Wexner Medical Center

2012 *TASSA Ohio Engineering Workshop*

Challenges and Lessons Learned in Data-Intensive Computing

Ümit V. Çatalyürek

Professor

Department of Biomedical Informatics

Department of Electrical & Computer Engineering

Department of Computer Science & Engineering (courtesy)

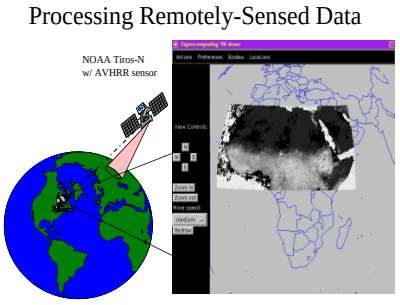
The Ohio State University

Outline

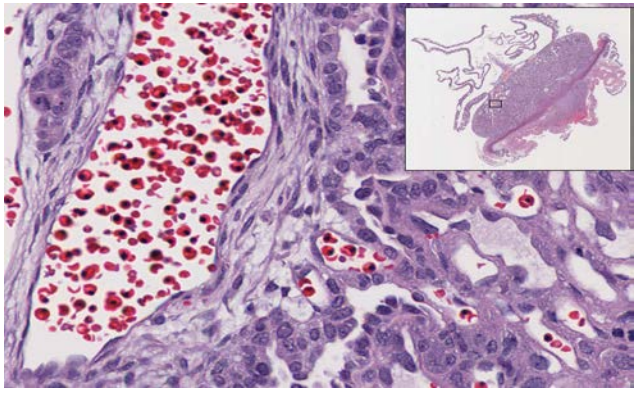
- 1999-2012: Data-Intensive Applications
  - Applications; Characteristics, Commonalities; Middleware
- Motivation
  - Hardware; Software
- Dataflow Application Model and Runtime System
- Runtime System Optimizations for
  - Dynamic Workloads; Finitely Divisible Loads
- Conclusion
  - Summary; Lessons Learned; Truth Telling

Where I am coming from: Applications associated with Large Datasets

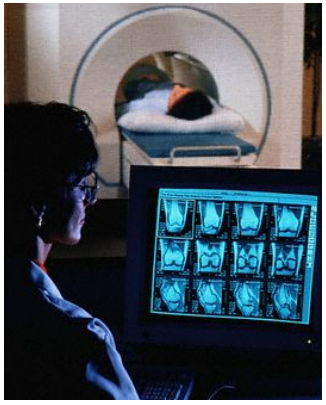
Wexner Medical Center



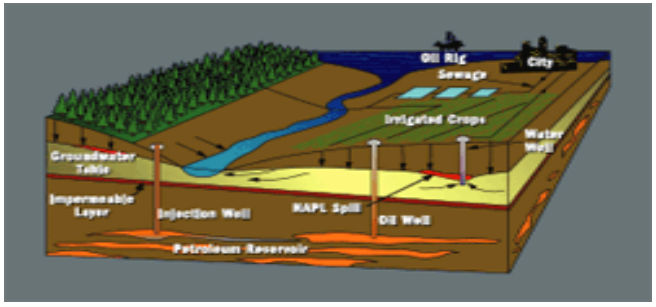
Satellite Data Processing



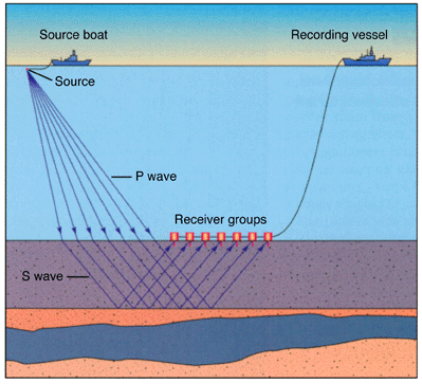
Digital Pathology



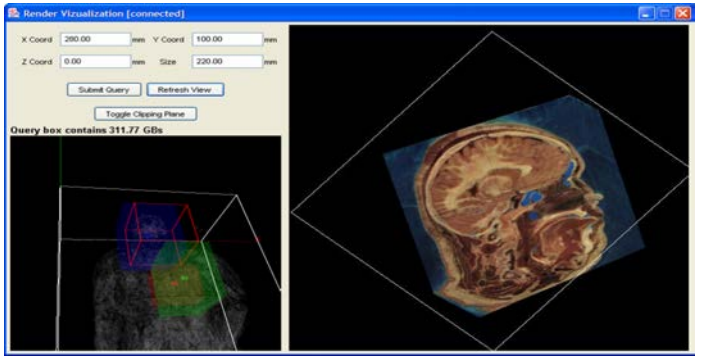
DCE-MRI Analysis



Managing Oilfields, Contaminant Transport



Seismic Imaging



Visualization -Virtual Human

Characteristics, Commonalities…

- Spatio-temporal datasets (generally low dimensional) – datasets describe physical scenarios
  - Multi-dimensional, Multi-resolution, Multi-scale
- Very large file-based datasets
  - Hundreds of gigabytes to 100+ TB data
  - Data is stored in a distributed collection of files
  - Lots of datasets, lots of files
- Data products often involve results from ensemble of spatio-temporal datasets
- Some applications require interactive exploration of datasets
- Common operations: subsetting, filtering, interpolations, projections, comparisons, frequency counts
- Modeling and management of data analysis workflows

Data Services

Wexner Medical Center

- Distributed data processing support
- Grid based data virtualization, data management, query, on demand data product generation
- Distributed metadata and data management
- Track metadata associated with data and data analysis workflows

Middleware Support

Wexner Medical Center

- Data Analysis on Clusters: Active Data Repository (ADR)
  - Targets storage, retrieval and manipulation of multi-dimensional datasets on parallel machines
  - Several services are customizable for various application specific processing
  - Data retrieval, memory management, and scheduling of processing
  - Data declustering/clustering, Indexing
- Active Semantic Data Cache: Active Proxy G
  - Employ user semantics to cache and retrieve data
  - Store and reuse results of computations
- Application of Grid Technology in Cancer Research: caGrid
  - Driven by cancer research community requirements
  - Services-Oriented, Metadata driven
  - Creation and management of a strongly-typed, multi-institutional, research grid
  - Leverages existing technology: e.g. caDSR, EVS, Mobius GME, Globus
- Distributed Metadata and Data Management: Mobius
  - Create, manage, version data definitions
  - Management of metadata and data instances
  - Data integration

Middleware Support (cont'd)

Wexner Medical Center

- Data Virtualization: STORM
  - Large data querying capabilities, layered on DataCutter
  - Distributed data virtualization
  - Indexing, Data Cluster/Decluster, Parallel Data Transfer

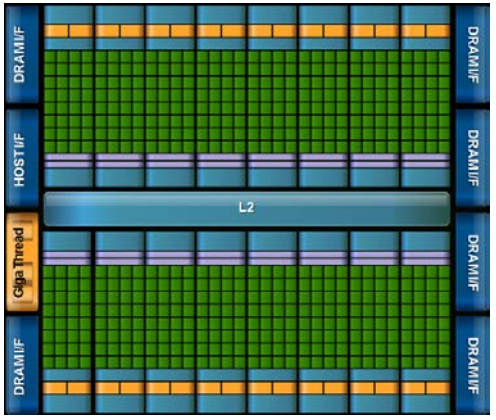
```
SELECT <DataElements>
        FROM Dataset-1, Dataset-2,..., Dataset-n
        WHERE <Expression> AND <MyFilterFunc(<DataElement>)>
        GROUP-BY-PROCESSOR ComputeAttribute(<DataElement>)
```

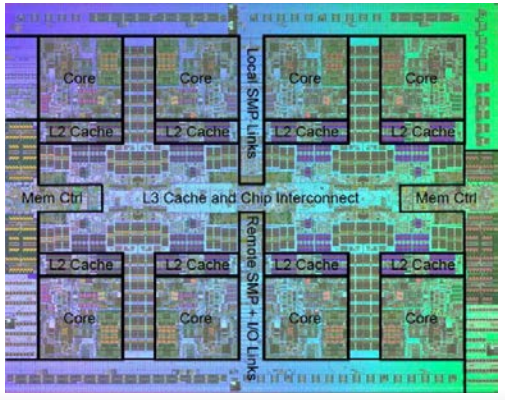
- Data Analysis/Processing Workflows: DataCutter/DataCutter-Lite and Anthill
  - Component Framework for Combined Task/Data Parallelism
  - Filtering/Program coupling Service: Distributed C++/Java/Python component framework
  - On demand data product generation
    - more later..

Hardware Motivation: Current & Future Systems

Wexner Medical Center

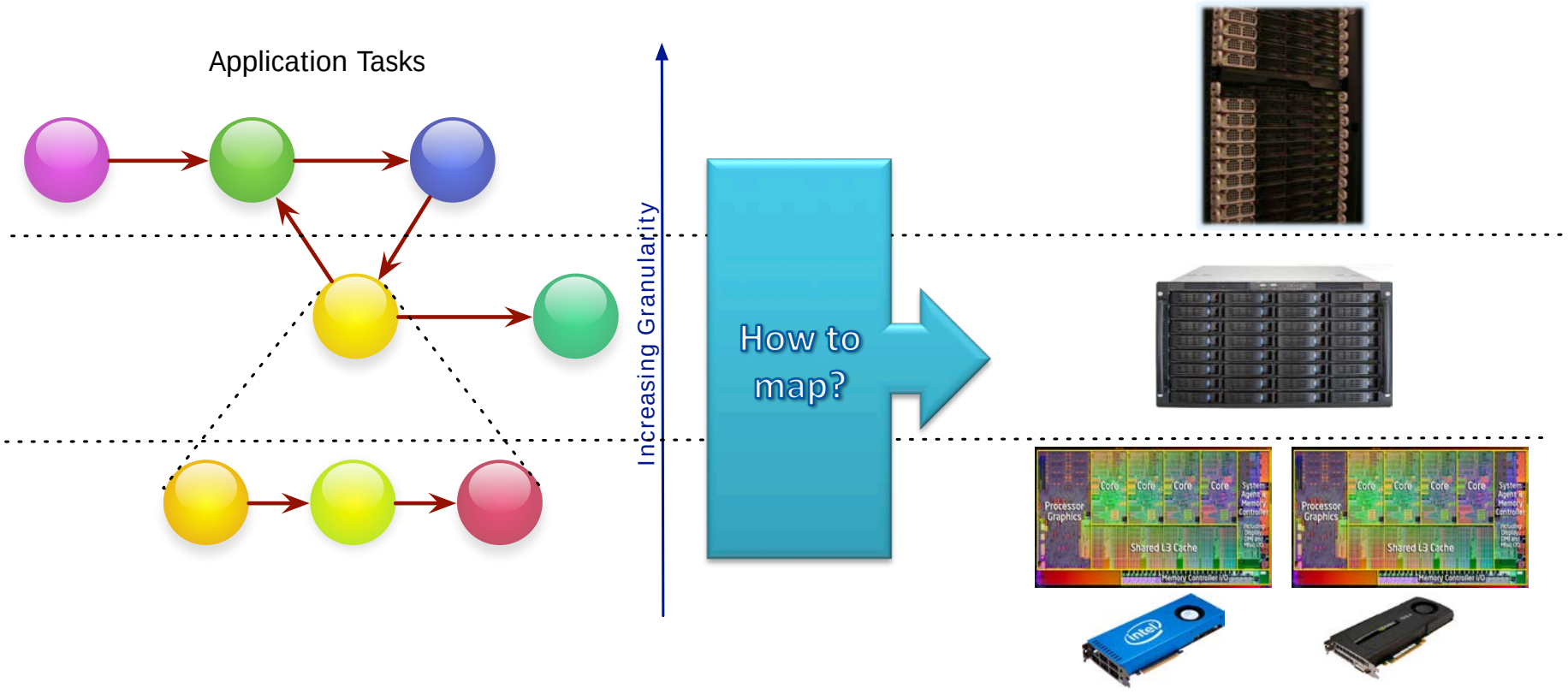
- More and more machines composed of multi-core and many-core CPUs, and accelerators
- Top500's top 5 has 3, Green500's top 10 has 6 supercomputer with GPUs (5 Nvidia, 1 ATI)
- Some examples:
  - Intel MIC Knights Corner
    - 50+ x86 cores, 4-way SMT per core
    - 512-byte SIMD registers
  - NVIDIA Fermi GPU
    - Up to 512 cores
    - Up to 600 GFLOPS of double-precision
  - IBM POWER7
    - Up to 8 cores, 4-way SMT per core
    - 32 MB on-die L3 eDRAM cache





Software Motivation:
How to utilize all resources effectively?

Wexner Medical Center



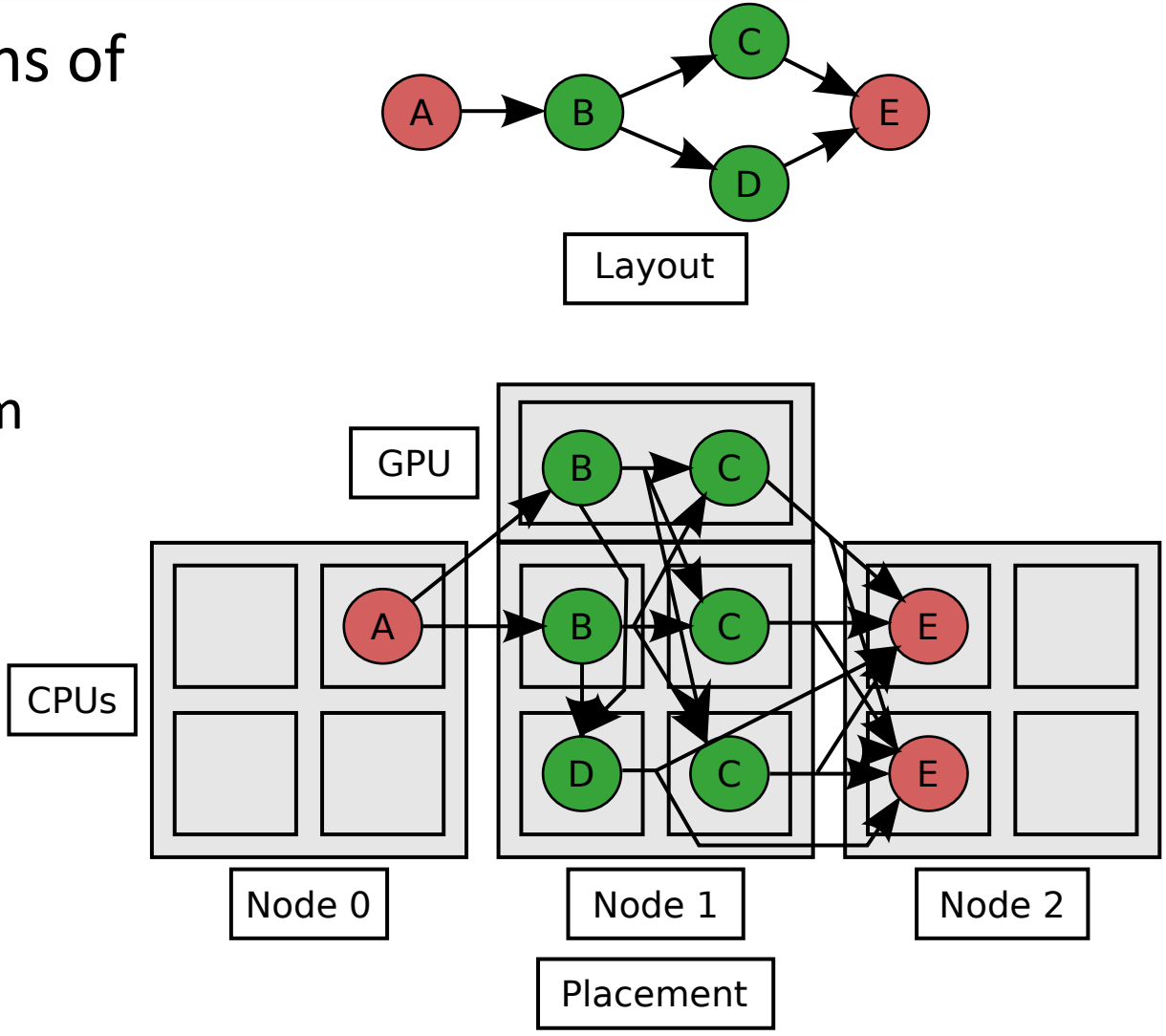
Component-Based Dataflow

Wexner Medical Center

- OSU/UMD-DataCutter, UMFG/OSU-Anthill
- Application decomposed into a task-graph (*filters & streams*)
  - Task graph performs computation
    - Individual tasks perform single function
  - Tasks (*filters*) are independent, with well-defined interfaces
  - Communication using *streams*
    - Transparent filter instances – Single stream illusion
  - Intra-task architecture-specific optimizations possible
    - Architectural heterogeneity abstracted
- Runtime systems' network interface matches high-bandwidth networks' communication model
  - Application tasks communicate using data buffers through streams
  - Asynchronous communications overlap with computation
- Robust load balancing techniques

Component-Based Dataflow

- Multiple dimensions of parallelism
  - Task parallelism
  - Data parallelism
  - Pipeline parallelism



Runtime Optimization Challenges

Wexner Medical Center

- We will look at two different class of applications:
  - Dynamic workloads with non-divisible databuffer sizes are given by the application at runtime
  - Applications with *finitely divisible load*

Dynamic Workloads

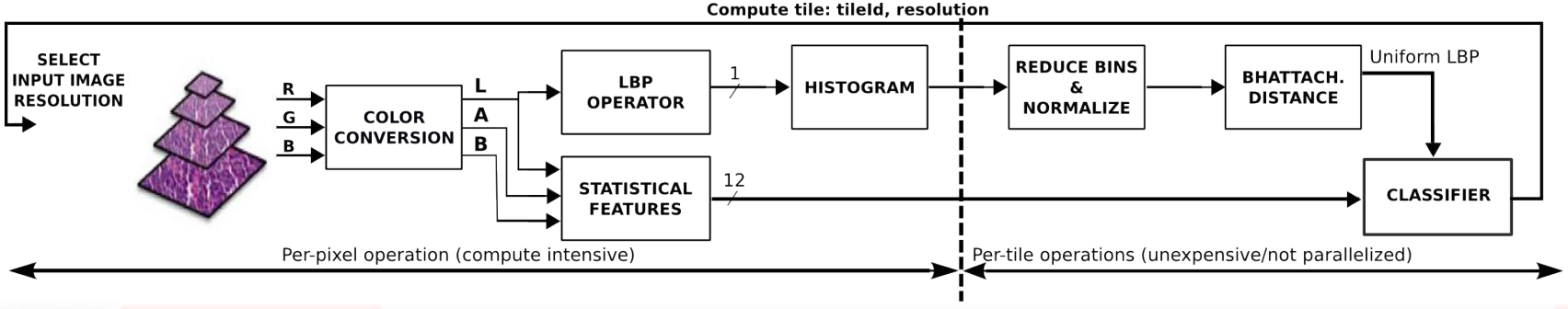
Sample Application: Neuroblastoma Image Analysis

Wexner Medical Center

- Classify biopsy tissue images into different subtypes of prognostic significance

- Very high resolution slides
  - Divided into smaller tiles

- Multi-resolution image analysis
  - Mimics the way pathologists perform their analysis
  - If classification at lower resolution is not satisfactory, analysis algorithm is executed at higher resolution(s), hence the dynamic workload.



Run-time optimizations

Wexner Medical Center

- Improving CPU/GPU data transfers
- Intra-filter task assignment
- Inter-filter optimizations
  - A novel stream policy: on-demand dynamic selective stream

Improving CPU/GPU data transfers

- Data bus bandwidth between CPU/GPU is a critical barrier

- Overlap the data transfer with useful computation
  - GPUs allow multiple concurrent transfers (events)
  - The optimal number of concurrent transfers varies: computation/communication rate

- An automated approach to dynamically configure the number of concurrent copies – based on the *kernel* throughput

Intra-filter task assignment

Wexner Medical Center

- The processing rates of the processors are data-dependent
- Exploit intra-filter task heterogeneity
  - Schedule appropriate tasks to appropriate processors
- **DDWRR** – Demand-Driven Weighted Round Robin
  - Events shared among processors
  - Sorts events according to their performance on each device
  - Selects the event with the highest speedup (out-of-order fashion)

Inter-filter optimization: On-Demand Dynamic Selective stream (ODDS)

Wexner Medical Center

- Stream premises
  - The number of data buffers at each filter copy should be only enough to keep its processors busy
  - Data buffers sent to a filter should maximize the performance of processors allocated to each filter instance
- Dynamic Queue Adaptation Algorithm
  - Executed at the receiver side
  - Calculates the number of data buffers necessary to keep processor busy
    - Ratio of request response time to data buffer processing time
- Data Buffer Selection Algorithm
  - Executed at the sender side
  - Sends the data buffer with the highest speedup to the requesting processor

Effect of runtime optimizations

Wexner Medical Center

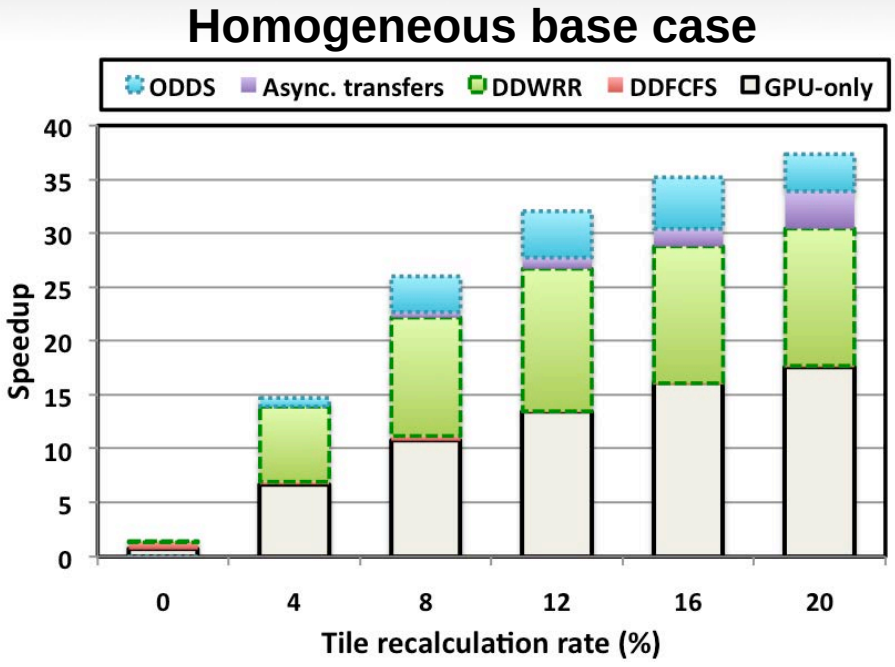
- Setup
  - Homogeneous cluster - 14 nodes w/ CPU/GPU
  - Heterogeneous cluster – 14 nodes
    - 7 use both the CPU and the GPU
    - 7 nodes do not use the GPU
- Stream policies

Demand-driven scheduling policy	Area of effect	Queue Policy		Size of request for data buffers
		Sender	Receiver	
DDFCFS*	Intra-filter	Unsorted	Unsorted	Static
DDWRR	Intra-filter	Unsorted	Sorted by speedup	Static
ODDS	Inter-filter	Sorted by speedup	Sorted by speedup	Dynamic

*DDFCFS: Anthill's default "Demand-driven first-come, first-served

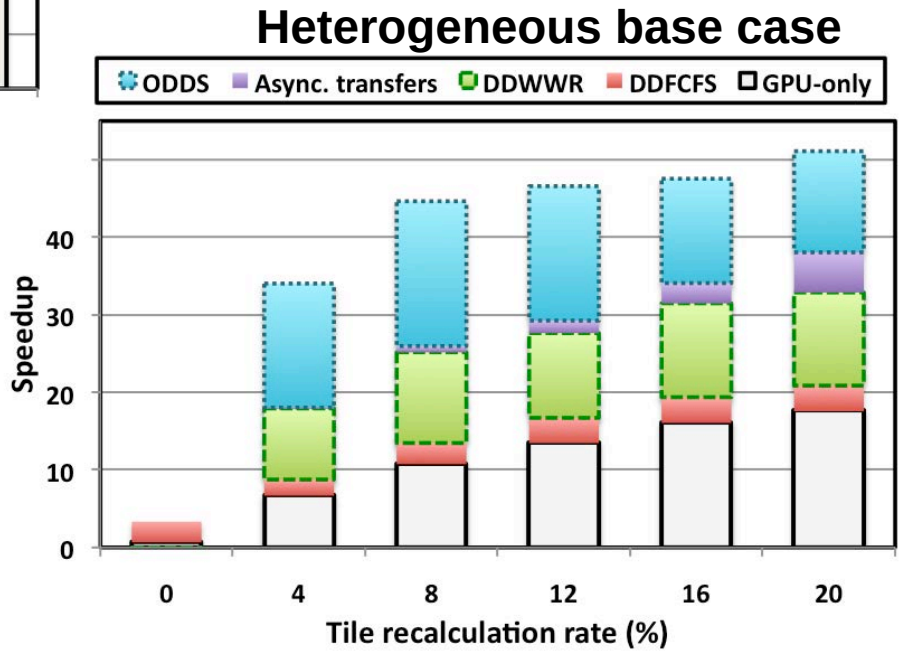
Effect of runtime optimizations

Wexner Medical Center



Tile recalculation rate: % of tiles recalculated at higher resolution.

ODDS improves performance even in the base case

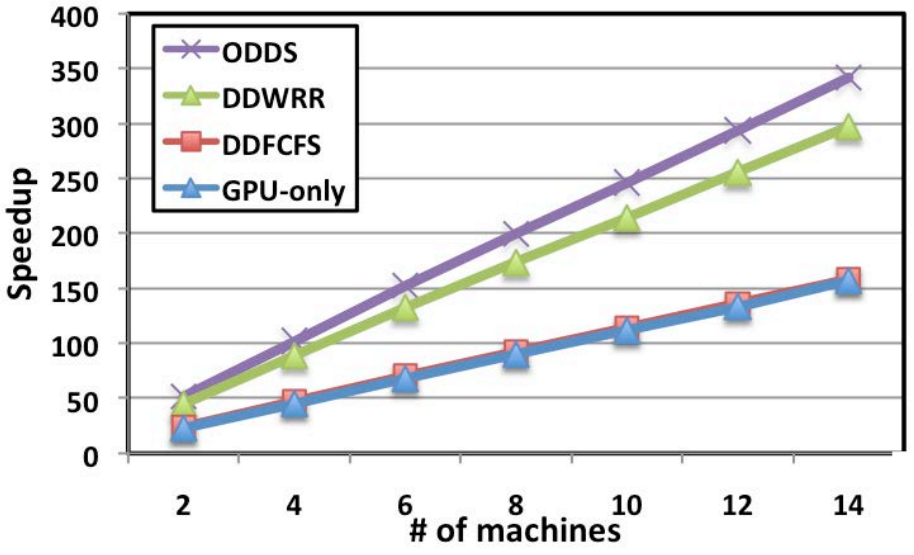


Using an additional CPU-only machine is more than 3x faster than GPU-only version

Strong Scalability

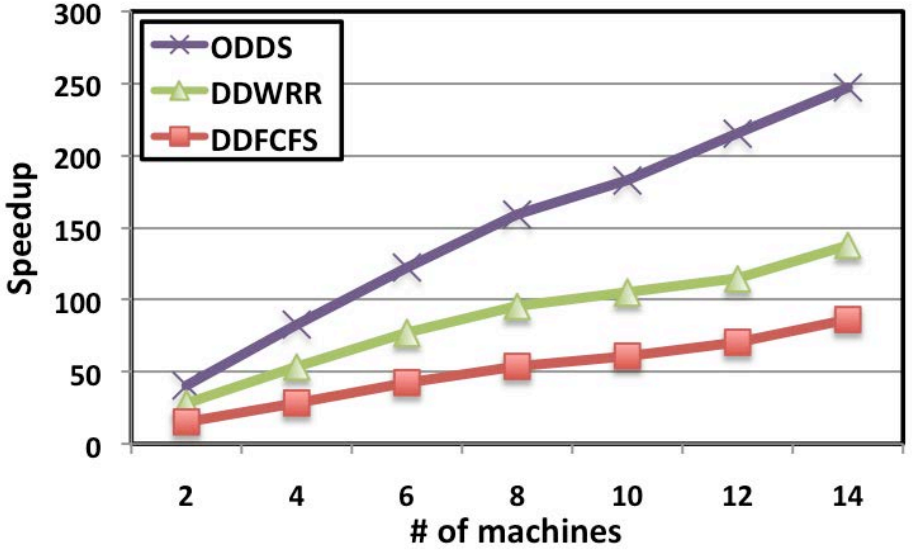
Wexner Medical Center

Homogeneous environment



All policies scale near to linear in homogeneous environments

Heterogeneous environment



Policies with static size of input requests fail to scale in heterogeneous environments

Wexner Medical Center

Adaptive Dataflow Middleware for *Finitely Divisible Load*

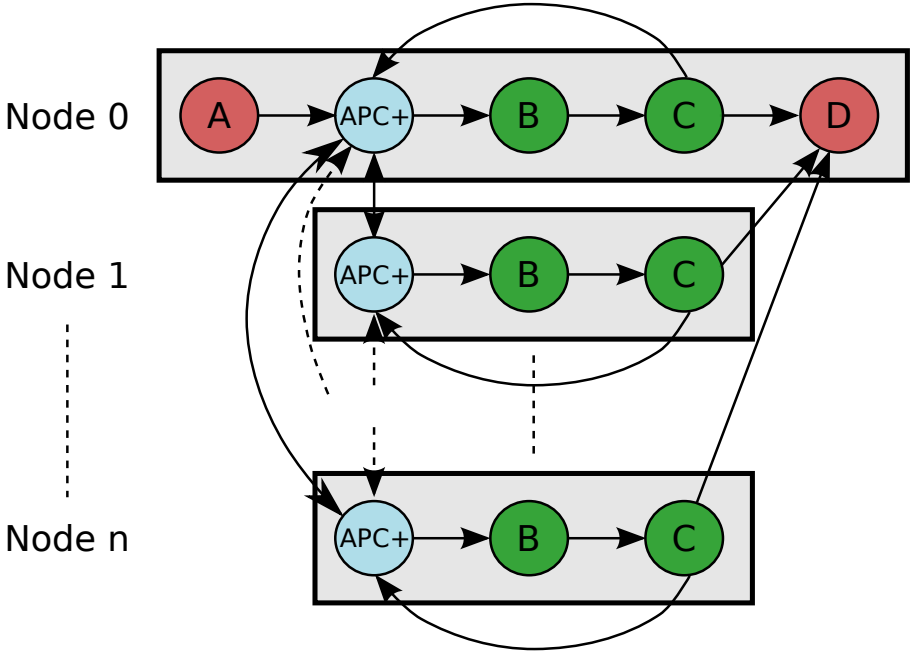
Adaptive Dataflow Middleware for Heterogeneous Systems

Wexner Medical Center

- Many real-world applications follow simple model
  - Independent tasks
  - Multi-dimensional (rectangular, cuboid etc.) *work area*
  - Finitely divisible (tiles)
    - Arbitrary partitions, within minimum and maximum constraints
- Even modern distributed middleware requires developers to perform manual tuning
  - Minimum execution time dependent on:
    - System configuration (# nodes, # CPUs / node, # GPUs / node, CPU speed, GPU speed, network speed)
    - Application specifics

A Simplified Application Model for *Finitely Divisible Load* Applications

Wexner Medical Center



- A component-based dataflow application
  - n processors
  - One copy of each filter pipeline on each processor
  - No communication between filters inside pipeline
  - Processors can be heterogeneous

Adaptive Partitioning Controller (APC+)

Wexner Medical Center

- Two concerns:
  - Choosing the best data buffer (work tile) size
    - Accelerators' performance highly dependent on amount of *work* per data buffer
    - Directly related to the application *work area*
    - Allow runtime system to automatically partition the *work area* into *tiles*
    - Keep track of processors' performance with performance model for each processor
  - Balancing the load
    - Use work-stealing to give processors appropriate work, such that all processors finish simultaneously

APC+ (cont'd)

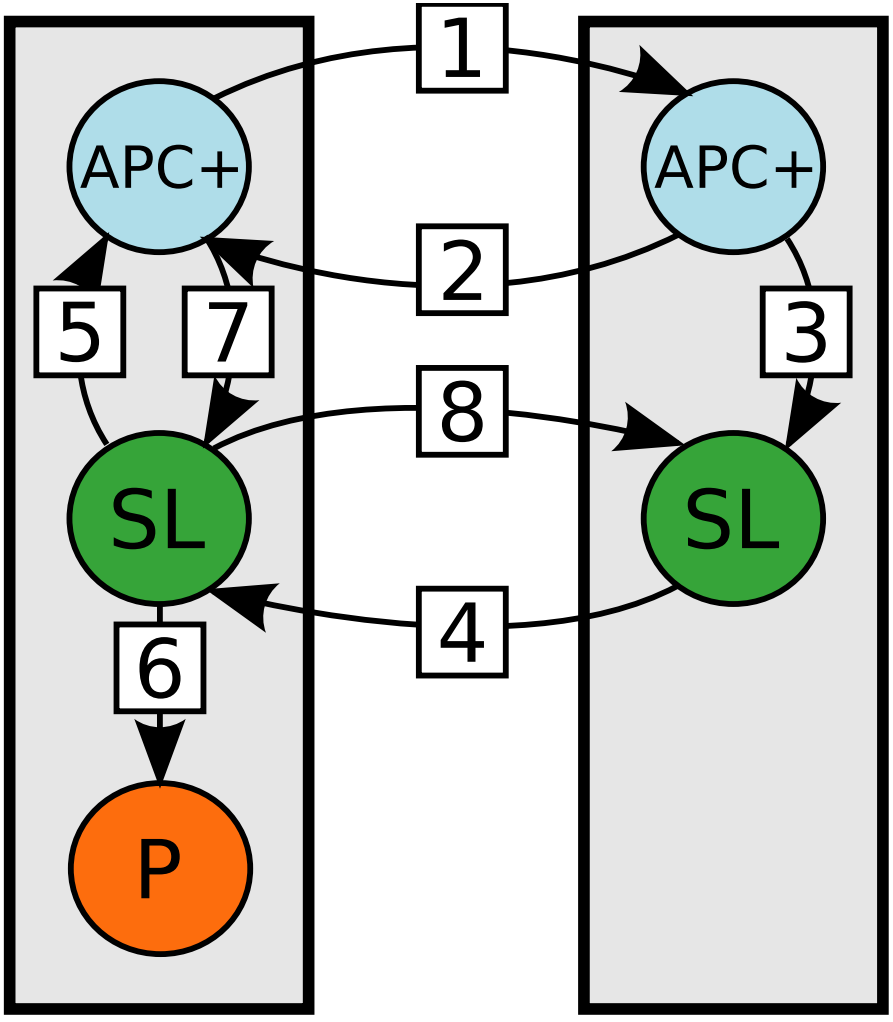
Wexner Medical Center

- Four parts
  - Performance Model
    - Tracks execution time performance for each data buffer size
    - Dynamically finds the highest-performing data buffer size
  - Work Partitioner
    - Dynamically partitions and schedules data buffers
  - Distributed Work-Stealing Layer
    - Lightweight work-stealing technique to balance the load across the nodes
  - Storage Layer
    - Enables efficient network use and separation of stealing of work and movement of data

APC+ Algorithm

- Three phases
  - Bootstrap
    - Initialize performance model by trying some data buffer sizes
  - Steady-state
    - Partition work area using maximum processing rates
    - Send work tiles which have the maximum processing rate
    - Keep processor queues full to hide latencies
    - Steal work from remote peer if none available locally
  - Flushout
    - If total work area is low, reduce tile sizes, for load balance

APC+: Work-stealing



- Storage Layer (SL), Processing Filter (P)
- Successful steal (1,2) schedules one data buffer for transfer (3)
- Upon receipt of the data buffer in the storage layer (4), the data buffer is unlocked (5), and scheduled for processing (6)
- Acknowledgement of receipt (7,8) schedules next data buffer

Experimental Setup

Wexner Medical Center

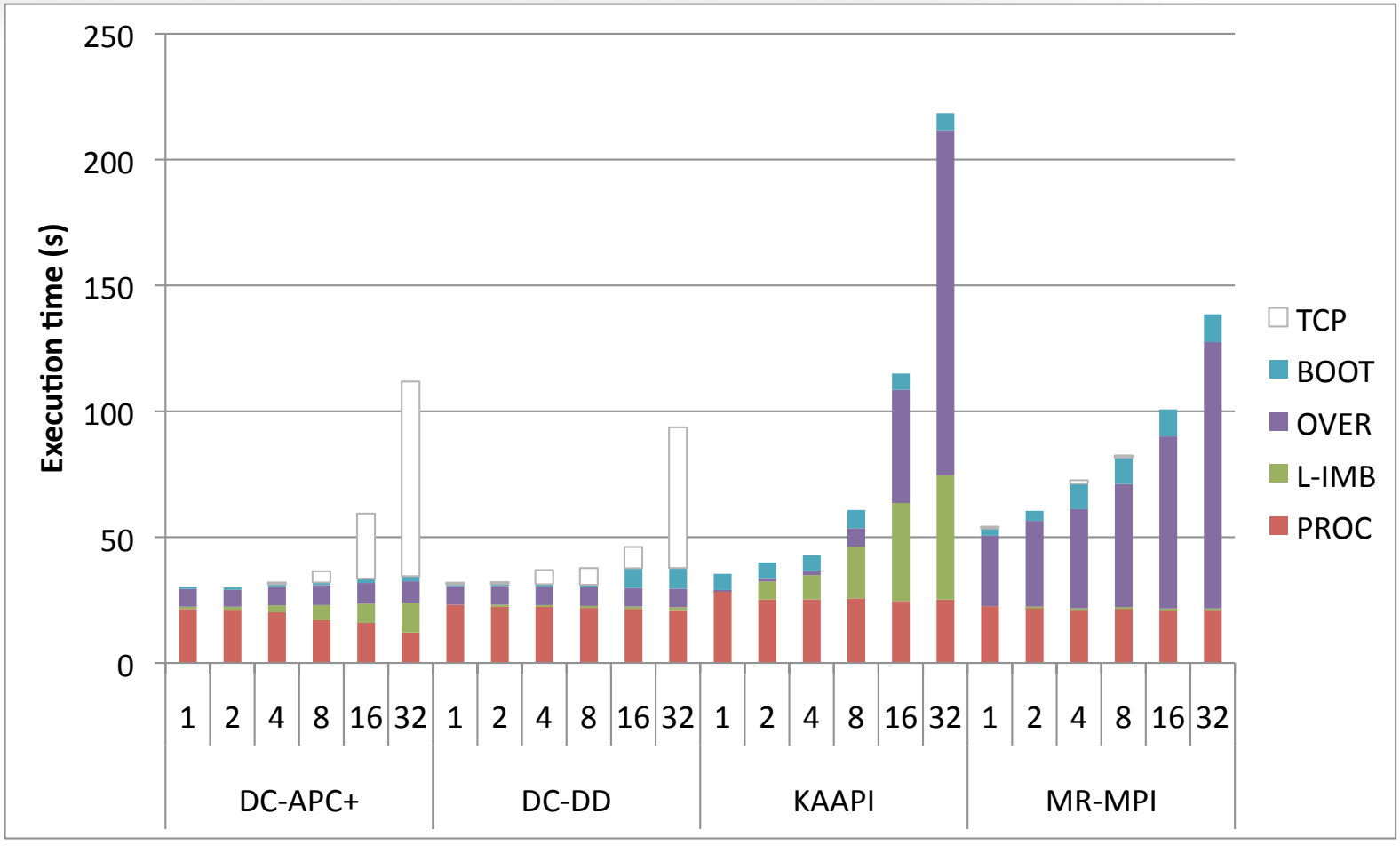
- Real-World Applications
  - Synthetic Aperture Radar Image Analysis
  - Histopathology Image Analysis
  - Black-Scholes European Stock Option Pricing Calculation
- Conduct weak scalability experiments
  - Increase work commensurate with increased computational resources
- *Owens* cluster
  - 32 nodes, dual quad-core Intel Xeon CPUs, single Nvidia C2050 GPU
  - 20Gbps Infiniband connected

State-of-the-art Distributed Middleware

Wexner Medical Center

- DataCutter
  - Demand-Driven load balancing (DC-DD)
    - Requires manual tuning
  - Simple, fast
- KAAPI/Athapascan-1
  - Work-stealing load balancing; scheduling with dependencies
    - Requires manual tuning
  - Theoretically asymptotic optimal behavior, in certain conditions
  - KAAPI has no native IB protocol
- MR-MPI
  - Simple MapReduce programming semantic
    - Requires manual tuning
  - Small API, no parallel programming knowledge is required

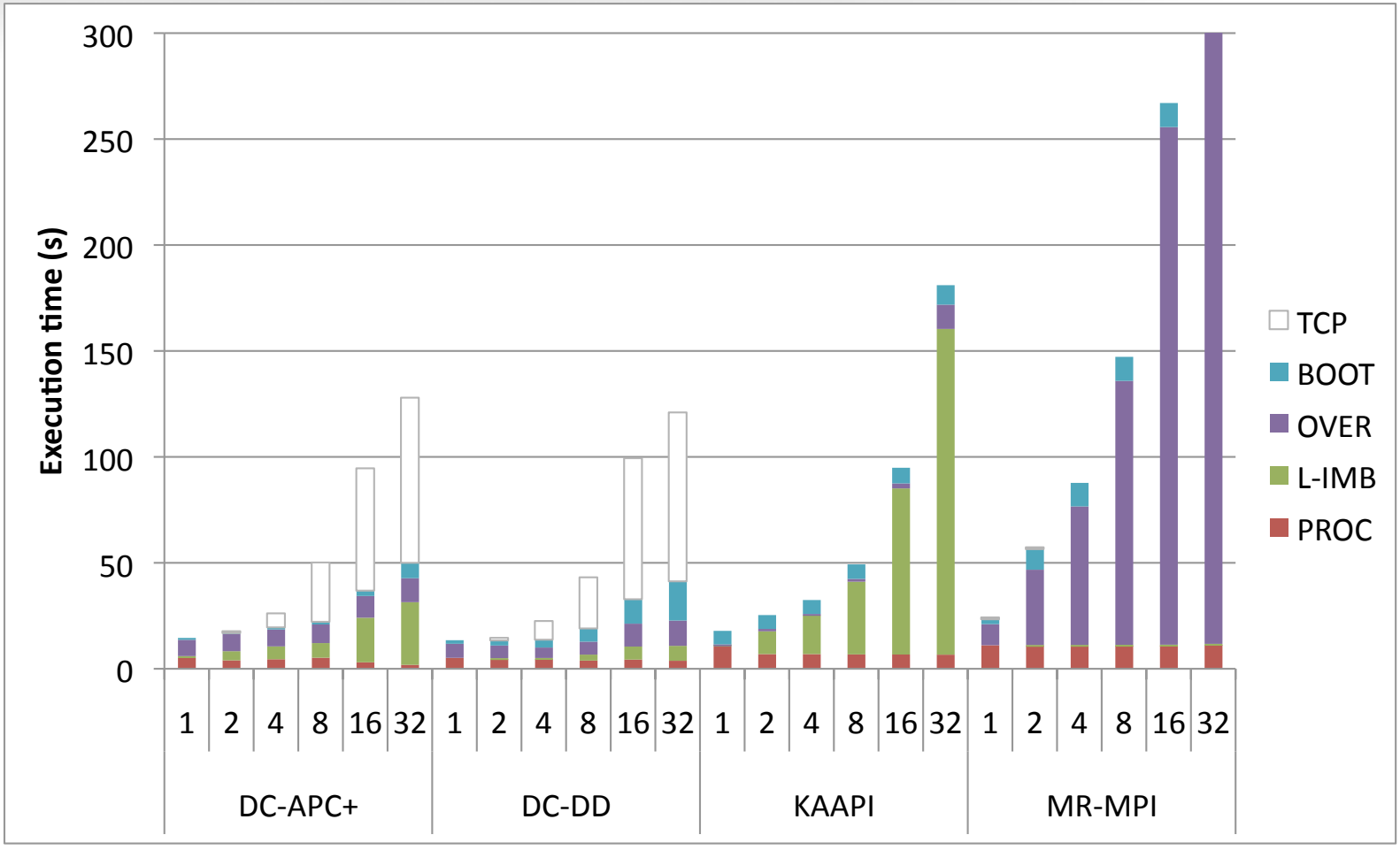
Synthetic Aperture Radar: CPU+GPU



- KAAPI perf. due to network bottleneck (16 thr./node)
- MR-MPI slow-down due to synchronization

Histopathology Image Analysis: CPU+GPU

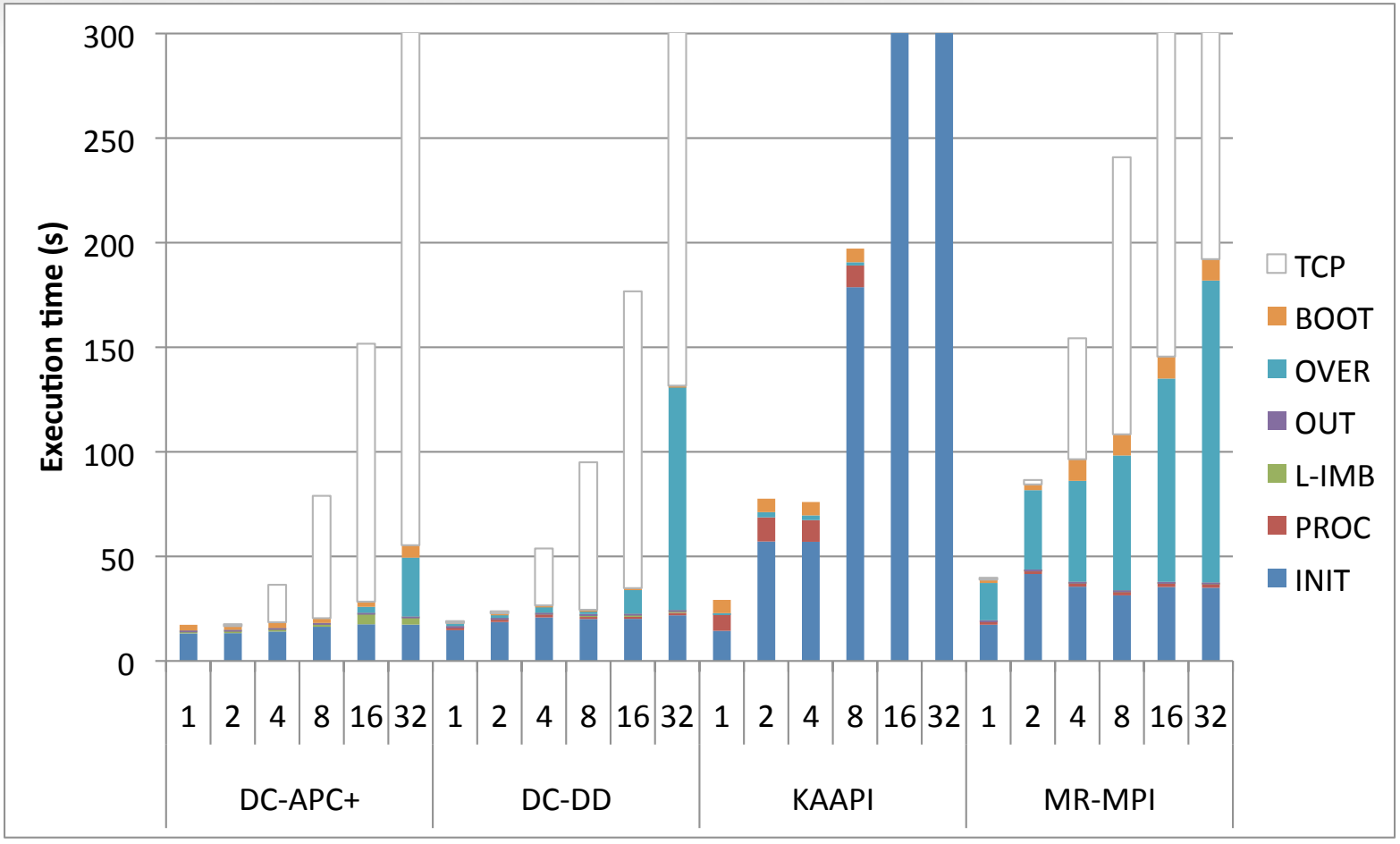
Wexner Medical Center



- Endpoint contention on node 0: cannot feed whole cluster sufficiently, especially using TCPoIB

Black-Scholes: CPU+GPU

Wexner Medical Center



- Data-intensive reduction to one node; KAAPI cannot pin distributed data read tasks; MR-MPI: no comm. overlap

Summary

- Dataflow systems are easy to program and good for efficiently using heterogeneous systems
- Dynamic runtime system optimizations were useful to utilize all computing resources
  - Don't neglect CPU cores even if you have powerful accelerators
  - Adequate use of CPU/GPU doubled GPU-only performance in several scenarios
- APC+ does a good job of application tuning and load balancing
  - Meets or beats well-tuned demand-driven policy
- There is always future work
  - Expand application and work area models for APC+

Lessons Learned

Wexner Medical Center

- Traditional low emphasis on performance – major end to end performance issues

- For real high-performing data-intensive applications
  - Dynamic resource management
  - Data-centric computing
    - Treat data as first-class citizen
    - Move work to data when needed
  - Global dependency graph (DAGs?) *exposed* to runtime system
  - Smart work-queues (*distributed work-stealing*)
    - Separate work-stealing and data transfer
  - Asynchronous execution (*pipelined execution*)

- Not surprising, not "much" different than others' recommendations for future roadmap to Exascale

Truth Telling

- Large scale data is a great buzz word but great need to systematically nail down user requirements now that we have plausible data systems with fast Hard disks and SSDs
- End to end performance characterization is rare and valuable. Traditionally hindered by shared nature of storage systems

- Claim: "I can do all in MPI+X+Y in my application!"
  - Sure you can! How do you think we implemented the runtime system!
  - But do you want to re-do in **all** of your applications?

Collaborators

(listed with their affiliations when the work was carried out)

Wexner Medical Center

- The Ohio State University
  - Timothy Hartley, Erik Saule, Olcay Sertel, Jun Kong, Metin Gurcan, Barla Cambazoglu, Mike Gray, Shannon Hastings, Steve Langella, Scott Oster, Li Weng, Gagan Agrawal, Krishnan Sivaramakrishnan, Vijay Kumar, Benjamin Rutt, Tony Pan, Tahsin Kurc (@Emory), Joel Saltz (@Emory)
- University of Maryland, College Park
  - Alan Sussman, Michael Beynon, Chialin Chang, Henrique Andrade
- Ohio Supercomputer Center
  - Don Stredney, Dennis Sessanna
- UMFH, Brazil:
  - Renato Ferreira, George Teodoro
- University of Malaga, Malaga, Spain
  - Manuel Ujaldon, Antonio Ruiz
- University Jaume I, Castellon, Spain
  - Rafael Mayo, Francisco Igual
- Rutgers University
  - Manish Parashar
- University of Texas at Austin
  - Mary Wheeler, Hector Klie, Paul Stoffa, Mrinal Sen, Malgorzata Peszynska, Ryan Martino

Thanks

Wexner Medical Center

- For more information
  - Email umit@bmi.osu.edu
  - Visit http://bmi.osu.edu/~umit or http://bmi.osu.edu/hpc

- Research at the HPC Lab is funded by