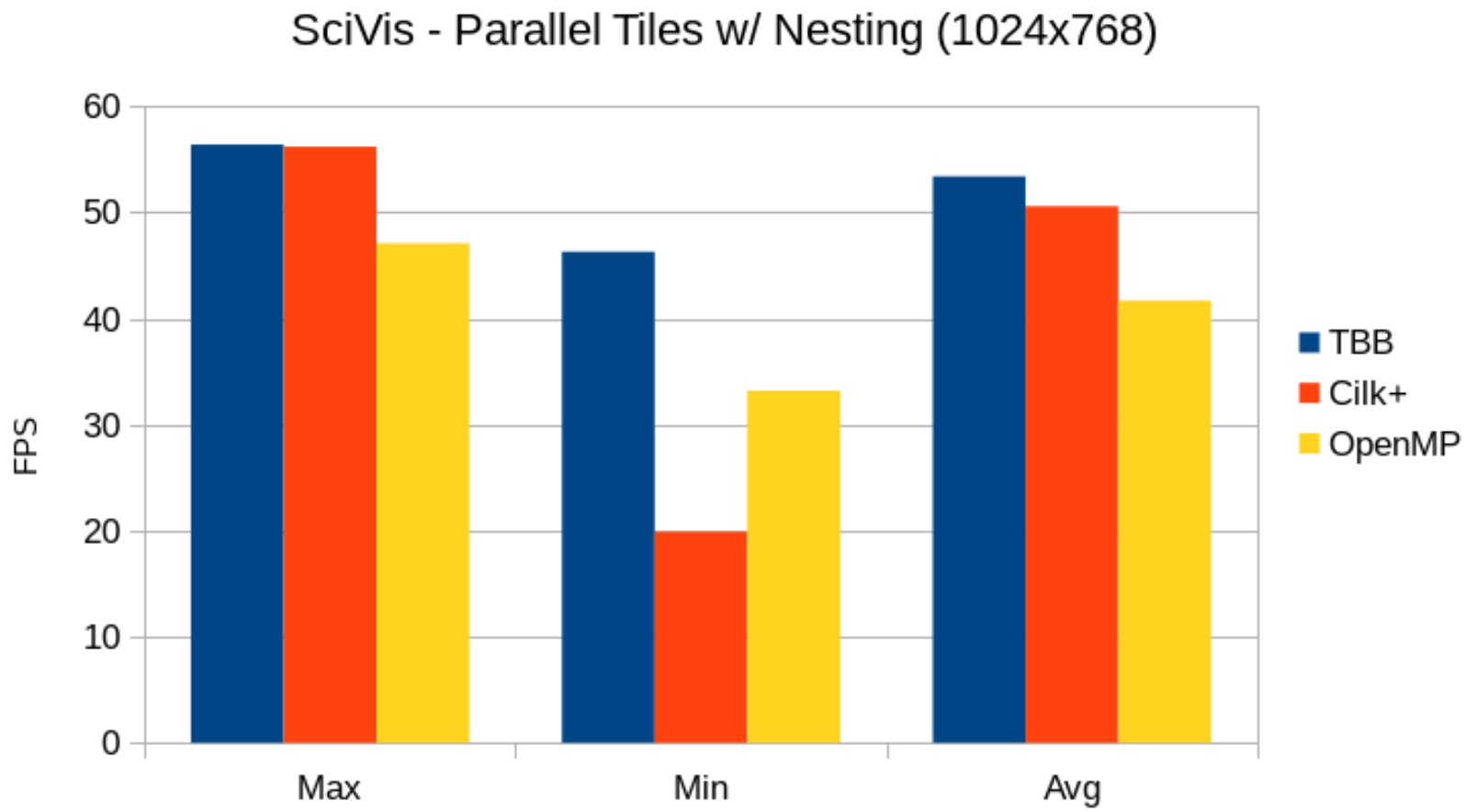
```
void renderFrame(Renderer* r, FrameBuffer* fb)
{
  parallel_for(fb->numTiles(), [&](int tileID) {
    Tile tile;
    setupTile(tile, tileID);
    parallel_for(tile.numJobs(), [&](int jobID) {
      renderer->renderTile(tile, jobID);
    });
    fb->write(tile);
  });
}
```

...finally write the tile back into the framebuffer

Multithreading – Abstracting “parallel_for()”



Multithreading – Abstracting “parallel_for()”

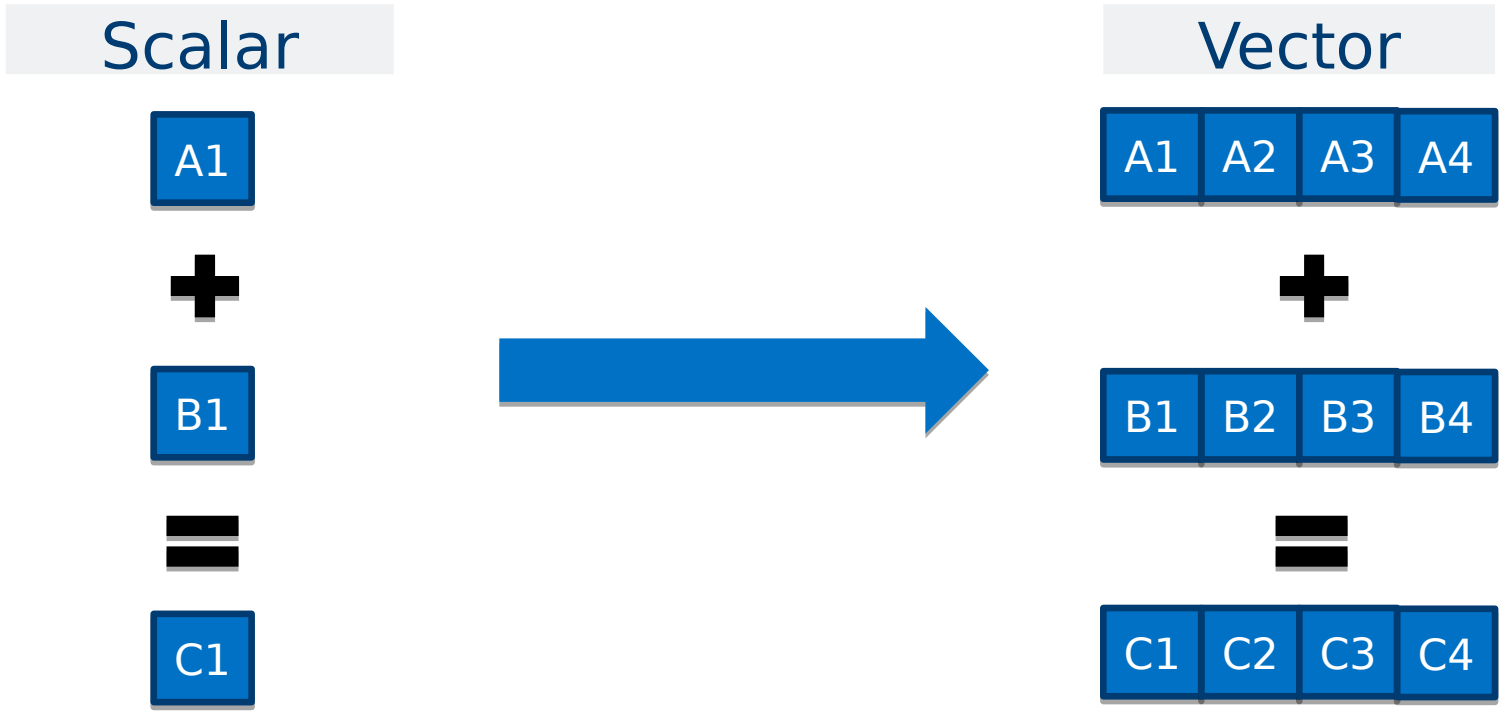


Multithreading – Tasking Systems

	Compiler Extension	Composable	Language Support	Offloading	Vectorization Support
OpenMP	✓	✗	C, C++, Fortran	✓	✓
Cilk+	✓	✓	C, C++	✗	✓
TBB	✗	✓	C++	✓*	✗*

Vectorization

- “Vectorized” code is code which uses SIMD instructions
  - **S**ingle **I**nstruction **M**ultiple **D**ata



Vectorization

```
void vector_add(float *a,
                float *b,
                float *c)
{
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

Vectorization

```
void vector_add(float *a,
                float *b,
                float *c)
{
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

LOOP:
1. LOAD a[i] → Ra
2. LOAD b[i] → Rb
3. ADD Ra, Rb → Rc
4. STORE Rc → c[i]
5. ADD i, 1 → i

Vectorization

LOOP:

1. LOAD a[i] → Ra
2. LOAD b[i] → Rb
3. ADD Ra, Rb → Rc
4. STORE Rc → c[i]
5. ADD i, 1 → i

LOOP:

1. LOADv4 a[i:i+3] → Rva
2. LOADv4 b[i:i+3] → Rvb
3. ADDv4 Rva, Rvb → Rvc
4. STOREv4 Rvc → c[i:i+3]
5. ADD i, 4 → i

Vectorization Options – “Autovectorization”

```
void vector_add(float *a,
                float *b,
                float *c)
{
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

Vectorization Options – “Autovectorization”

```
void vector_add(float *a,
                float *b,
                float *c)
{
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

Can’t vectorize because…

- Potential overlapping input arrays
- If ‘SIZE’ isn’t known at compile time or isn’t big enough
- Data alignment problems

Vectorization – Helping the compiler

- The biggest issue with getting code to vectorize is using the language to “rule out” code being incorrect if it is vectorized
- Things that can help narrow possibilities of incorrect code when vectorized:
  - constness
  - types which can’t be aliased
  - data alignment directives and declarations
  - data layout (structures) and access patterns

Vectorization Options – Compiler Directives

```
void vector_add(float *a,
                float *b,
                float *c)
{
#pragma omp simd
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

Vectorization Options – Compiler Directives

```
void vector_add(float *a,
                float *b,
                float *c)
{
#pragma omp simd
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

Tells the compiler, “ignore many of the things that you are concerned about and just vectorize the loop”

Vectorization Options – Compiler Directives

```
void vector_add(float *a,
                float *b,
                float *c,
                int   *cond)
{
#pragma omp simd
  for (int i = 0; i < SIZE; ++i) {
    if (cond[i])
      c[i] = a[i] + b[i];
  }
}
```

Vectorization Options – Compiler Directives

```
void vector_add(float *a,
                float *b,
                float *c,
                int   *cond)
{
#pragma omp simd
  for (int i ?????i < SIZE; ++i) {
    if (cond[i])
      c[i] = a[i] + b[i];
  }
}
```

Vectorization Options – SPMD

```
void vector_add(float *a, float *b,
                float *c, int   *cond) {
  for (int i = 0; i < SIZE / SIMD_WIDTH; ++i) {
    varying float va    = a[i:i+SIMD_WIDTH];
    varying float vb    = b[i:i+SIMD_WIDTH];
    varying float vc    = c[i:i+SIMD_WIDTH];
    varying float vcond = cond[i:i+SIMD_WIDTH];
    if (vcond)
      vc = va + vb;
  }
}
```

Vectorization Options – SPMD

```
void vector_add(const varying float &a,
                const varying float &b,
                      varying float &c,
                const varying int   cond)
{
  if (cond[i])
    c = a + b;
}
```

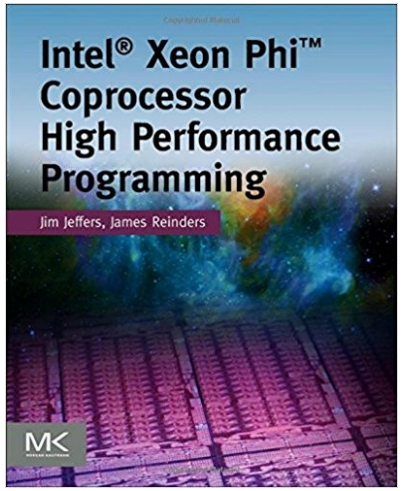
Vectorization Options – SPMD

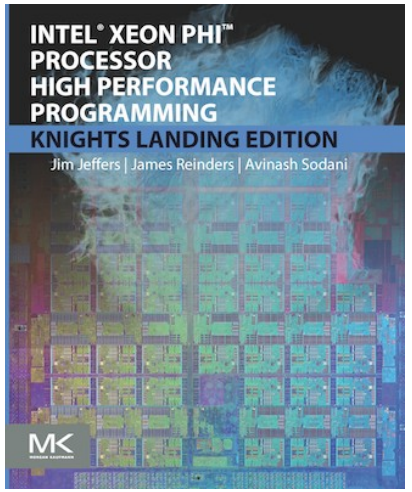
```
void vector_add(const varying float &a,
                const varying float &b,
                      varying float &c,
                const varying int   cond)
{
  if (cond[i])
    c = a + b;
}
```

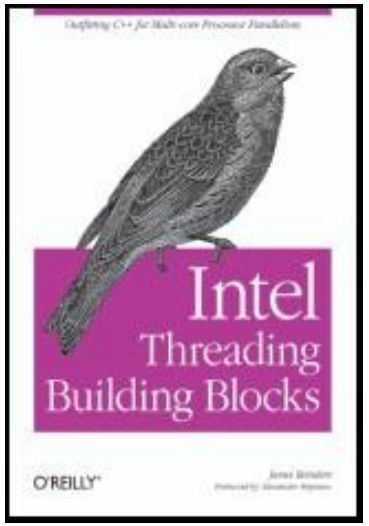
Compiler generates execution mask to only apply operations to ”active” SIMD lanes

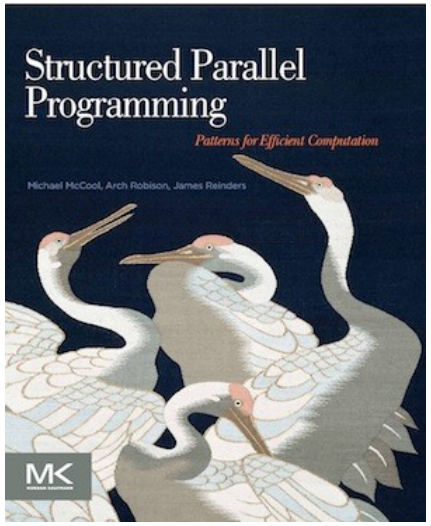
Exploiting Parallelism - Resources

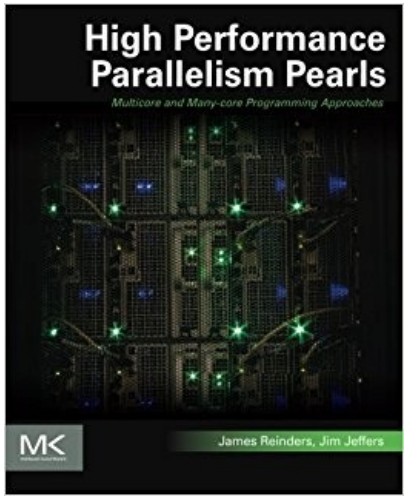
- Lots of resources! Books, talks, tutorials, classes, etc.

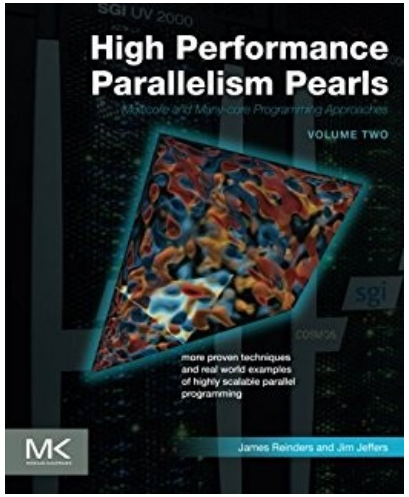












Exploiting Parallelism - Resources

Some of my favorite talks on parallelism:

- Sean Parent (Adobe) – https://youtu.be/QIHy8pXbneI
- James Reinders (Formerly Intel) – https://youtu.be/JpVZww1hL4c
- Hartmut Kaiser (LSU) – https://youtu.be/4OCUEgSNIAY
- …and many more!

Exploiting Parallelism - Resources

- Effective vectorization options:
  - ISPC, OpenMP SIMD, Cilk+ extensions, boost.simd (library), …
- Effective multithreading options:
  - TBB, Cilk+, OpenMP, Qthreads (SNL), libdispatch, PPL, …
- “Whole system” scaling libraries (including multi-node scaling)
  - HPX, Kokkos, RAJA, Charm++, …

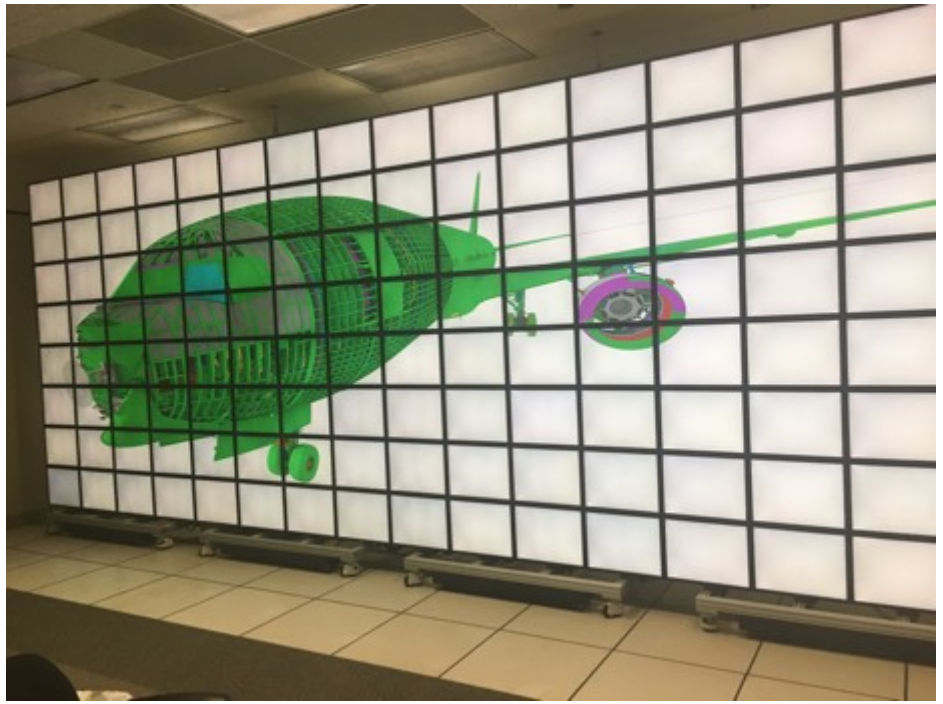
Exploiting Parallelism - Resources

- Effective vectorization options:
  - ISPC, OpenMP SIMD, Cilk+ ext
- Effective multithreading options
  - TBB, Cilk+, OpenMP, Qthreads
- “Whole system” scaling libraries
  - HPX, Kokkos, RAJA, Charm++,

Learn about each option and pick what will work best given your constraints

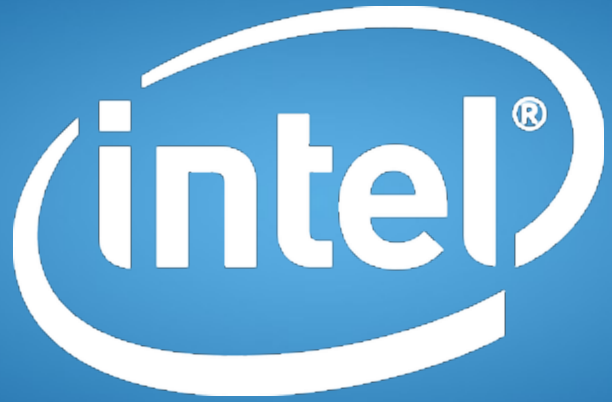
- Am I writing new code or enhancing an existing implementation?
- Do I care more about very low-level control or productivity?
- How much time do I have to learn another technology?
- What languages am I comfortable with?

Simulation: Jeff Onufer and Tom Pulliam, NASA Ames
Visualization: Tim Sandstrom and Pat Moran, NASA Ames



THANK YOU!

Q & A?


intel®