SAND2022-1599C

Data-Parallel Primitives for Minimizing Many-core Development Cost

Mark Bolstad

February 17, 2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Acknowledgements

- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Numbers 10-014707, 12-015215, and 14-017566.
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.
- Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.
- Thanks to many, many partners in labs, universities, and industry.

VTK-m

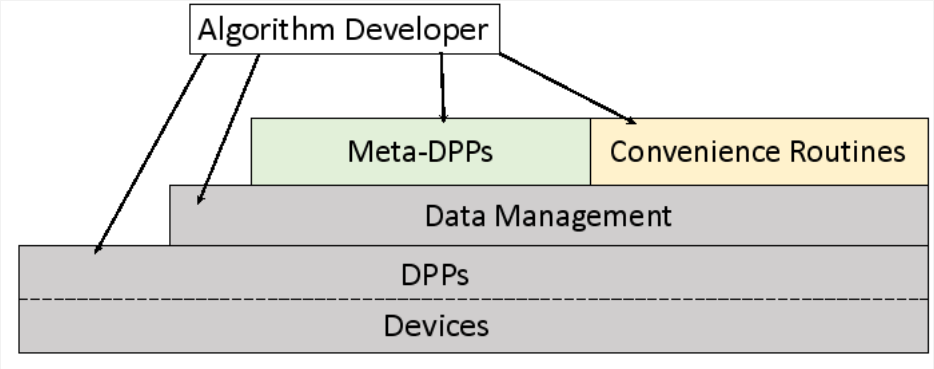
- Scientific Visualization has a rich history with GPUs
  - Rending, but also algorithms and transformations
- In 2010, three development teams focused on portable performance:
  - EAVL (data models)
  - DAX (execution models)
  - PISTON (algorithms)
- Three teams merge to form VTK-m
  - Teams at Oak Ridge, Kitware, Los Alamos, OSU, and Sandia
- Open source, tutorial materials available (VIS19), public github
- Key publications:
  - K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K.- L. Ma, H. Childs, M. Larsen, C.-M. Chen, R. Maynard, and B. Geveci. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. IEEE Computer Graphics and Applications (CG&A), 36(3):48–58, May/June 2016.
  - K. Moreland, R. Maynard, D. Pugmire, A. Yenpure, A. Vacanti, M. Larsen, and H. Childs. Minimizing Development Costs for Efficient Many-Core Visualization Using MCD3. Parallel Computing, 108:102834, Dec. 2021.

Data Parallel Primitives (DPP)

- Origins: Guy Blelloch Ph.D dissertation(1990), “Vector Models for Data-parallel Computing”
- What are they?
  - Assume you have N elements in an array
  - Assume there are >= N cores
  - A DPP must complete in O(log N) time
- Examples: Map, Reduce, Scan, Gather, Scatter
- Why?
  - Performant algorithms
  - Increased reliability
  - Hardware agnostic
- AMP, BOLT, Boost.Compute, Thrust, and recent additions to C++ standard implement DPPs

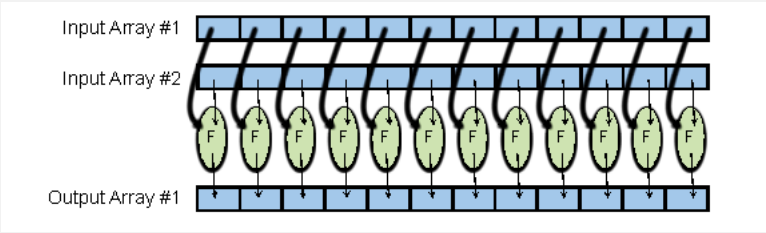
MCD^3

- **Meta-DPPs**: which are parallel processing patterns that involve one or more DPPs. The word choice of "meta" is meant to evoke its definition of "denoting something of a higher or second-order kind."
- **Convenience routines:** which encapsulate common operations for scientific visualization.
- **DPPs:** which provide parallel processing patterns.
- **Data management:** which insulates algorithms from data layout complexities. These complexities range from how data is organized (e.g., structure-of-arrays vs array-of-structures) to different types of meshes (e.g., unstructured, rectilinear, etc.) to different memory spaces (e.g., host memory, device memory, or unified managed memory)
- **Devices:** which enable code to run on a given hardware architecture.

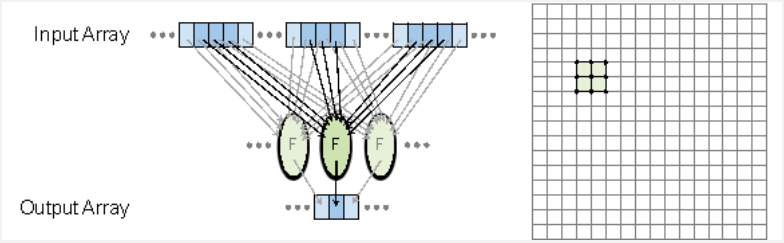


5 Meta-DPPs + Modifiers*

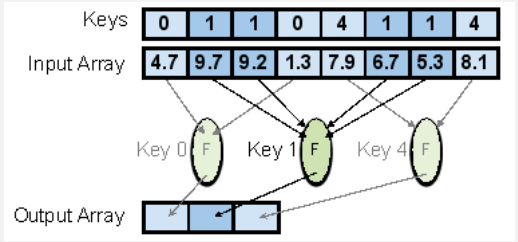
Map Field



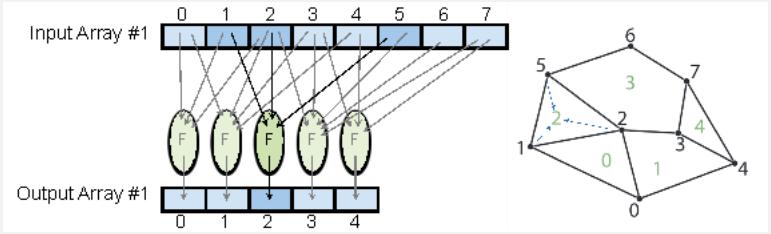
Point Neighborhood



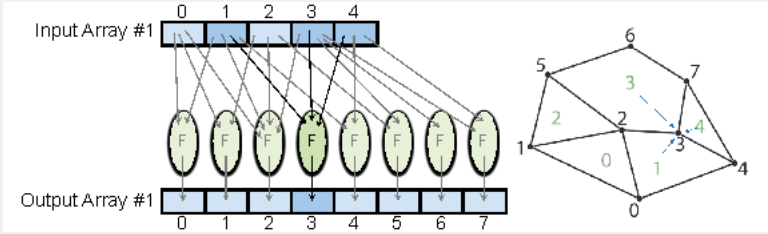
Reduce By Key



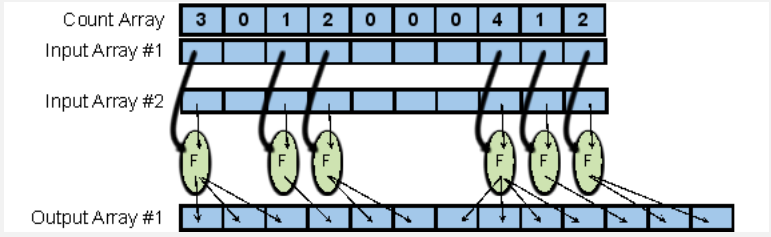
Visit Cell with Points

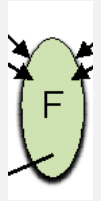


Visit Point with Cells



Scatter*





Functor

Convenience Routines

- **ArrayRange:** Given an array, ArrayRange finds the minimum and maximum value in that array.
- **CountToOffset:** Because visualization algorithms often deal with jagged data, it is common to need to pack items of different sizes into a larger array. Often an algorithm will start with a count of how many components are with each group (e.g. a count of how many vertices are in each cell of an unstructured grid such as the first has 8 vertices, the second has 4 vertices, etc.). countToOffset will efficiently compute the necessary offsets from the counts.
- **Locators:** An algorithm sometimes needs to identify which cell in a mesh contains a point at a given coordinate. For irregular meshes, finding these cells efficiently requires special search structures.
- **MapFieldMergeAverage:** Visualization algorithms sometimes need to merge elements together. Often this is a simplification of a mesh with elements that are coincident or that can be combined with minimal error. When elements are merged, the fields on the elements need to be combined in some way.
- **MapFieldPermutation**: Many visualization algorithms modify the structure of a mesh and need to pass data according to the modifications. For example, a threshold algorithm will remove cells from the mesh. MapFieldPermutation can reorder the cell fields on the input mesh to match the new cell ordering of the output mesh.

Data-Parallel Primitives (DPPs)

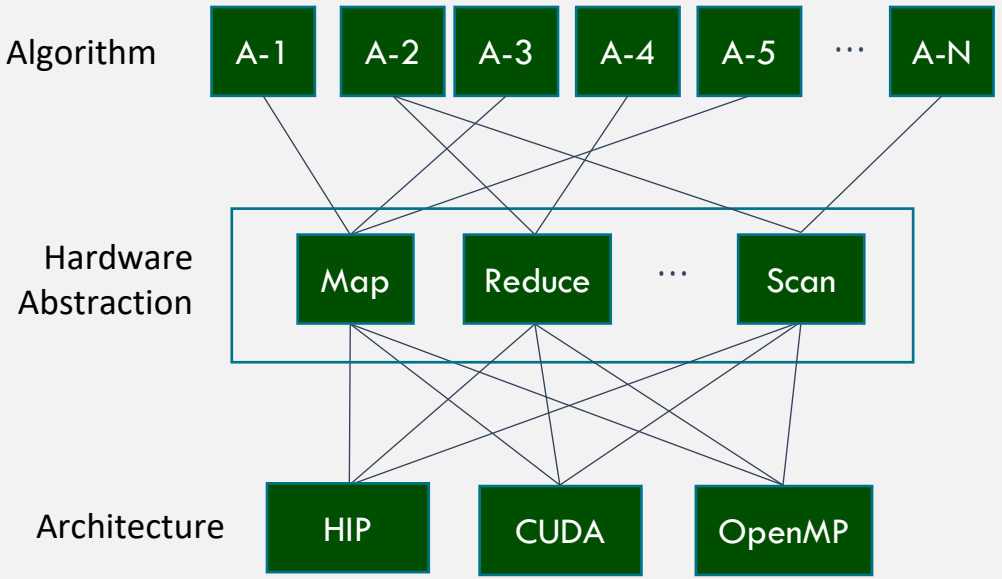
- Currently used DPPs in VTK-m
  - Copy
  - CopyIf
  - CopySubRange
  - CountSetBits
  - Fill
  - LowerBounds
  - Reduce
  - ReduceByKey
  - ScanInclusive
  - ScanInclusiveByKey
  - ScanExclusive
  - ScanExclusiveByKey
  - ScanExtended
  - Sort
  - SortByKey
  - Synchronize
  - Transform
  - Unique
  - UpperBounds

Data Management (DM)

- The DM layer decouples memory layout from execution
  - Insulates meta-DPP and convenience routines from data layout and reorganization issues
  - It enables a single code base to support many data layouts
    - For a *functor*, data is always points, cells, …
    - DM layer reorganizes prior to execution
- Reorganization
  - AOS vs SOA
- Data Layout
  - Isolating cells from structured/unstructured meshes
- In-situ processing is a major motivator

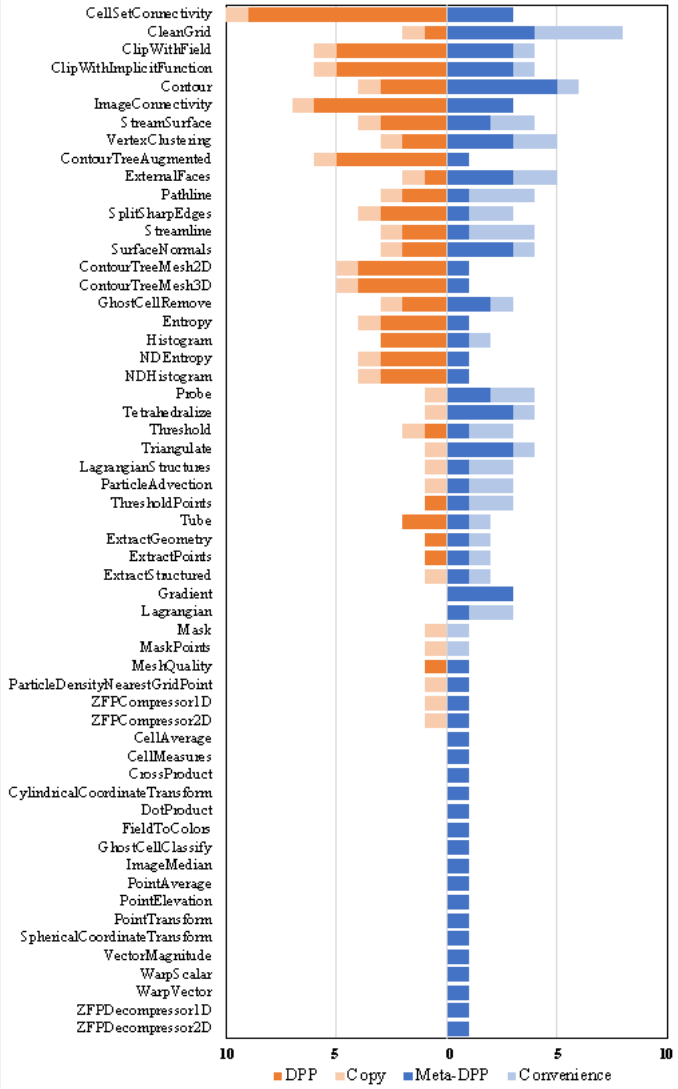
Devices

- Represents the physical hardware
  - Assumes compiler for each device supports C++
  - Extended language features are hidden from developers
- DM and DPPs work to unify interface to devices
- DM layer prepares data for use on devices
  - Copying data for devices with separate memory spaces
  - Allocating memory for managed uniform memory
- Each device has a unique implementation for the DPPs



Reducing Development Costs

- Assertion: Meta-DPP Reuse → Reduced development cost
  - VTK-m has 57 visualization algorithms
    - 85 meta-DPPs, 44 convenience routines, 32 Copy DPPs, 78 non-Copy DPPs
    - 97% use at least 1 meta-DPP, 47% a convenience routine, 56% a Copy DPP, 47% a non-Copy DPP
  - On average, an algorithm uses
    - 1.5 meta-DPPs
    - 0.8 convenience routines
    - 0.6 Copy DPPs
    - 1.4 non-Copy DPPs
- Estimating developer cost is difficult, our methodology is necessarily an approximation



Methodology

- $Uses(A, X)$ is 1 if A uses X, otherwise 0
- $Equiv(X)$ is the cost to implement X in a DPP-only system
- Cost Functions

$$C_{MCD^3}(A) = \sum_{m \epsilon M} Uses(A, m) + \sum_{d \epsilon D} Uses(A, d)$$

$$C_{DPP-Only}(A) = \sum_{m \epsilon M} Uses(A, m) * Equiv(m) + \sum_{c \epsilon C} Uses(A, c) * Equiv(c) + \sum_{d \epsilon D} Uses(A, d)$$

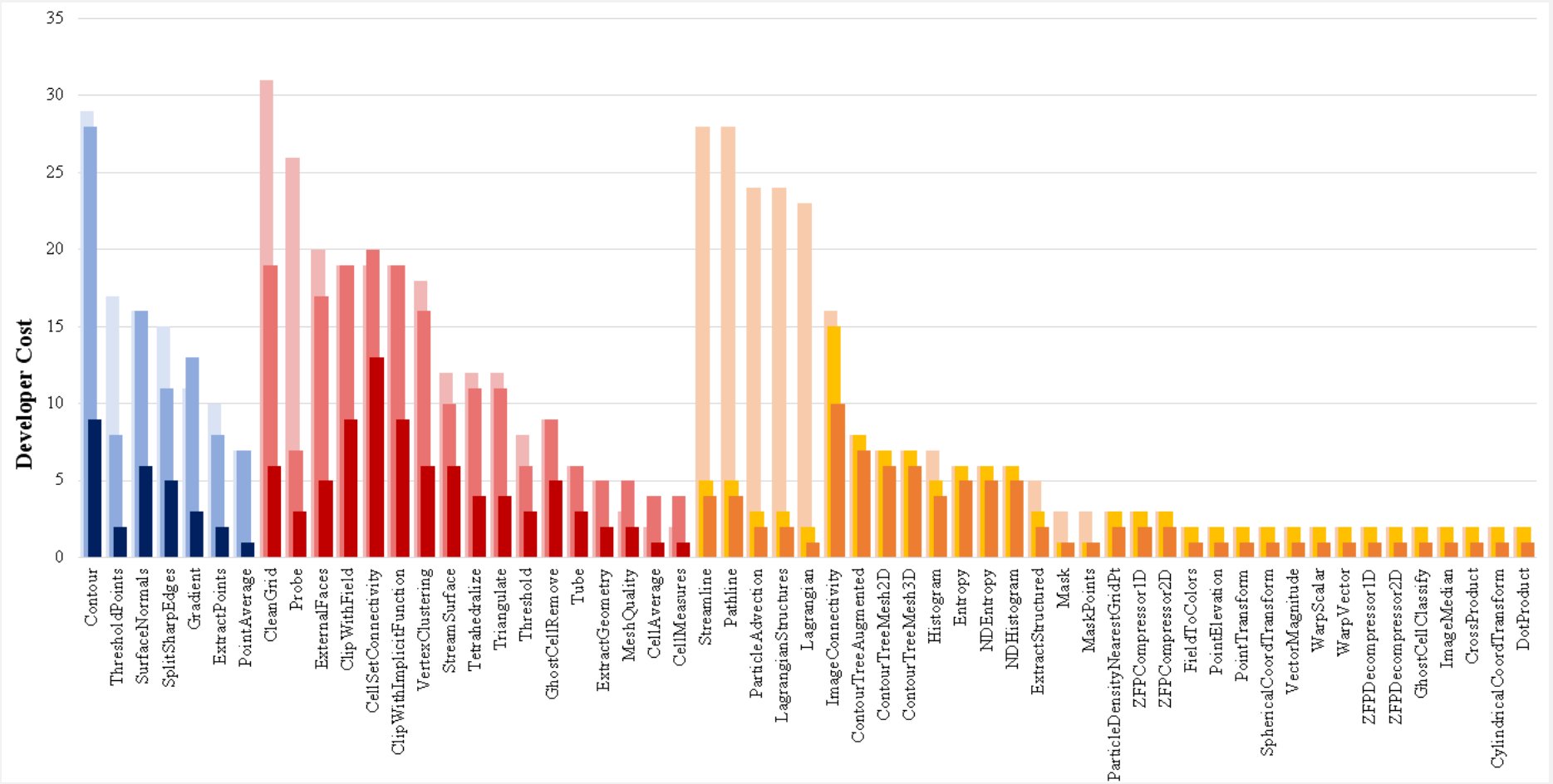
$$C_{DPP+Conv}(A) = \sum_{m \epsilon M} Uses(A, m) * Equiv(m) + \sum_{d \epsilon D} Uses(A, d)$$

- Results (Implementation Costs)
  - $C_{MCD^3} = 195$
  - $C_{DPP-Only} = 605$
  - $C_{DPP+Conv} = 399$

	Feature	# DPPs
meta-DPP	Visit Point With Cells	6
	Reduce By Key	6
	Scatter Counting	4
	Visit Cell With Points	1
	Point Neighborhood	1
	Map Field	1
Convenience	Locator	18
	MapFieldMergeAverage	6
	FieldMapPermutation	1
	CountToOffset	1
	ArrayRange	1

Algorithm Uses	Savings Factor DPP-Only	Savings Factor DPP + Conv
Visit Points with Cells	4.1X	3.5X
Visit Cells with Points	3.2X	2.3X
All Other	2.7X	1.4X
Other: Particle Advection	8.8X	1.4X
Other: Non-Particle Advection	1.6X	1.4X

Results



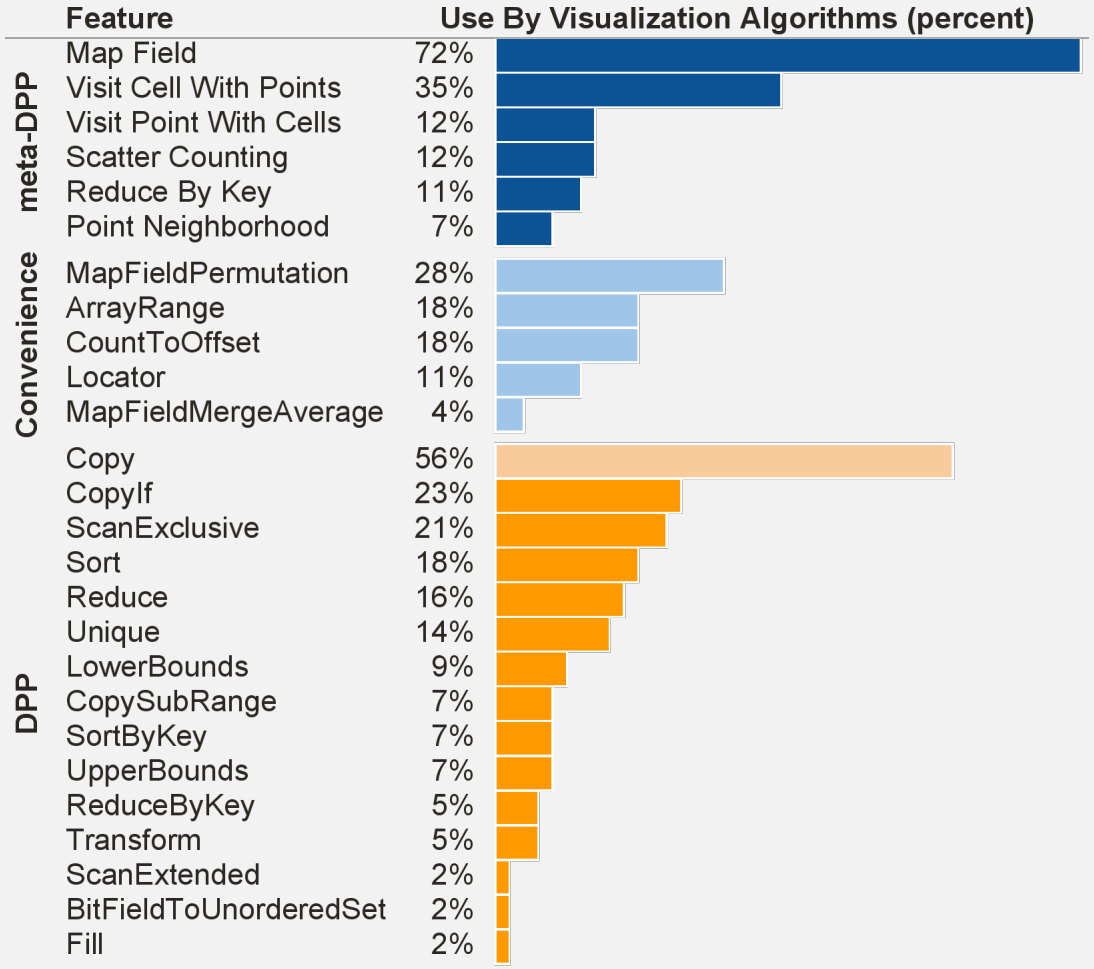
Legend

- Visit Point with Cells
- Visit Cell with Points
- Other

- C_{MCD^3}
- $C_{DPP+Conv}$
- $C_{DPP-Only}$

Efficacy

- Is MCD^3 overly complicated?
  - No, see chart
- Is MCD^3 too simple?
  - Main visualization library behind Ascent
    - Used by 12+ computational sim teams
    - ➢ Our *57* algorithms provide a sufficient feature set
- Originally designed with 20 core algorithms
  - All implemented
  - Does not exhaustively cover the space of possible visualization algorithms
    - Future algorithms may be difficult to fit within the framework



Performance

- MCD^3 vs Hardware-Specific
  - Mix of rendering/visualization software
    - Embree, HAVS, OptiX, VisIt, VTK, Vapor
  - And direct implementations
    - CUDA, OpenMP, pthreads, TBB, Thrust
- Two algorithms had significantly worse performance
  - External facelist
    - Essentially a serial algorithm
  - Ray Tracing
    - OptiX and Embree are just stinkin' good

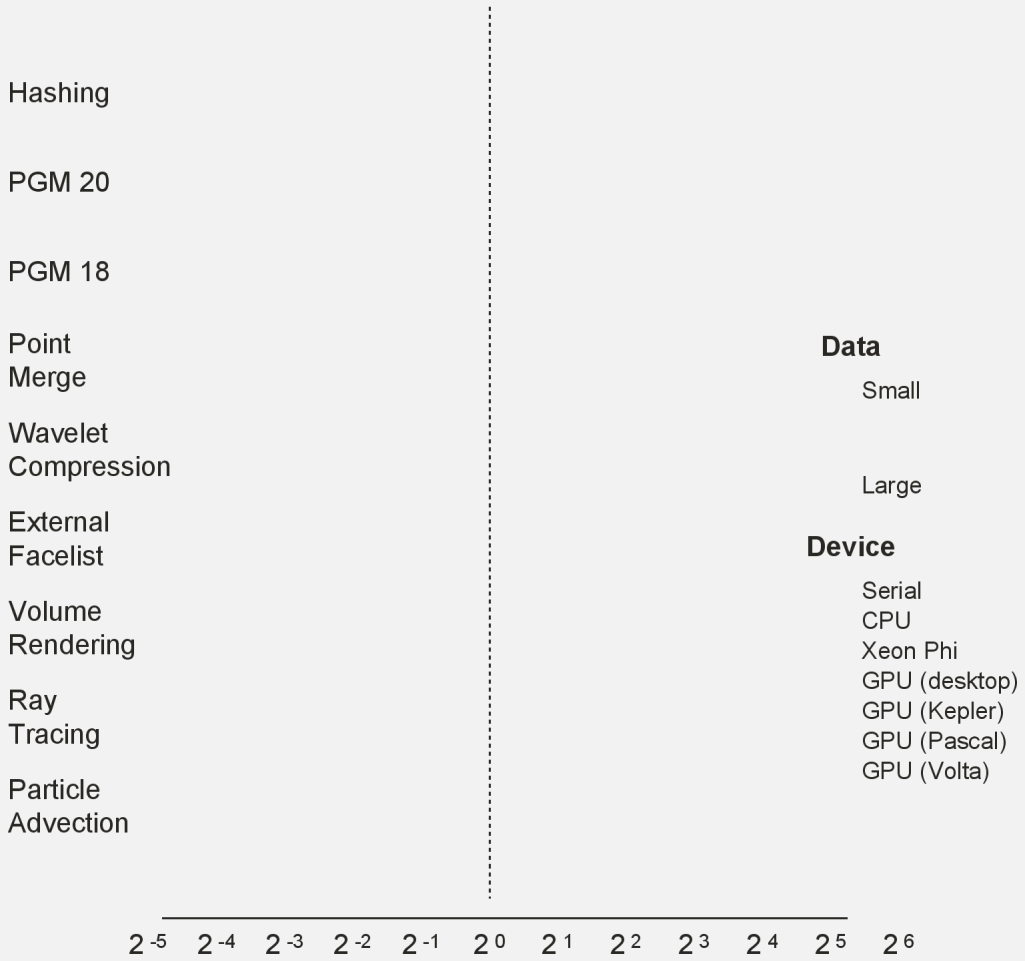
Algorithm	Architecture	Comp.	Perf.
Ray Tracing [45]	I I7 4770K	Embree	0.28-0.48
	I Ivy Bridge	Embree	0.4-0.58
	N GTX Titan Blk	OptiX	0.44-0.56
	N Tesla K80M	OptiX	0.37-0.51
	N GeForce 750Ti	OptiX	0.69-0.89
	N GeForce 620M	OptiX	0.73-1.16
Volume Rendering [46]	I Ivy Bridge (1)	VisIt	0.73-9.1
	I I7 4770K (8)	VTK	2
	N GTX Titan Blk	HAVS	0.33-2
External facelist [47]	I Ivy Bridge (1)	VTK	0.50-1.4
	I Ivy Bridge (1)	VisIt	0.08-0.25
Wavelet Compression [48]	I Haswell (16)	Vapor	0.8-1.5
	N Tesla K40	CUDA	0.6-0.8
Particle Advection [49]	I Ivy Bridge (16)	VisIt	2.4-3.6
	I Haswell (28)	pthreads	0.03-1.6
	IB Power8 (20)	pthreads	0.05-1.03
	N Tesla K20x	CUDA	0.37-2.26
	N Tesla K80	CUDA	0.54-2.45
	N Tesla P100	CUDA	0.48-4.08
Point Merge [39]	IB Power9 (1)	VTK	1.07-6.86
	IB Power9 (40)	VTK	1.4-2.5
	N Tesla V100	VTK-m	0.48-4.0
Probabalistic Graphical Modeling (PGM) 2018 [50]			
	I Ivy Bridge (24)	OpenMP	2-7
	I Phi 7250 (68)	OpenMP	0.75-4.25
PGM 2020 [51]	I Ivy Bridge (8)	{OpenMP,	2.2, 2.6
	I Xeon Phi 7250	pthreads}	0.04, 1.58
Hashing [52]	I Skylake (32)	{TBB,	1.2-37
	N Tesla K40	CUDPP,	0.09-13
	N Tesla V100	Thrust}	0.27-5.96

* References in chart are from the original MCD^3 paper in Parallel Computing, 108:102834, Dec. 2021

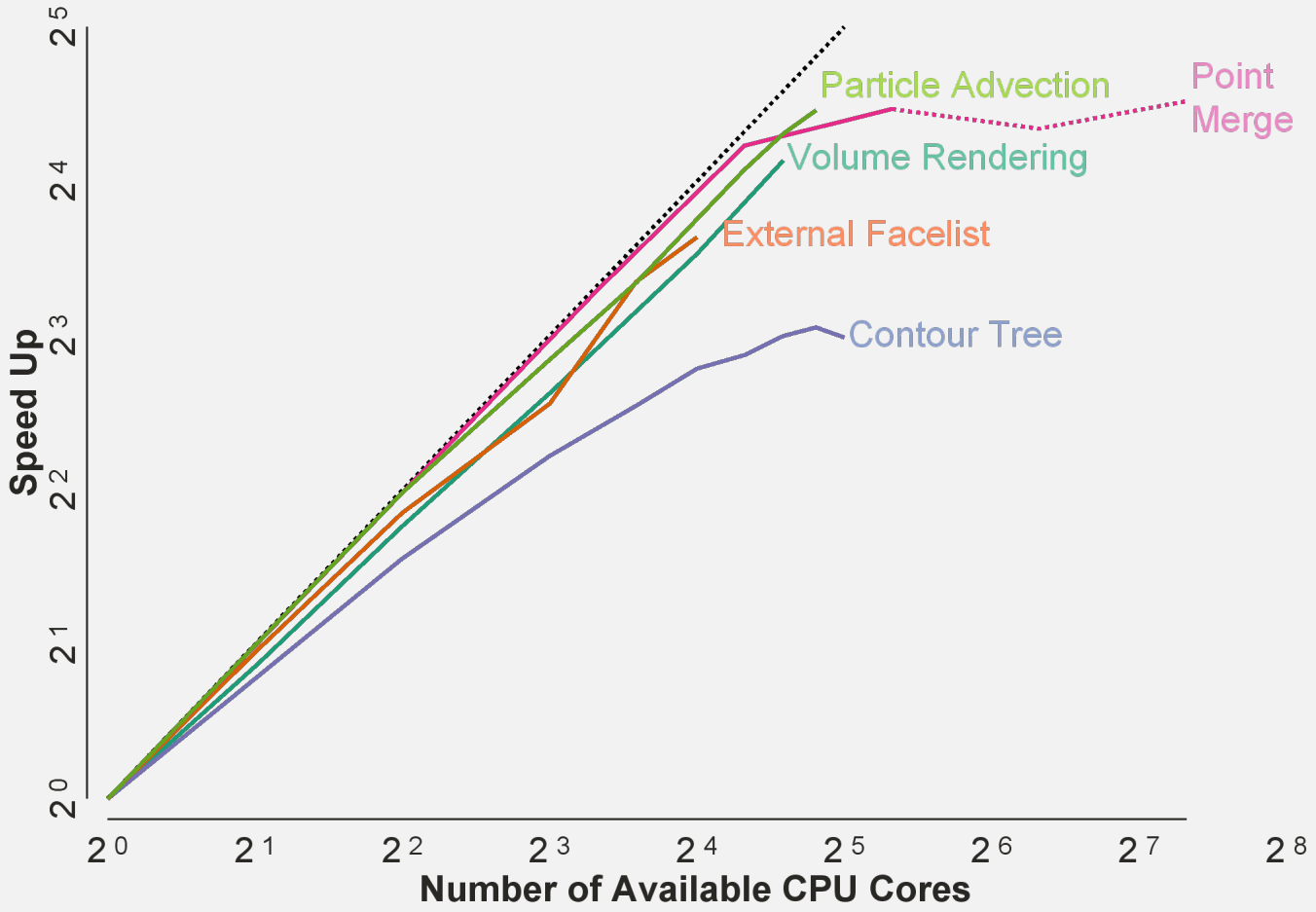
Performance (Detailed)

- MCD^3 vs Hardware-Specific
  - Horizontal axis is the ratio in MCD^3 performance against it's comparator
    - 2^{-1} ➔ MCD^3 took twice as long
    - 2^{1} ➔ MCD^3 took half the time

Algorithm	CPUs	GPUs	X. Phi	Serial	Total
External facelist	-	-	-	0.34	0.34
PGM 18	3.32	-	0.87	-	1.69
PGM 20	2.39	-	0.25	-	0.78
Particle advection	0.38	1.53	-	-	0.76
Point merge	1.82	-	-	3.10	2.38
Ray tracing	0.47	0.55	-	-	0.51
Volume rendering	1.13	0.83	-	3.10	1.43
Wavelet compression	1.13	0.75	-	-	0.92
Hashing	5.97	1.45	-	-	2.94
Total	1.45	0.95	0.47	1.48	**1.14**



Performance (MCD³ Scaling on Multi-Core CPUs)



Algorithm	Architecture	Max Cores	Parallel Efficiency
Volume Rendering [46]	I Ivy Bridge	24	0.73
External Facelist [47]	I Ivy Bridge	16	0.77
Contour Tree [57]	I Sandy Bridge	32	0.24
Point Merge [39]	IB Power 9	40	0.55
Particle Advection [49]	I Haswell	28	0.78

Conclusion

- Results demonstrate efficacy of MCD^3 in minimizing developer time while achieving portable performance on multi-core architectures
- 3.1x for developer efficiency
  - An approximation, but high enough to indicate savings for the VTK-m development
- Overall 1.14x faster than hardware-specific implementations
  - Somewhat surprising to the team
    - Ideal was 1.0x, expected 0.8x - 0.9x
    - 0.95x for GPUs is compelling
- Underlying system is time-consuming to implement
  - Heavy use of template meta-programming increases barrier to developers, increased compile time