Communication and Task Mapping Framework for Heterogeneous Systems

James Brock

Miriam Leeser

Mark Niedre

September 15, 2010

Outline

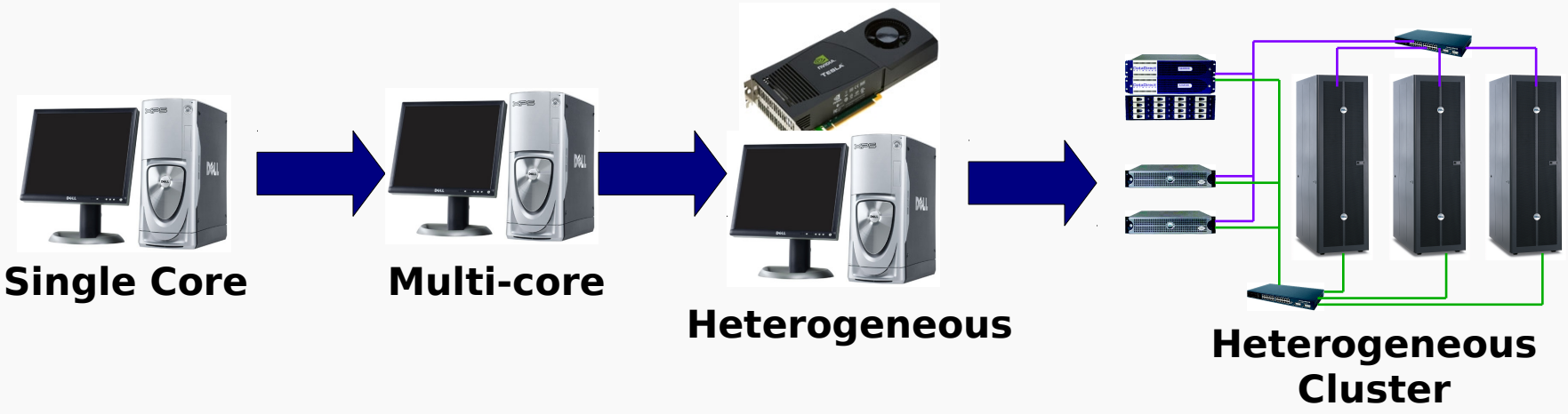
- Motivation
- PVTOL Tasks and Conduits
- PVTOL GPU Tasks and Conduits
- Fluorescence Mediated Tomography
- Current Results
- Future Work

Motivation

- Heterogeneous systems consisting of a wide variety of processing units are now common
  - Single-core, multi-core, GPU, etc.
- Each type of processing element has different characteristics
  - More adept at different types of work
  - Different core speed
- Parallel Vector Tile Optimizing Library (PVTOL) provides a means of writing high-performance, portable applications
- PVTOL Tasks and Conduits framework is the basis for exploiting data and task parallelism

Motivation

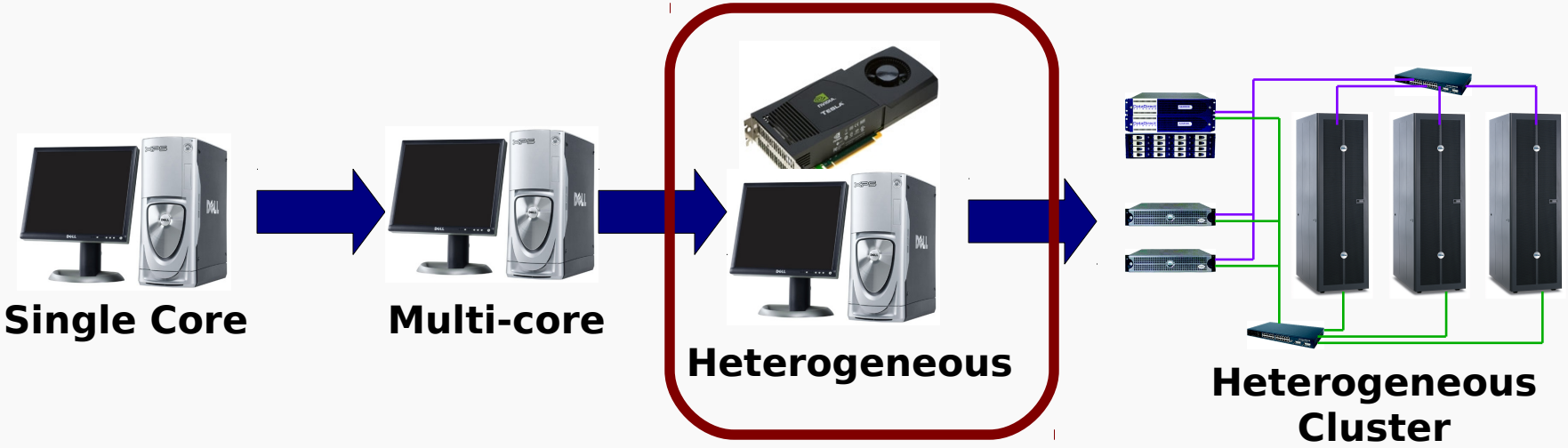
- Minimal programmer effort to change application platforms
- Enable rapid high-performance application development for complex heterogeneous platforms



[1] S. Mohindra, J. Daly, R. Haney, and G. Schrader, “Task and Conduit Framework for Multi-core Systems,” in DoD HPCMP Users Group Conference, 2008. DOD HPCMP UGC, pp. 506-513, 2008.

Motivation

- Minimal programmer effort to change application platforms
- Enable rapid high-performance application development for complex heterogeneous platforms



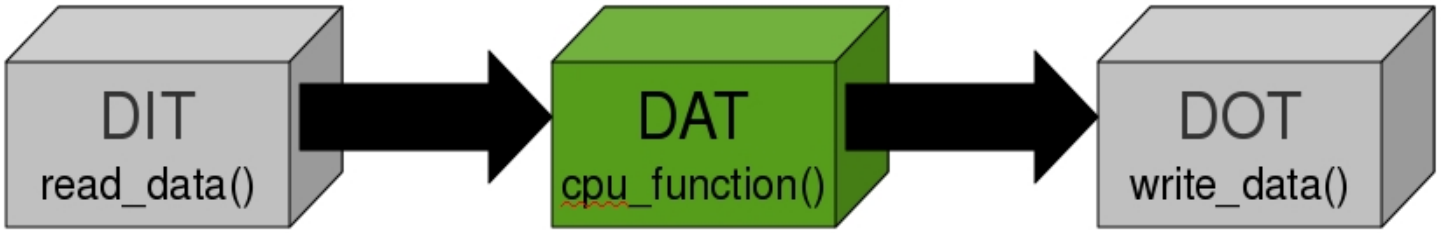
[1] S. Mohindra, J. Daly, R. Haney, and G. Schrader, “Task and Conduit Framework for Multi-core Systems,” in DoD HPCMP Users Group Conference, 2008. DOD HPCMP UGC, pp. 506-513, 2008.

PVTOL Tasks and Conduits

- Application framework that allows for the development of portable applications
  - Originally developed by MIT Lincoln Laboratory
- Provides an intuitive API for constructing complex, platform independent pipelines
- Provides abstractions for different types of concurrency
  - Task and data parallelism
- Tasks are mapped to processing elements
- Conduits abstract data movement among tasks

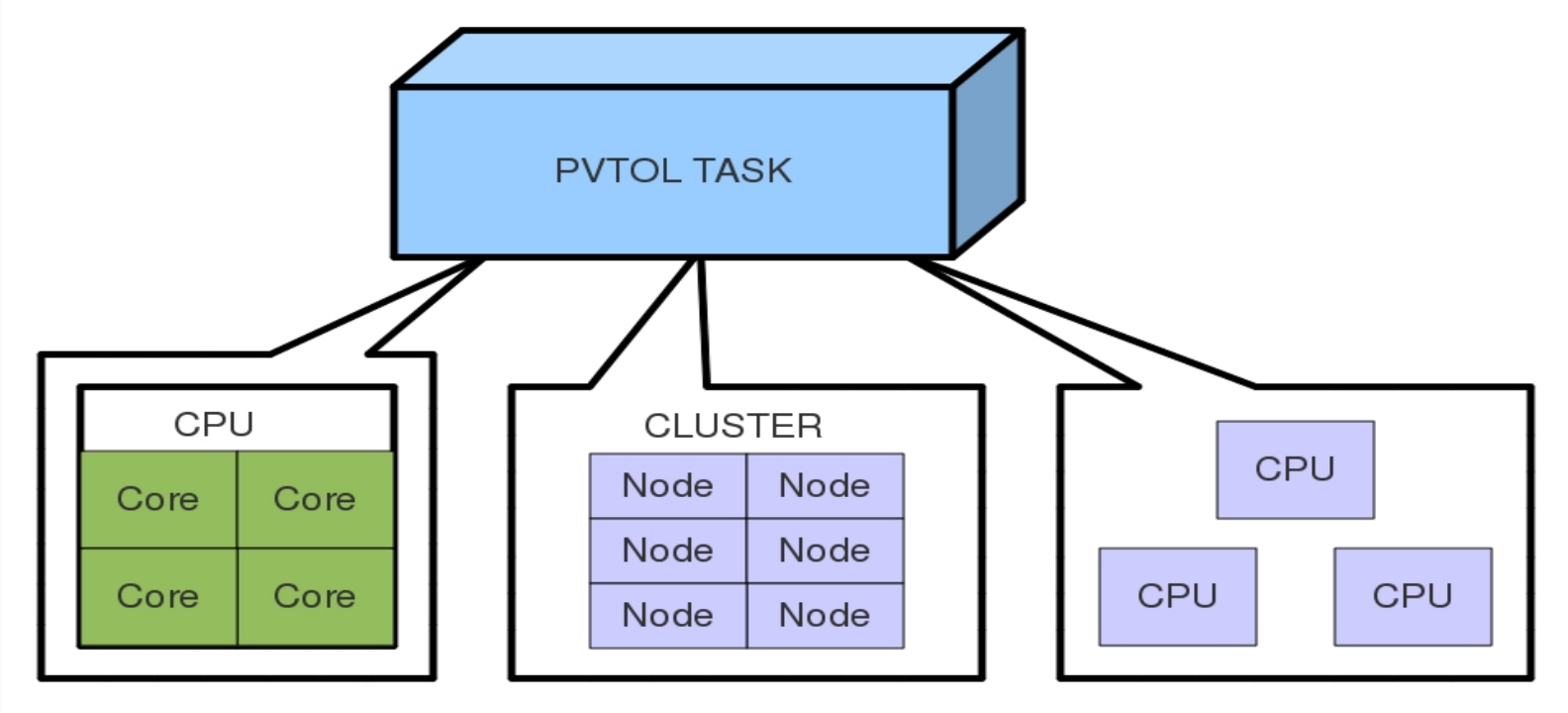
PVTOL Tasks and Conduits

- Basic Example: pipelined Data Input Task (DIT), Data Analysis Task (DAT), and Data Output Task (DOT)
- Data I/O, memory management, and data processing are all separated
  - Allows the programmer to concentrate on core algorithm development (cpu_function)
- The tasks each encapsulate and run some algorithm



PVTOL Tasks

- Task maps are used to assign tasks to processing elements in the system
- PVTOL tasks are mapped to one or more CPU cores, distributed CPUs, etc.

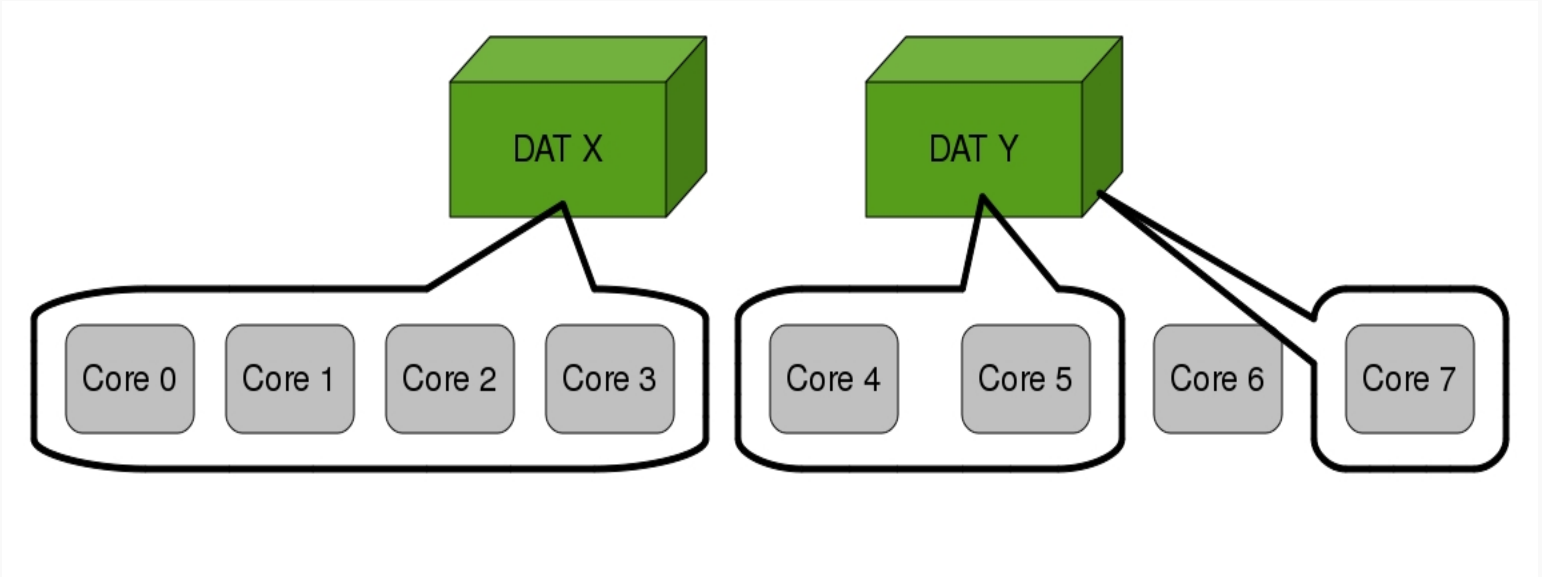


PVTOL Tasks

- **PVTOL Tasks** are hierarchical, modular structures that provide an abstraction for data processing
- Data Maps
  - Designate which data is processed by which task
  - SPMD style of data parallelism
- Task Maps
  - Assign tasks to processing elements
  - Achieves task parallelism
- Tasks communicate with conduits
  - Request incoming data access
  - Call data processing function
  - Signal valid output data

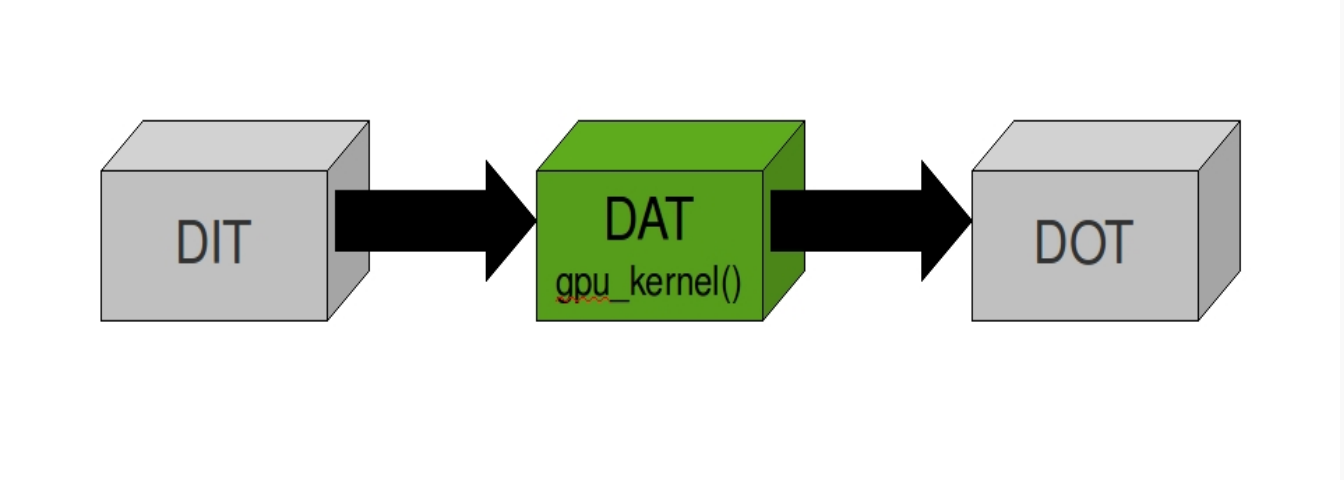
PVTOL Task Maps

```
vector<RankId> rankArrayX = {0,1,2,3};
vector<RankId> rankArrayY = {4,5,7};
TaskMap mapX(rankArrayX);
TaskMap mapY(rankArrayY);
Task<foo> datX(mapX);
Task<bar> datY(mapY);
```



PVTOL GPU Tasks

- Provide an abstraction for GPU computation
  - Same API and constructs as other tasks
  - GPU tasks can be mapped to the same or separate GPU devices
  - Calls to the tasks and conduits API are mapped to CUDA using a set of GPU utility functions



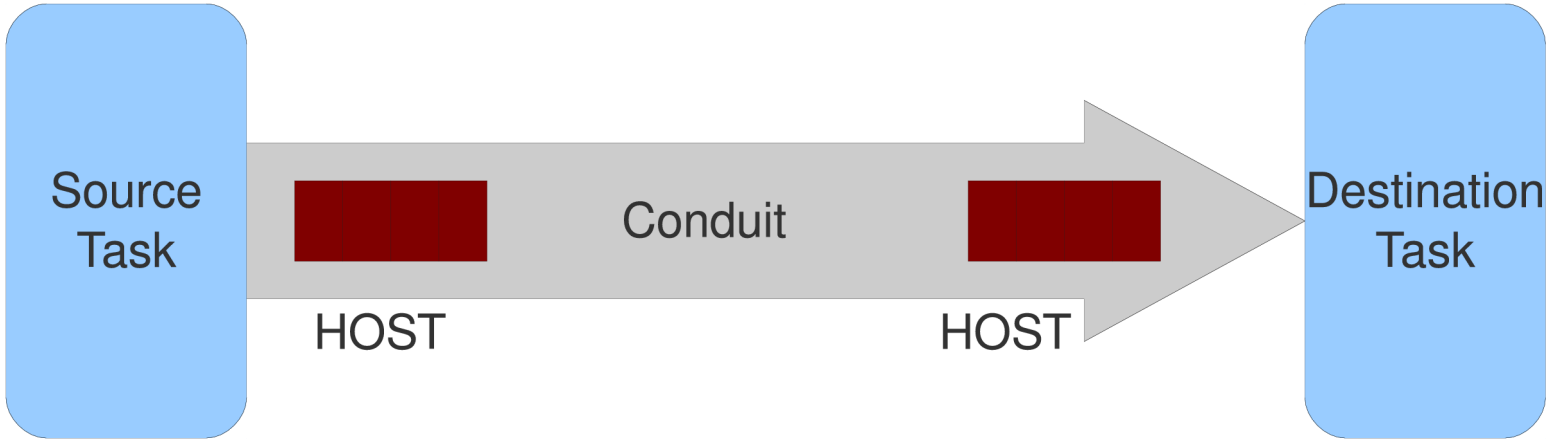
PVTOL GPU Tasks

- Use a single host thread
- Initialize the co-processor
- Execute 1 or more CUDA kernels
  - Including 3rd party kernels (CUFFT, CUBLAS)
- Communicate with GPU conduits to
  - Request incoming data access
  - Signal valid output data

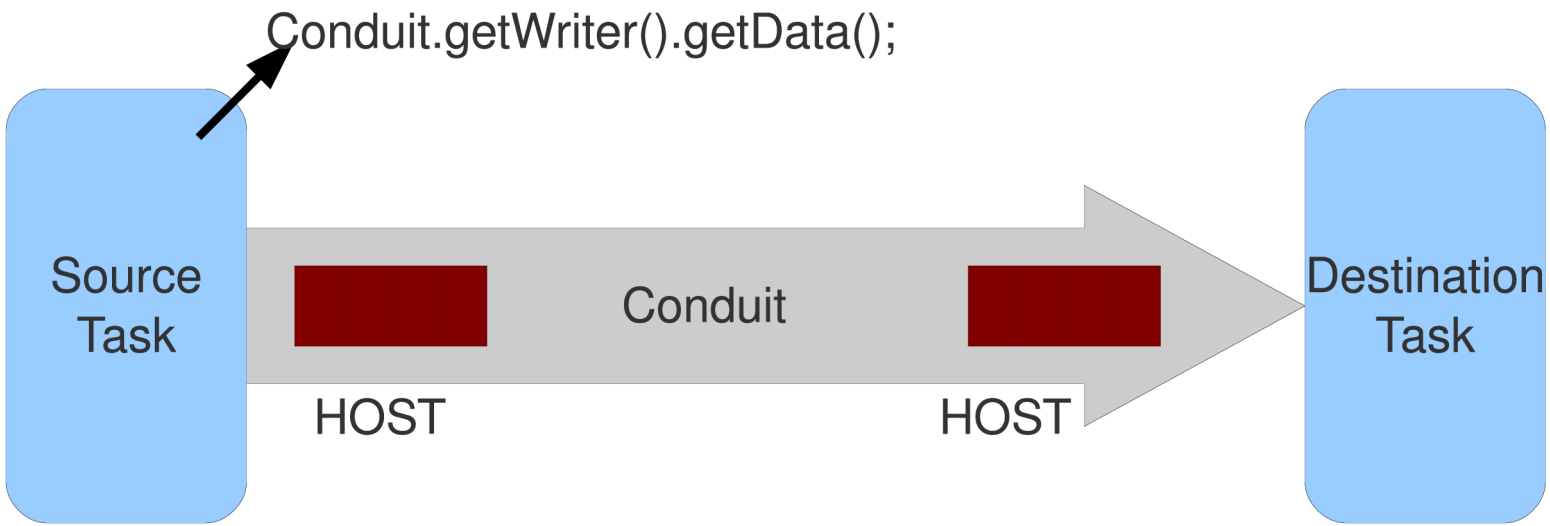
PVTOL Conduits

- **Conduits** are an abstraction for data communication and transfer
  - Support point-to-point data management
  - Brokered Conduits support broadcast/subscribe style communications
  - Used through their writer (insertion) and reader (extraction) interfaces
  - Tasks asynchronously query for access to data buffers
  - Abstract out handling of data synchronization
  - Support multi-buffering (FIFO/Queue)

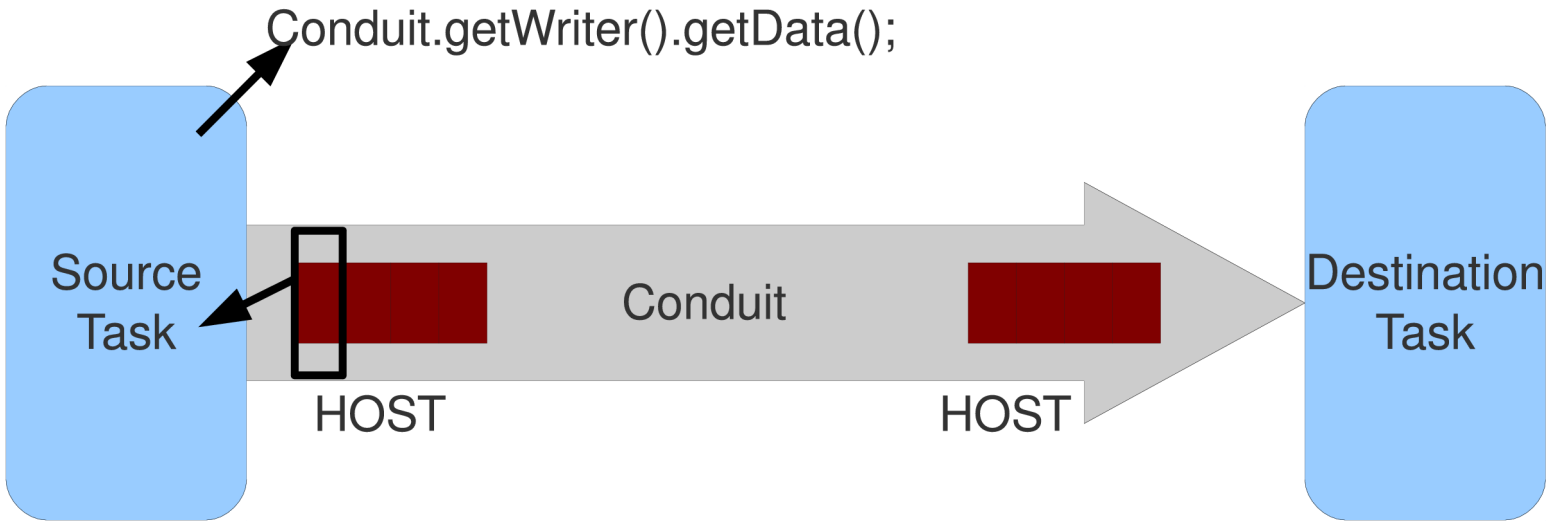
PVTOL Conduits



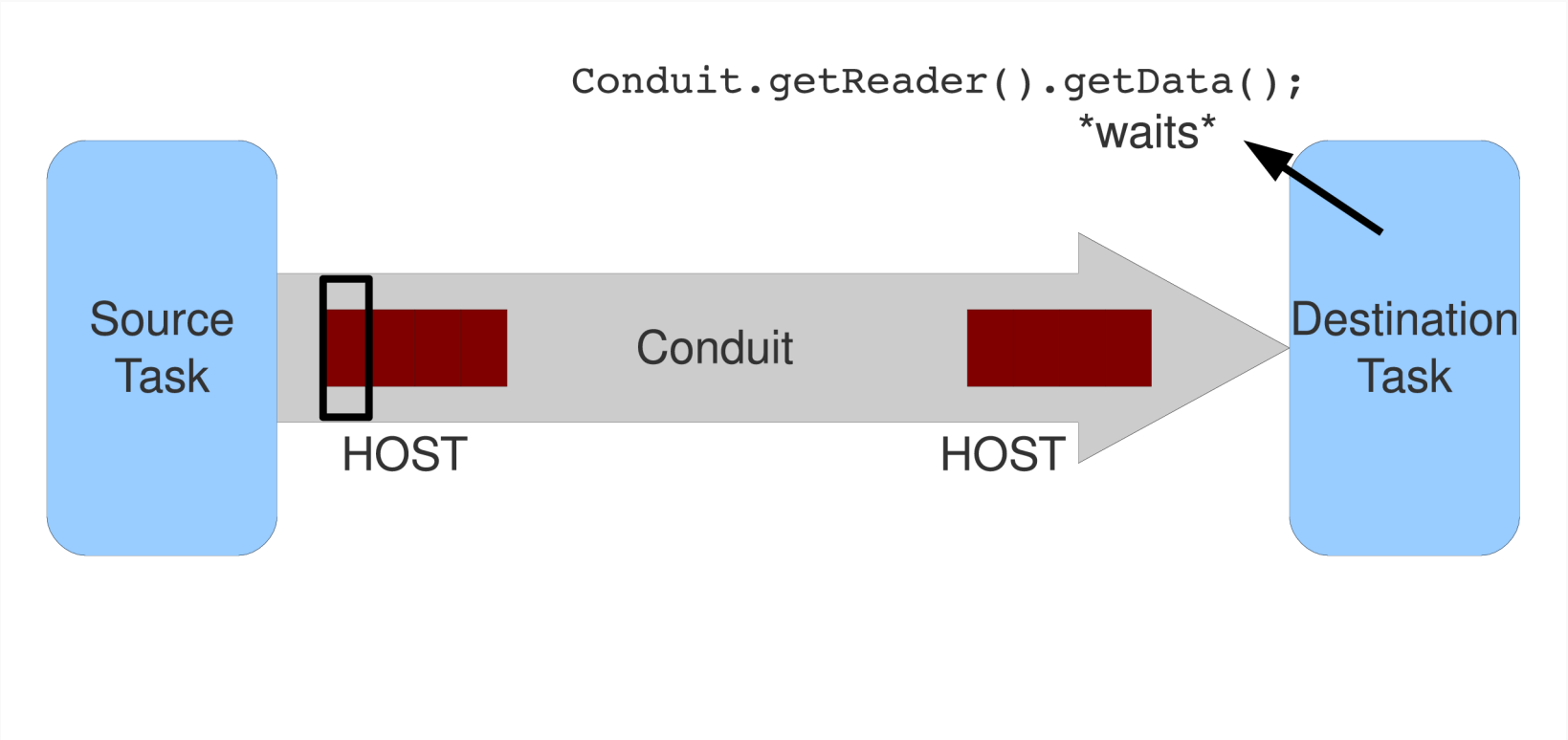
PVTOL Conduits



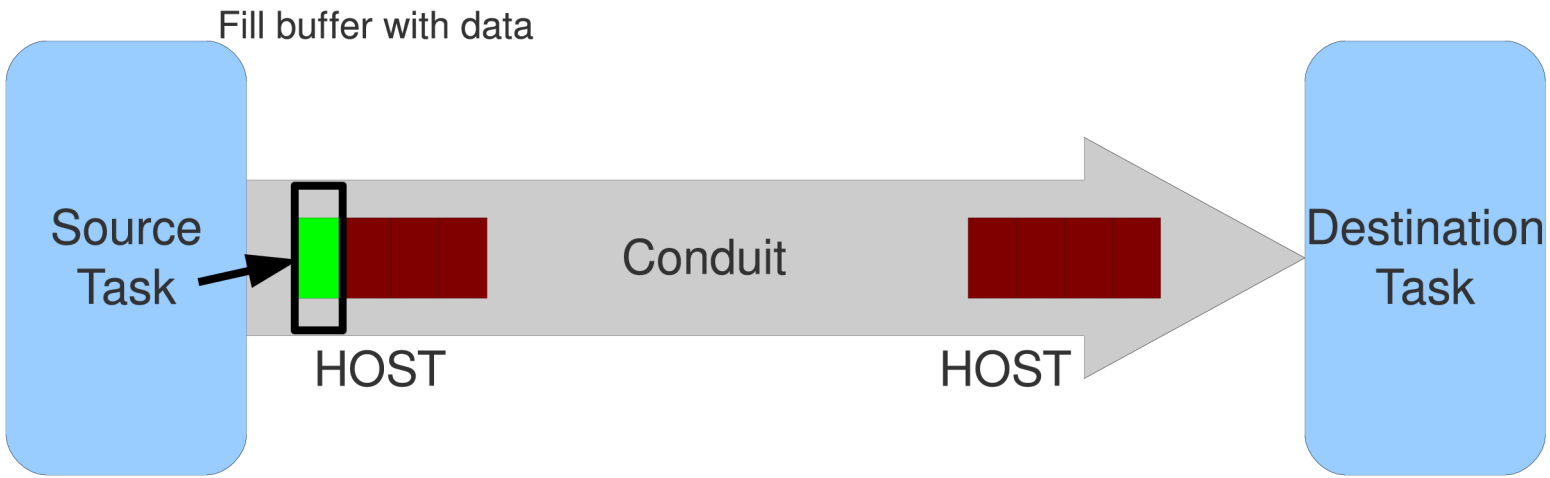
PVTOL Conduits



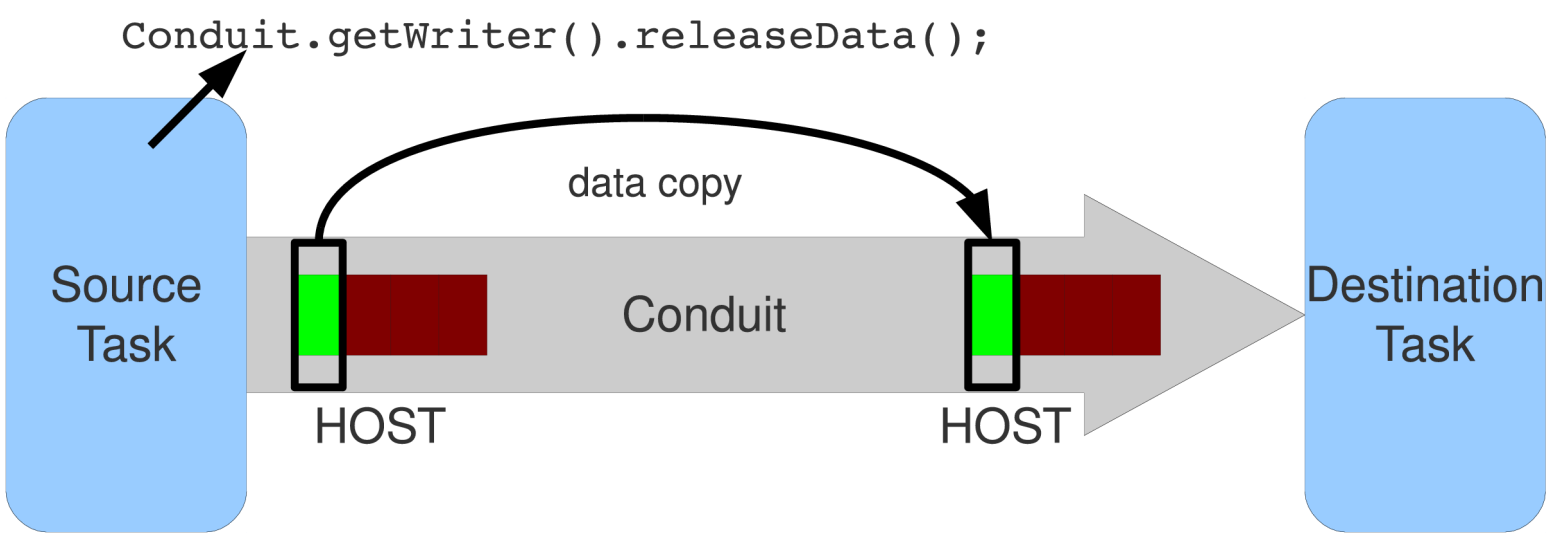
PVTOL Conduits



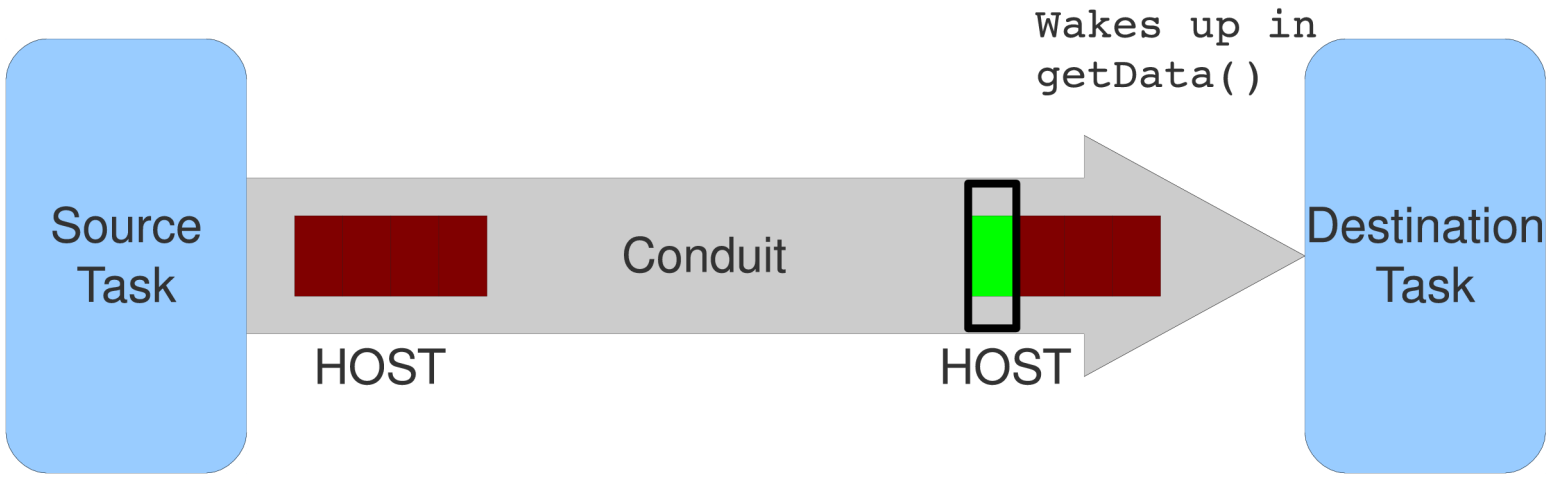
PVTOL Conduits



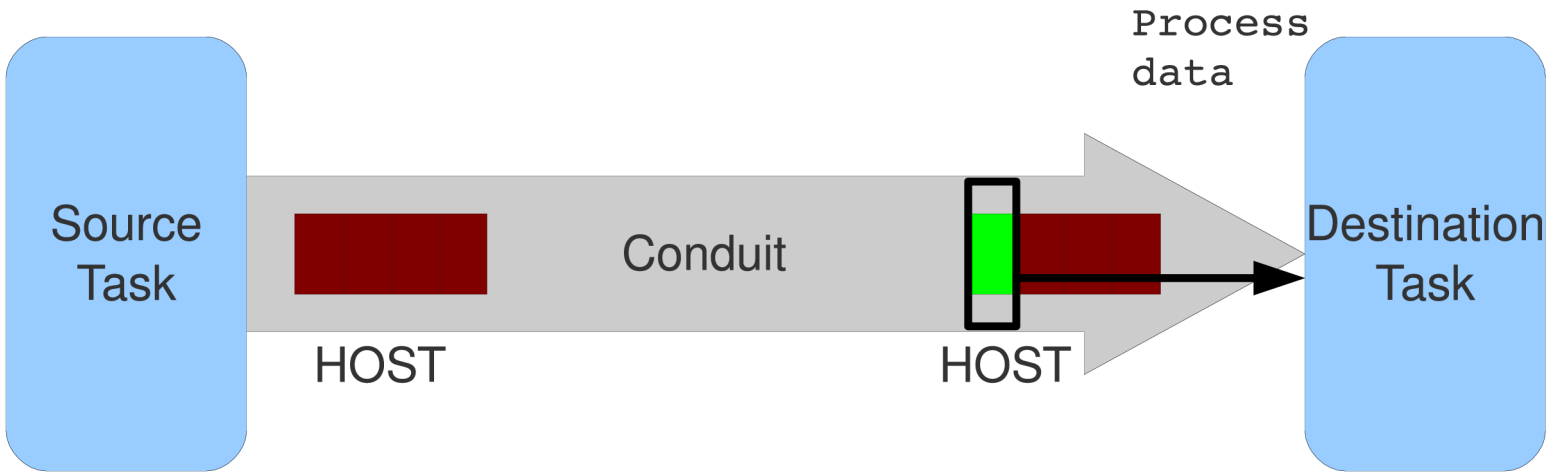
PVTOL Conduits



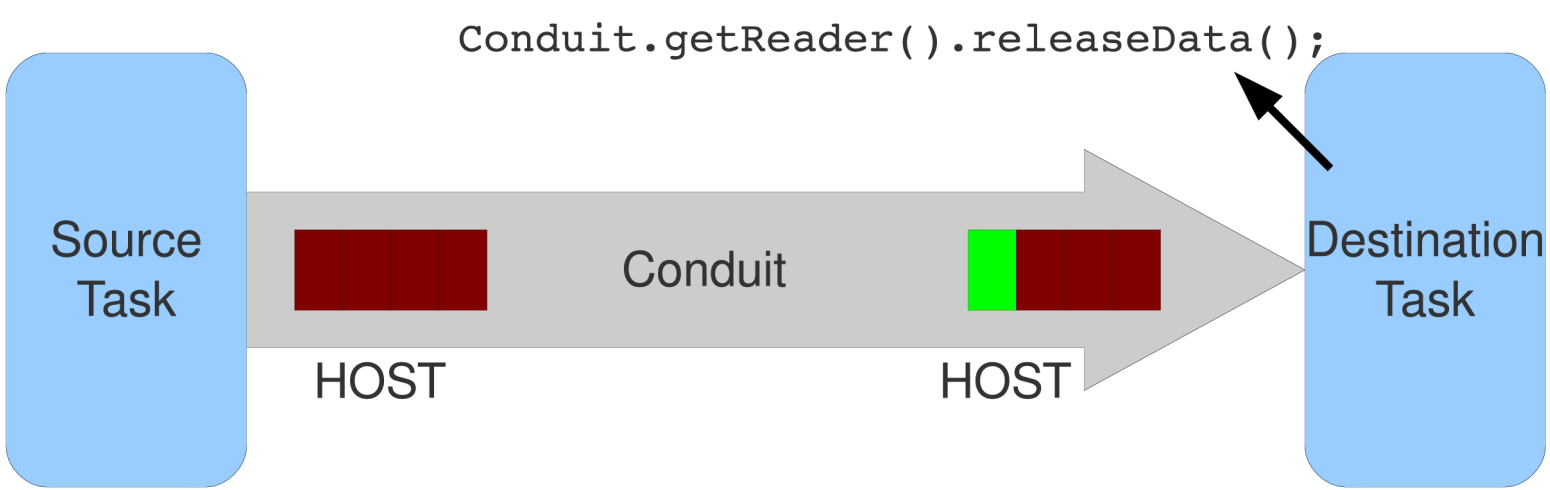
PVTOL Conduits



PVTOL Conduits

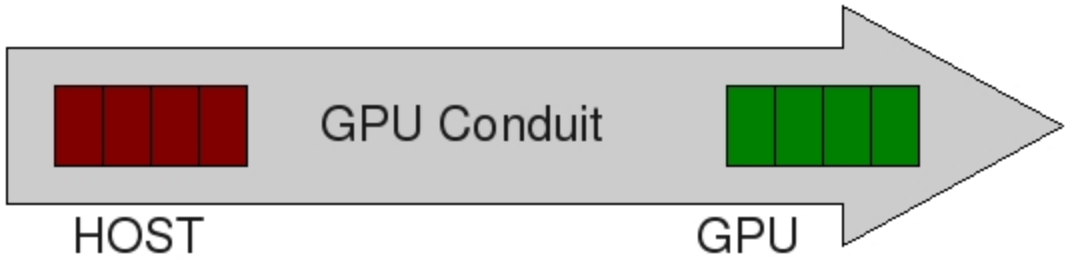


PVTOL Conduits



PVTOL GPU Conduits

- Provide an abstraction for data communication to/from GPU tasks
- Responsible for managing the data buffers and memory transfers
  - Currently, all GPU buffers use global device memory



PVTOL GPU Conduits

- Support any data type in up to three dimensions
- Uses data dimensions to intelligently allocate buffers and transfer data
- Can share buffers with other conduits
- Lock conduits to avoid unnecessary data transfers

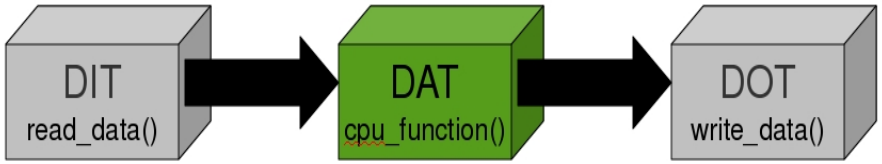
PVTOL GPU Conduits

- Current library of conduits (CUDA)
  - HOST → GPU
  - GPU → HOST
  - GPU(X) → HOST → GPU(Y)
  - GPU(X) → GPU(X), shared data buffer
- Future library of conduits
  - OpenCL Tasks and Conduits
  - Conduit(s) for interoperability between NVIDIA and AMD GPU devices

GPU Utility Functions

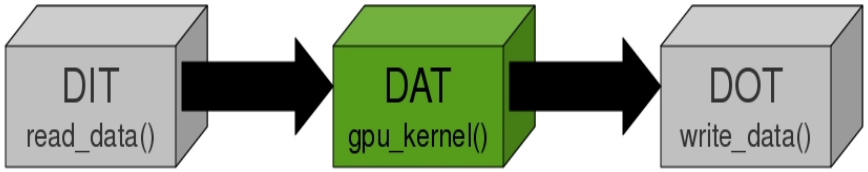
- Utility functions create a platform independent interface to CUDA
  - GPU initialization
  - Memory allocation
  - Data transfers
  - Parameter checking
  - Launching kernels
  - Memory deallocation

Simple pipeline Example



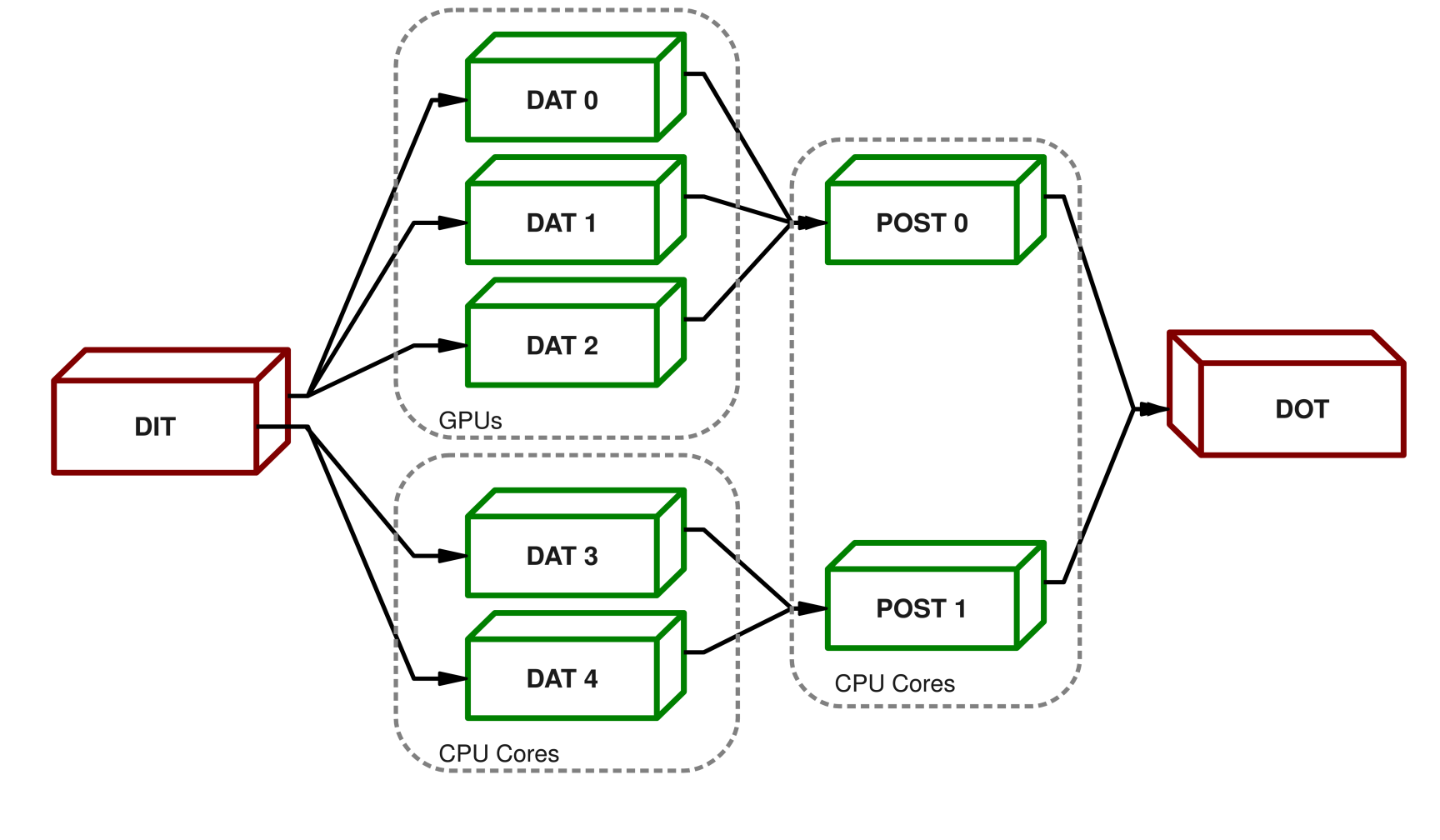

```
/* Main Function */
int main1(int argc, char *argv[]) {
   // Initialize PVTOL
   PvtolProgram prog(argc, argv);

      // Create rank ID vectors and task maps.
   vector<RankId> rank;
   rank.push_back(0);
   RankList ranks(rank);
   TaskMap tMap(ranks);

   // Construct tasks
   Task<Dit>   dit("DIT", tMap);
   Task<Dat>   dat("DAT", tMap);
   Task<Dot>   dot("DOT", tMap);

   // Construct conduits connecting tasks
   SharedConduit<HOST2HOST, float> cdtDitToDat("DIT to DAT");
   SharedConduit<HOST2HOST, float> cdtDatToDot("DAT to DOT");

   // Set up conduit data dimensions
   int cdtDitToDatDims[NDIMS] = {X_SIZE, Y_SIZE, 1};
   int cdtDatToDotDims[NDIMS] = {X_SIZE, 1, 1};

   // Initialize the tasks
   dit.init(N_BUFFERS, N_DATASETS, 1, cdtDitToDat, cdtDitToDatDims);
   dat.init(N_BUFFERS, N_DATASETS, 1, cdtDitToDat, cdtDitToDatDims, 1, cdtDatToDot, cdtDatToDotDims);
   dot.init(N_BUFFERS, N_DATASETS, 1, cdtDatToDot, cdtDatToDotDims);

   // Run the tasks
   dit.run(); dat.run(); dot.run();

   // Wait until tasks complete
   dit.waitTillDone(); dat.waitTillDone(); dot.waitTillDone();
```

Simple pipeline Example




```
/* Main Function */
int main1(int argc, char *argv[]) {
   // Initialize PVTOL
   PvtolProgram prog(argc, argv);

      // Create rank ID vectors and task maps.
   vector<RankId> rank;
   rank.push_back(0);
   RankList ranks(rank);
   TaskMap tMap(ranks);

   // Construct tasks
   Task<Dit>   dit("DIT", tMap);
   Task<Dat>   dat("DAT", tMap);
   Task<Dot>   dot("DOT", tMap);

   // Construct conduits connecting tasks
   SharedConduit<HOST2GPU, float> cdtDitToDat("DIT to DAT");
   SharedConduit<GPU2HOST, float> cdtDatToDot("DAT to DOT");

   // Set up conduit data dimensions
   int cdtDitToDatDims[NDIMS] = {X_SIZE, Y_SIZE, 1};
   int cdtDatToDotDims[NDIMS] = {X_SIZE, 1, 1};

   // Initialize the tasks
   dit.init(N_BUFFERS, N_DATASETS, 1, cdtDitToDat, cdtDitToDatDims);
   dat.init(N_BUFFERS, N_DATASETS, 1, cdtDitToDat, cdtDitToDatDims, 1, cdtDatToDot, cdtDatToDotDims);
   dot.init(N_BUFFERS, N_DATASETS, 1, cdtDatToDot, cdtDatToDotDims);

   // Run the tasks
   dit.run(); dat.run(); dot.run();

   // Wait until tasks complete
   dit.waitTillDone(); dat.waitTillDone(); dot.waitTillDone();
```

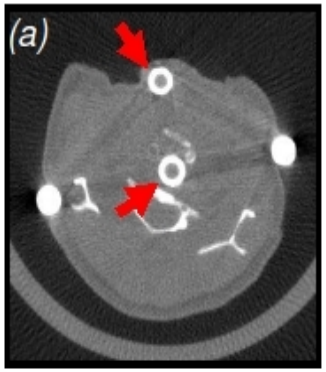
Complex Example



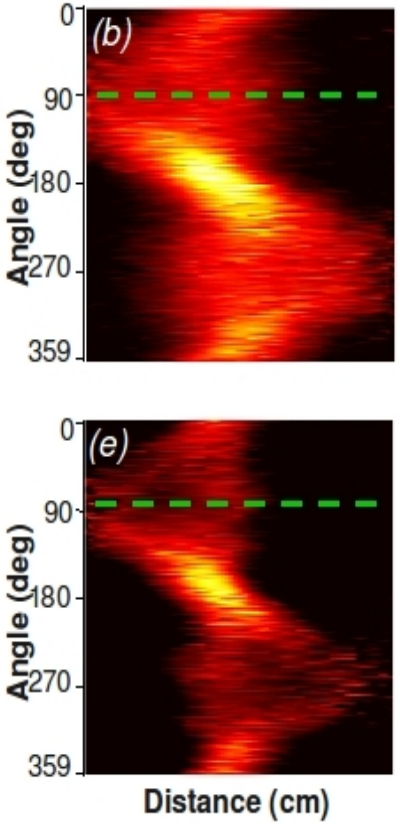
Fluorescence Mediated Tomography (FMT)

- A means of non-invasive molecular medical imaging
  - Supports 3D visualization of live animals
- Fluorescent indicators are used to highlight types of tissue
- Transmitted photons (at different angles) are used to reconstruct the image
- Monte Carlo simulations are a very accurate way of tracking light propagation through the tissue

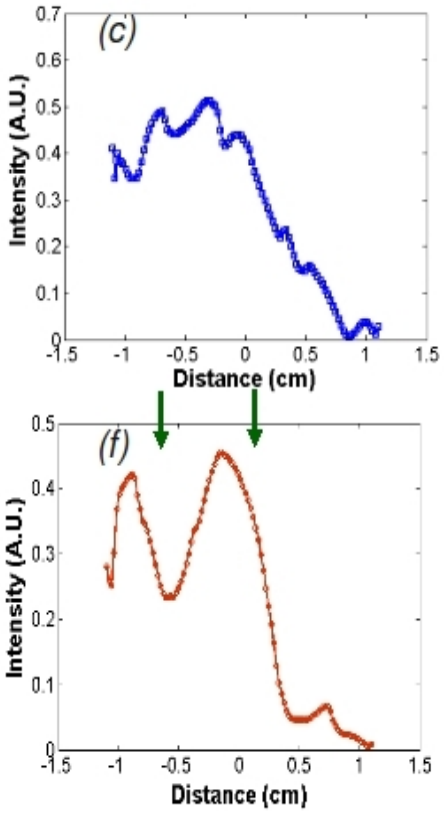
Fluorescence Mediated Tomography (FMT)



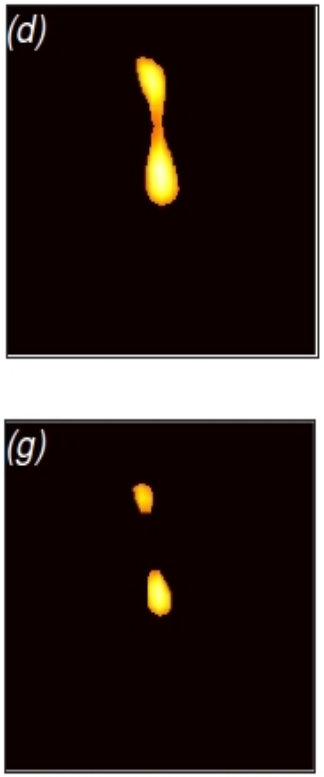
Mouse body with Flourescent Indicators



Transmitted Photon Intensity



Transmitted Normalized Intensity



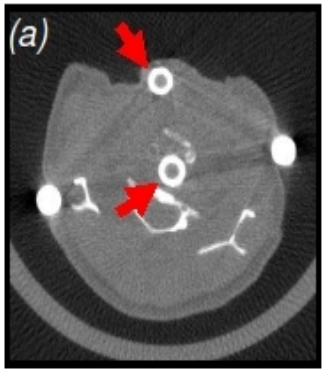
All Photons

Early Photons

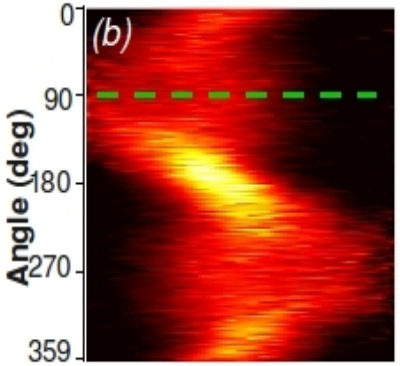
Reconstructed FMT Images

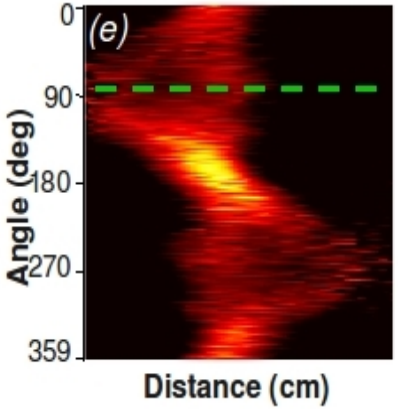
[2] Niedre, M.J. and Ntziachristos, V. Comparison of fluorescence tomographic imaging in mice with early-arriving and quasi continuous-wave photons. Opt Lett 2010; 35:369-371.

Fluorescence Mediated Tomography (FMT)

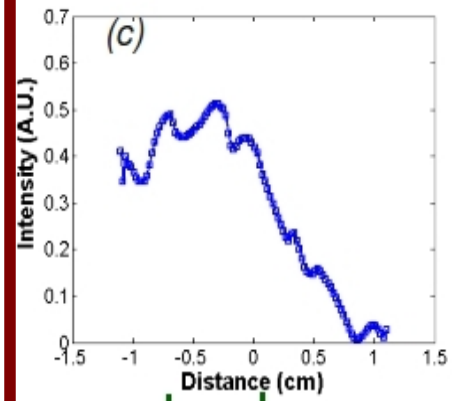


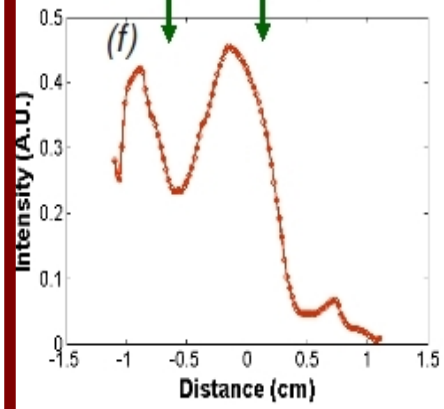
Mouse body with Flourescent Indicators



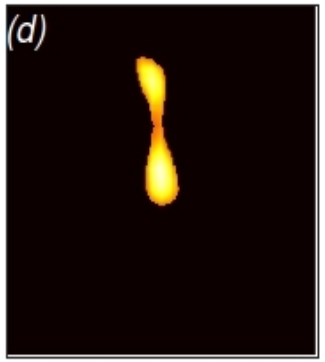


Transmitted Photon Intensity

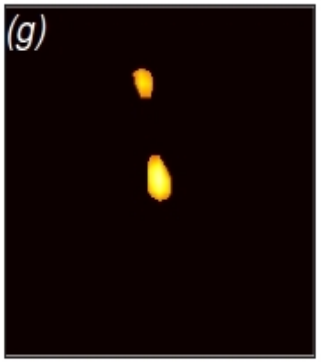




Transmitted Normalized Intensity



All Photons

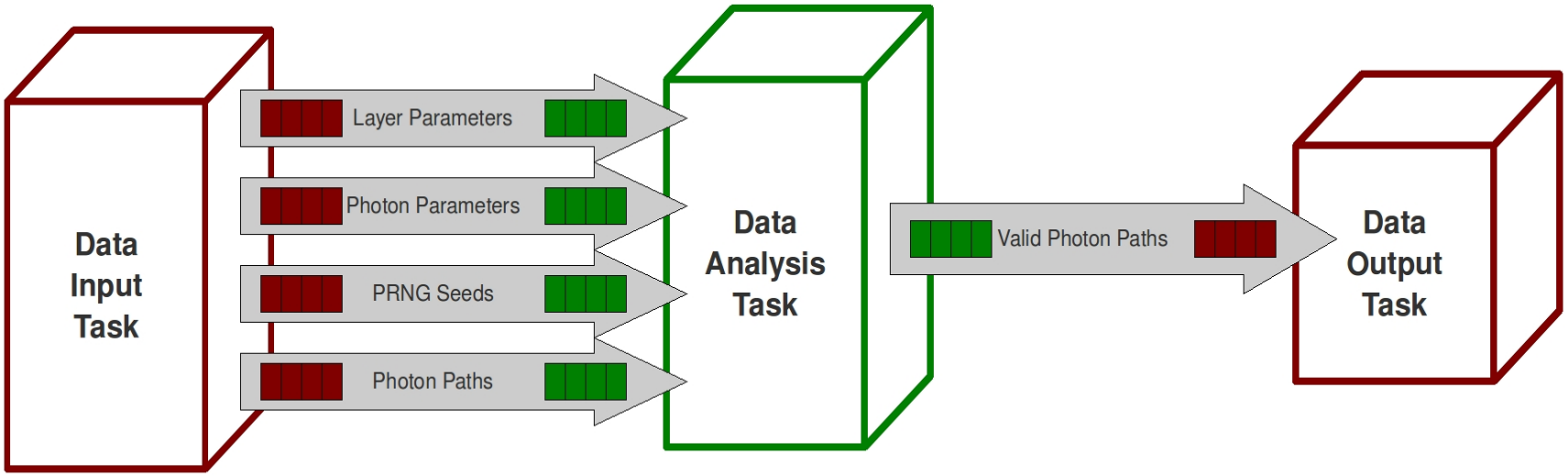


Early Photons

Reconstructed FMT Images

[2] Niedre, M.J. and Ntziachristos, V. Comparison of fluorescence tomographic imaging in mice with early-arriving and quasi continuous-wave photons. Opt Lett 2010; 35:369-371.

FMT Application (current)



- Tracks photons from one source to one detector
- Porting C/C++ application to GPU application
  - Create GPU kernel
  - Change ~10 SLOC in PVTOL Tasks and Conduits

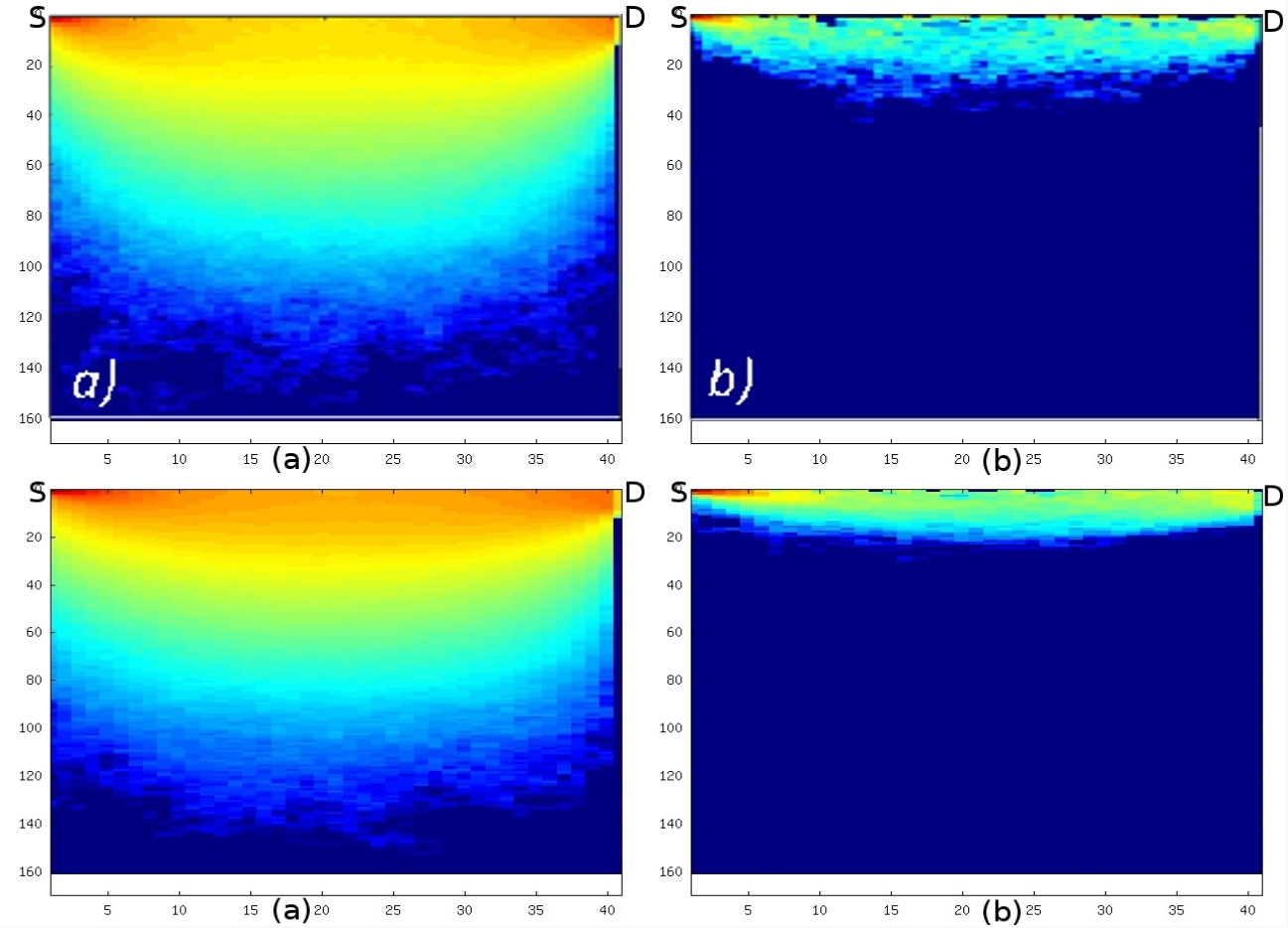
Test Environment

- Hardware
  - CPU: Quad-core AMD Opteron
  - GPU: NVIDIA Tesla S1070
  - 24 GB RAM
- Software
  - 64-bit CentOS
  - NVIDIA CUDA SDK 3.0
- Test Scenario
  - 100 Million photons simulated
  - 2 cm of homogeneous tissue
  - 1 source, 1 detector

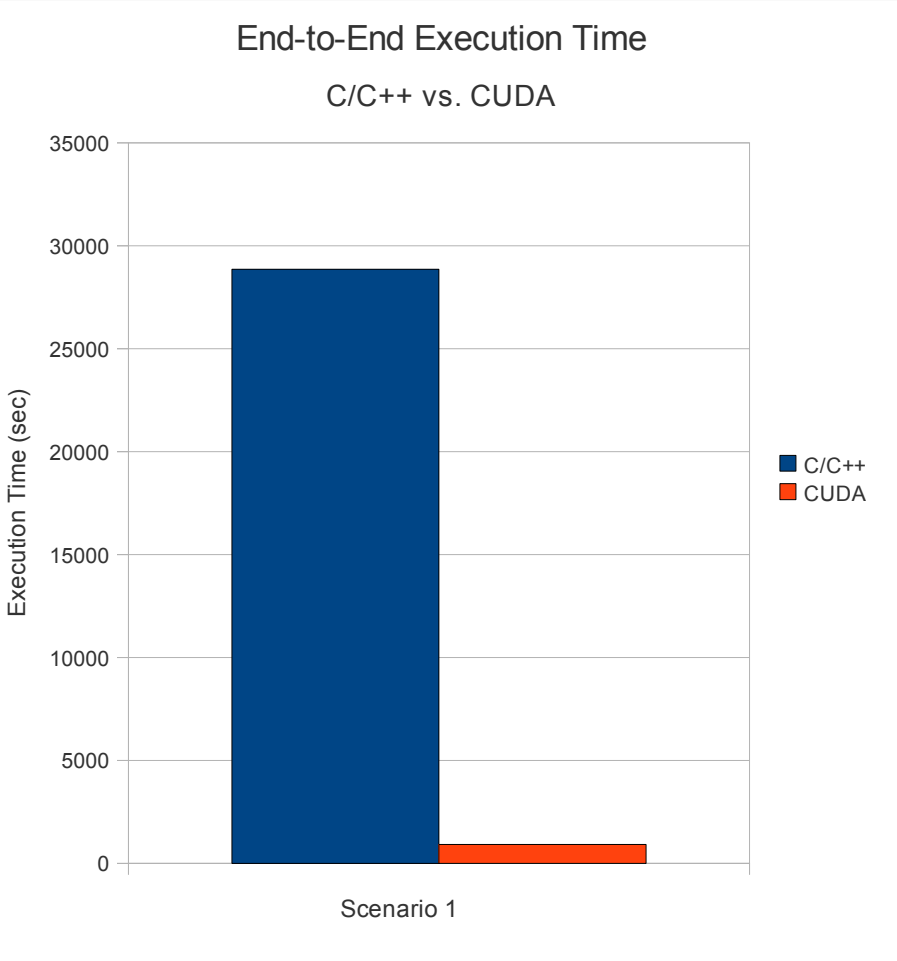
Photon Path Density Output

No Time Gate

Early Time

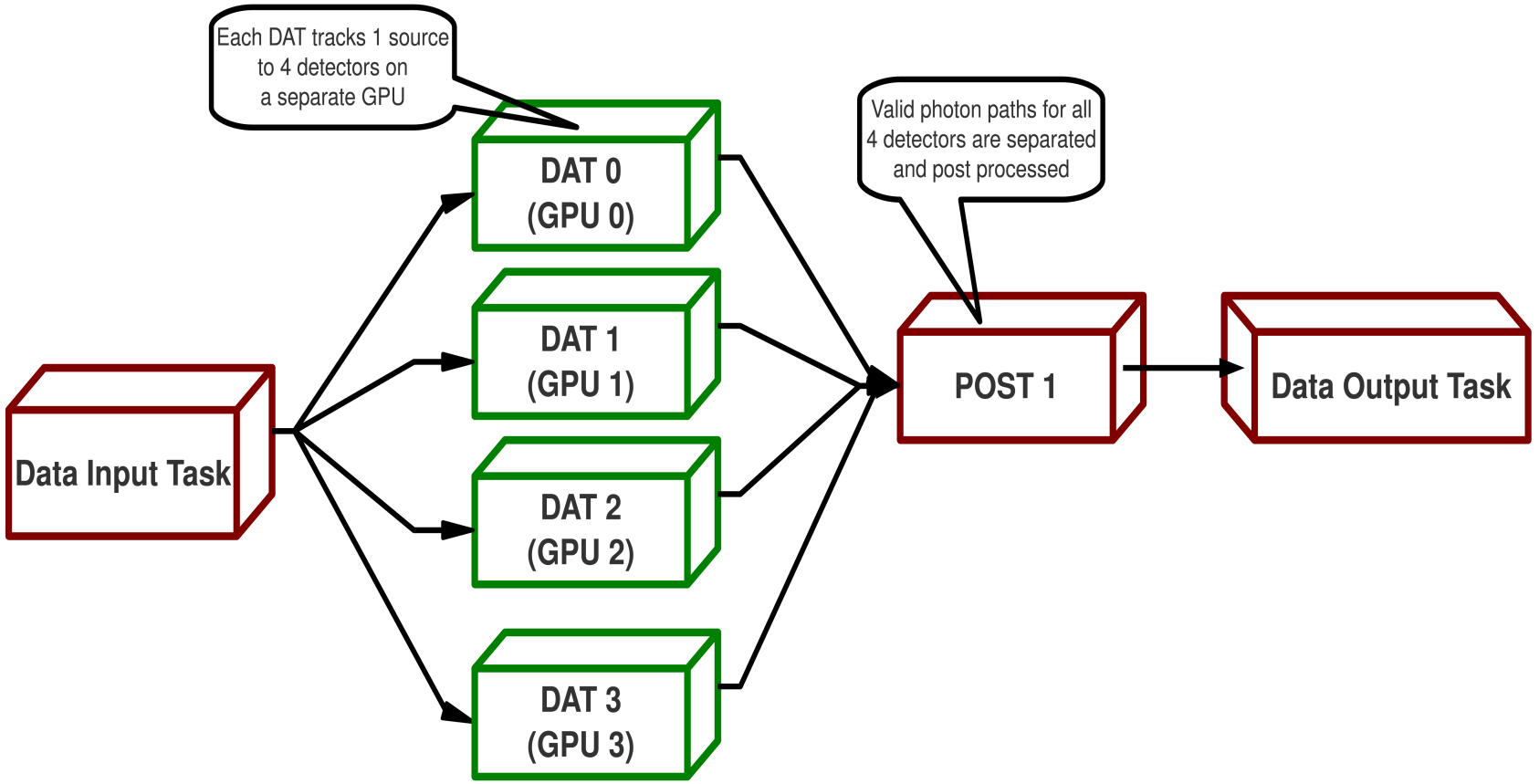


Current Results



- **~31.5x speedup**
- Reduces simulation time from 8 hours to 15 minutes
- CUDA code is not fully optimized
  - Lots of conditionals left
- Global memory is used very little, but accesses can't currently be coalesced

FMT Application (in progress)



Conclusions

- GPU Tasks and Conduits enables rapid application development on heterogeneous platforms
- GPU Tasks and Conduits extends PVTOL to a wider variety of platforms
- Monte Carlo simulation of photon propagation is a good target for GPU speedup
- FMT Monte Carlo Simulations are greatly sped up through the use of GPUs

Future Work

- Tasks and Conduits
  - OpenCL Support
  - Automated Pipeline Construction
  - Improved results for CPU, NVIDIA GPU, and AMD GPU
  - Better GPU memory support (local, constant, texture)

- FMT
  - New Version of the algorithm for complex geometries of tissue
  - Single source, multiple detectors
  - Parallel pipeline stages (DAT's) for increased concurrency
  - OpenCL version of the algorithm

Questions?

James Brock
jbrock@ece.neu.edu

Miriam Leeser
mel@coe.neu.edu

Mark Niedre
mniedre@ece.neu.edu