Building an Infrastructure to Support Massive Computational Resource Requirements

John J. Tran

Information Sciences Institute / USC

Marina del Rey, CA 90292

Agenda

- Two Approaches
  - Linux Clustering Solution
  - Specialized Hardware
- Motivation
- Background
- Findings & Results
- Applications to Computational Finance
- Forward and future thoughts

Dual-Approach

- Scaling computing resources by increasing cluster size
  - Homogeneous cluster in heterogeneous environment
  - Meeting all requirements of distributed, high performance, and scalable computing
  - Secret Ingredient is virtual machines (VM)
- Use commodity computation accelerators (when possible)
  - GPGPU or general purpose graphics processing unit
  - SIMD architecture – massively parallel
  - Readily available – not so easy to program BUT performance is virtually unlimited

Approach #1

Motivation

- Flexible development and deployment
  - Multiple OS's, multiple platforms
- Uniformity across clusters or clouds of computers
  - Thin host ⇨ Fat guest
- Reduce costs and overhead
  - Support and maintenance
  - Upgrade is a breeze
- Improvements to security?
  - Really needs to be carefully studied and analyzed…
  - Isolation is certainly there

Virtualization Defined

- From *Wikipedia*

  Virtualization is a broad term that refers to the abstraction of computer resources.

  One useful definition is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple logical resources; or it can include making multiple physical resources (such as storage devices or servers) appear as a single logical resource."

- More practical definition:
  - Computing paradigm letting users run multiple OS's concurrently on the same HW platform
  - Virtualization is NOT emulation

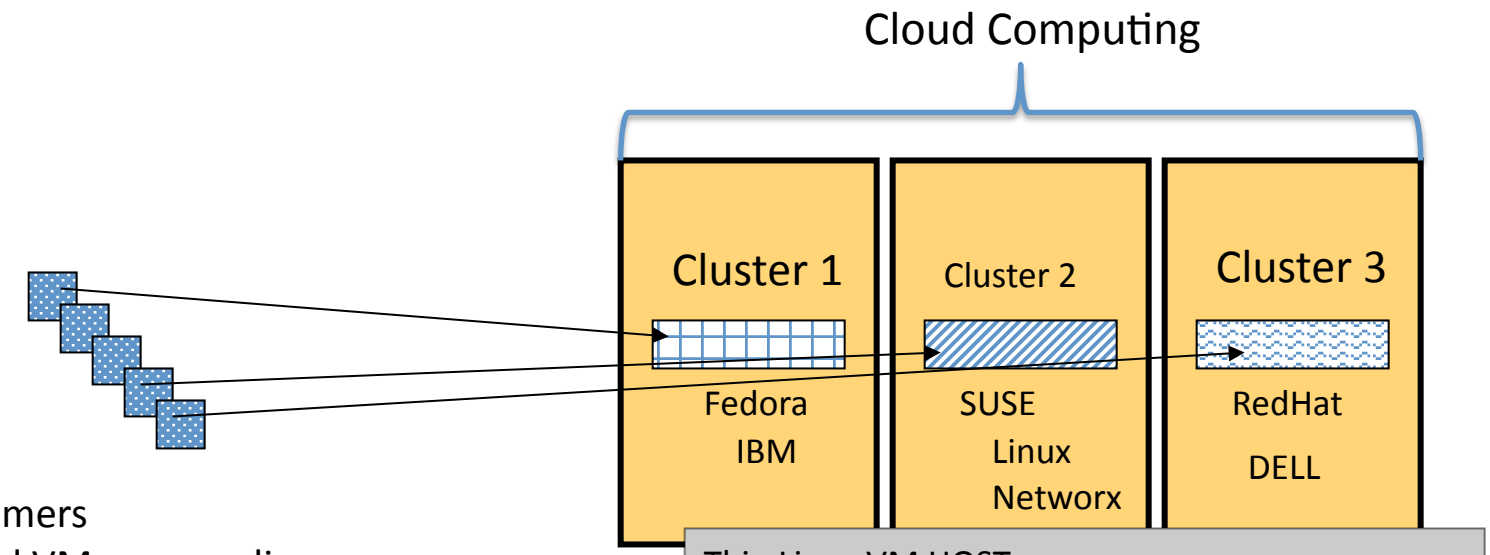
Virtualization Hierarchy

- **Emulation** or **Simulation** – virtual machine simulates the complete hardware (PearPC)
- **Native virtualization** or **Full virtualization** – simulates hardware to allow guest OS to run in isolation on host machine
- **Hardware-enabled virtualization** – machine-level hardware enabled for virtualization (e.g. Parallels for Duo Core 2 Macs running XP)
- **Partial virtualization** – host machine simulates subset of hardware; does not allow for completely separate OS, but allows for process isolation
- **Paravirtualization** – Host OS does not simulate hardware; guest OS is "severely" modified to make hypervisor calls to access the HW
- **Operating System-level virtualization** – analogue to chroot environment, but guest & host OS share the same OS
- **Application Virtualization** – application OS environment that lives inside the host OS (JVM)

Applications of Virtualization

- Server Consolidation
- Disaster recovery
- Testing and Training
- Portable Application
- Portable Workspace
  - Appliance – case study

Virtualization in Action



Customers
* Build VMware appliance
* Ship appliance to cluster
* Appliance is checked and queued
* Run appliance in hosted environment

Existing Virtualization Technologies

- Xen – open source virtualization platform
- VMware – commercial powerhouse*
- Parallels – desktop virtualization vendor (Macs only)
- VirtualBox – open source virtualization platform (Macs, Windows, Linux & Solaris)
- Microsoft Virtualization - Windows-based virtualization solution

Benefits and Drawbacks

- Benefits
  - Flexibility and Agile computing strategy
  - Scalable: from one desktop to huge clusters (hundreds if not thousands of nodes)
  - Can be very affordable: Xen & OpenBox are open-source
- Drawbacks
  - Performance penalty
  - Scalable – management procedures not well-defined
  - Can be expensive: VMware is $$

Preliminary Performance Analysis

- Pure IO, CPU benchmark
  - Classic Byte Benchmark (pure CPU)
  - Intel IO meter software test
- Hardware Platform for Real and Virtual Cluster
  - HP NetServer LPR 550 & 600 Mhz P3
  - 512MB memory
  - 2 x 9.1 SCSI disk
  - 10 nodes: node00..node10 non-virtual (also host OS) nodes
  - 10 nodes: node11..node20 virtual nodes

CPU benchmark

- Using the BYTE Benchmark
  - Well-studied and tested synthetic performance analysis tool

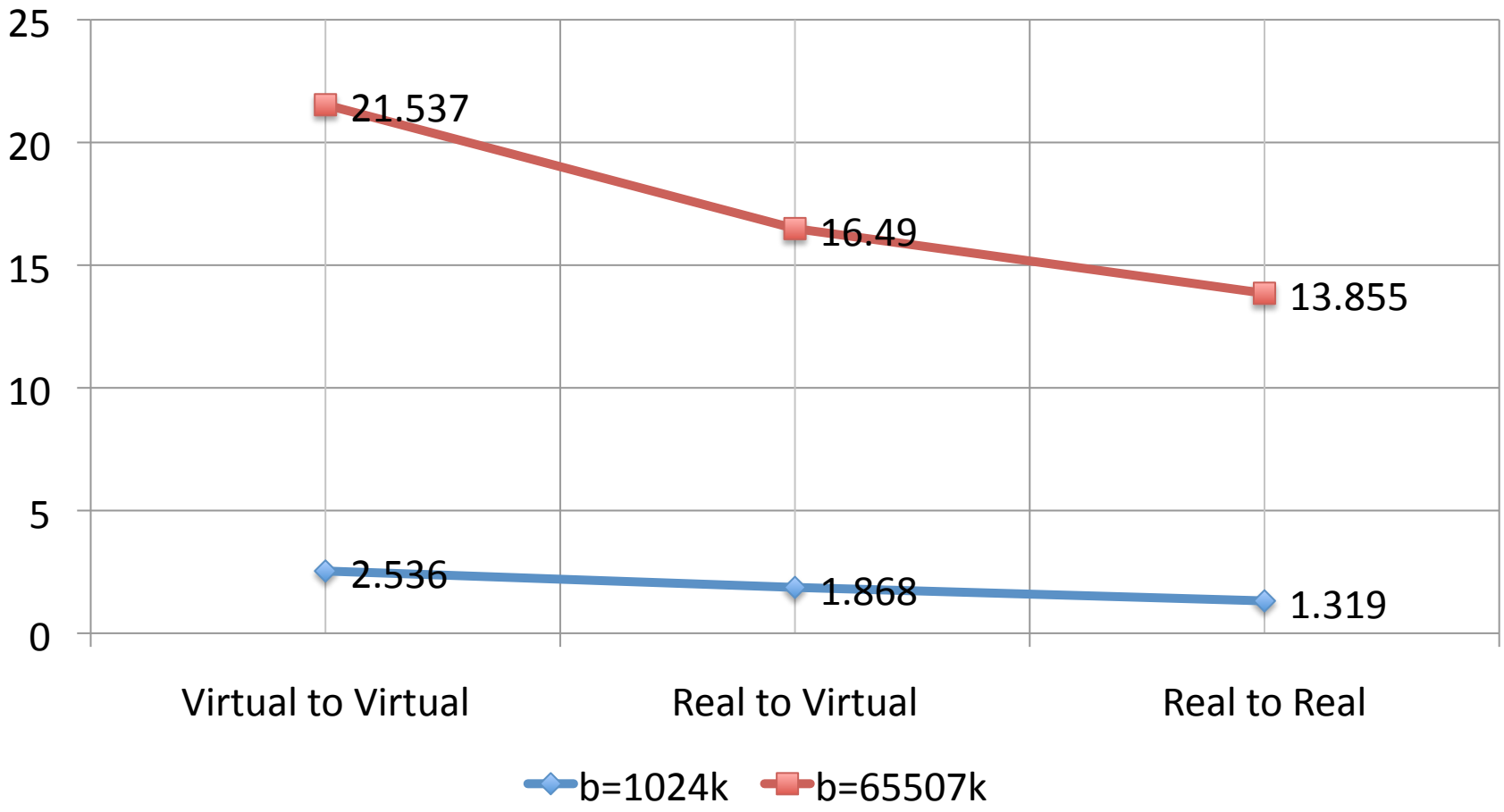
- Non-virtual node (iterations/sec)
  - Numeric Sort 247.28
  - String Sort 26.217
  - Bitfield 9.949E7
  - FP Emulation 35.315
  - Fourier 5854.8
  - IDEA 1098.3
  - Huffman 359.2
  - Neural Net 6.8261
  - LU Decomposition 212.72

- Virtual node (iterations/sec)
  - Numeric Sort 224.8
  - String Sort 23.954
  - Bitfield 9.145E7
  - FP Emulation 32.802
  - Fourier 5532.6
  - IDEA 1012.4
  - Huffman 330.4
  - Neural Net 6.2402
  - LU Decomposition 187.29

Network IO Comparisons

- Network IO – using simple ping program
  - ping -c10 -s 1024 from virtual to virtual node
    - rtt min/avg/max/mdev = 1.426/2.536/3.442/0.581 ms
  - ping -c10 -s 1024 from real to virtual node
    - rtt min/avg/max/mdev = 1.249/1.868/2.494/0.552 ms
  - ping -c10 -s 1024 from real to real node
    - rtt min/avg/max/mdev = 0.692/1.319/1.753/0.511 ms
  - ping -c10 -s 65507 from virtual to virtual node
    - rtt min/avg/max/mdev = 16.071/21.537/43.094/8.225 ms
  - ping -c10 -s 65506 from real to virtual node
    - rtt min/avg/max/mdev = 15.527/16.490/19.580/1.073 ms
  - ping -c10 -s 65507 from real to real node
    - rtt min/avg/max/mdev = 12.877/13.855/14.485/0.513 ms

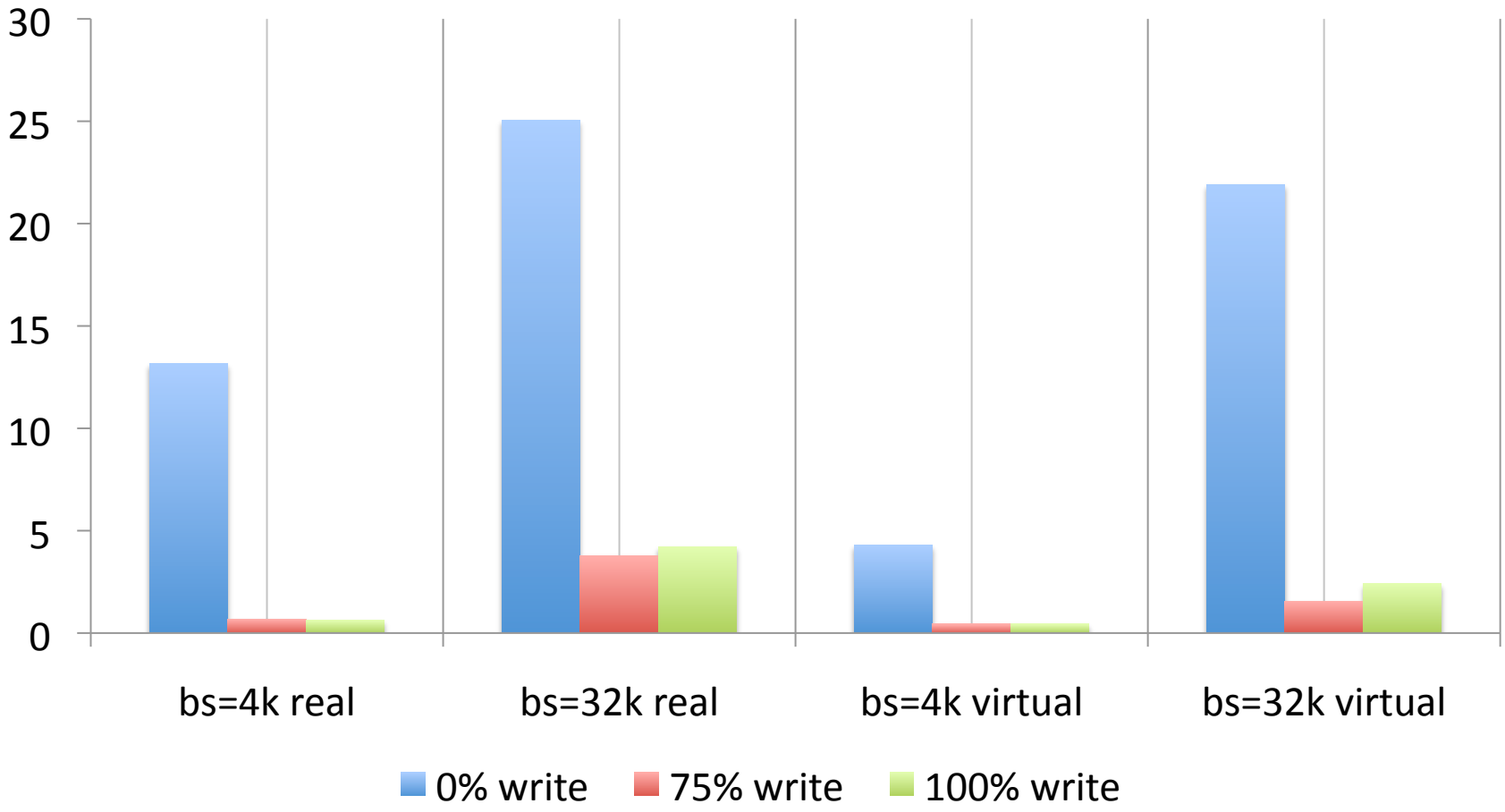
Average Ping Time (ms)



Disk IO Comparisons

- Numerical results (Mb/s)
  - 100% Write
    - Host OS 4k/32k => 0.62837/4.216848
    - Virtual OS 4k/32k => 0.441583/2.404602
  - 75% Write
    - Host OS 4k/32k => 0.694418/3.75723
    - Virtual OS 4k/32k => 0.434743/1.5625
  - 0% Write
    - Host OS 4k/32k => 13.16839/25.04023
    - Virtual OS 4k/32k => 4.290378/21.9113
- As expected, read operations are must better than writes
- Interesting to see that mixing reads and writes (75% write) yields worst performance than all writes (100% write)
- For large block sizes relative differences between real and virtual diminish

Disk IO Metrics (Mb/s)



Approach #2

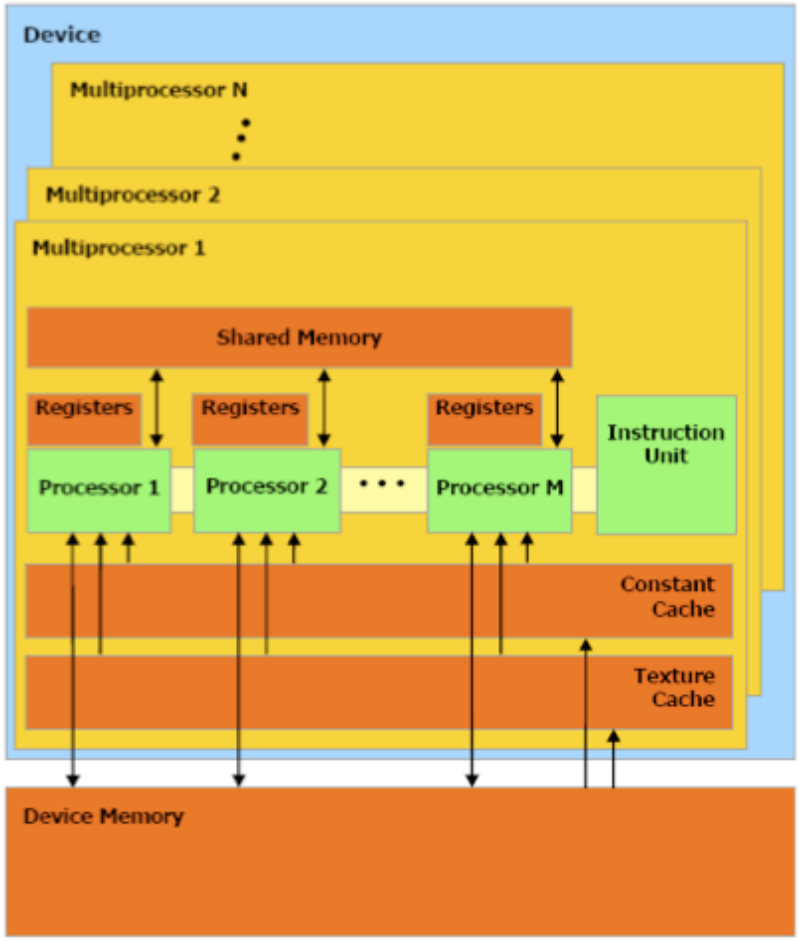
GPGPU Accelerator

- GPU performance can be 100x host performance. This differential is expected to grow
- Line of Sight (LOS) and Route Finding algorithms identified by Dinesh Manocha (UNC) and others
- ISI performed experiments to quantify CPU intensive algorithms as candidates for conversion to GPU
  - Measured performance of Line of Sight (LOS) and Route Finding algorithms
  - Preliminary work on route finding
- Candidate algorithms use large amount of time in small amount of code to enable conversion – Most bang for the buck!

NVIDIA GPU

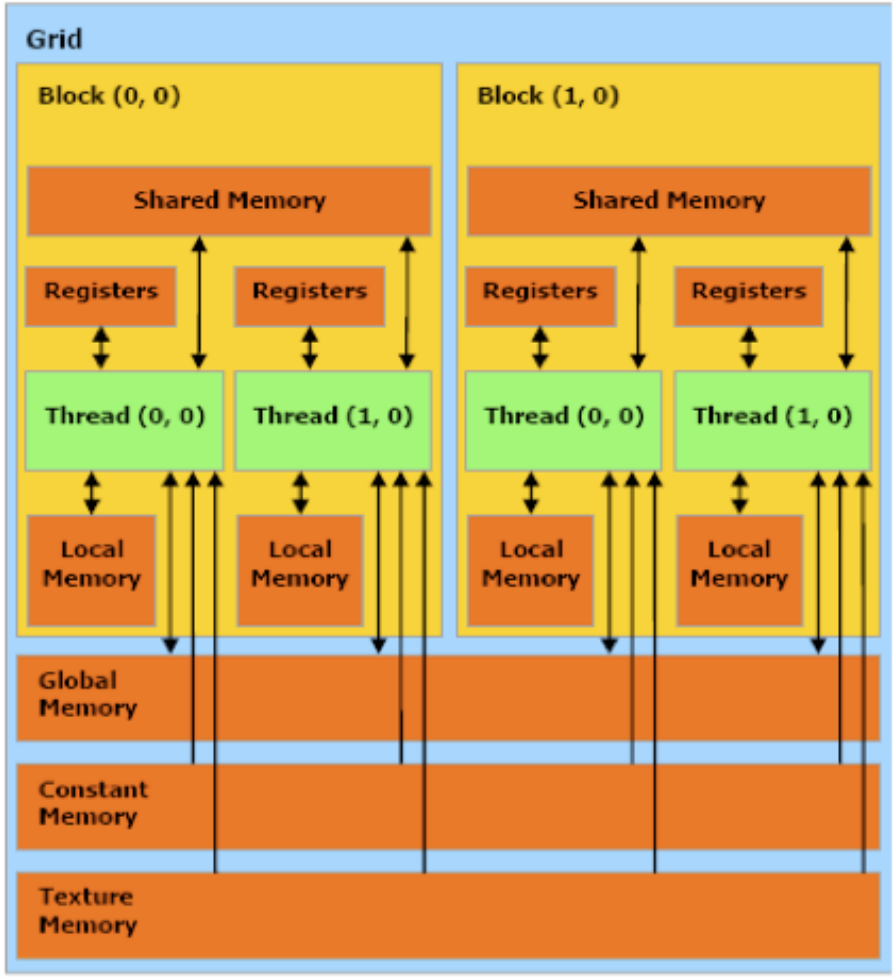
- Similar to CELL processor and other GPU's
- Hundreds of GIGAFLOPS single precision performance. Up to 100X speedup over host
- Performance differential expected to continue to grow
- Efficient libraries for linear algebra, FFT
- Supports CUDA

NVIDIA GPU Architecture



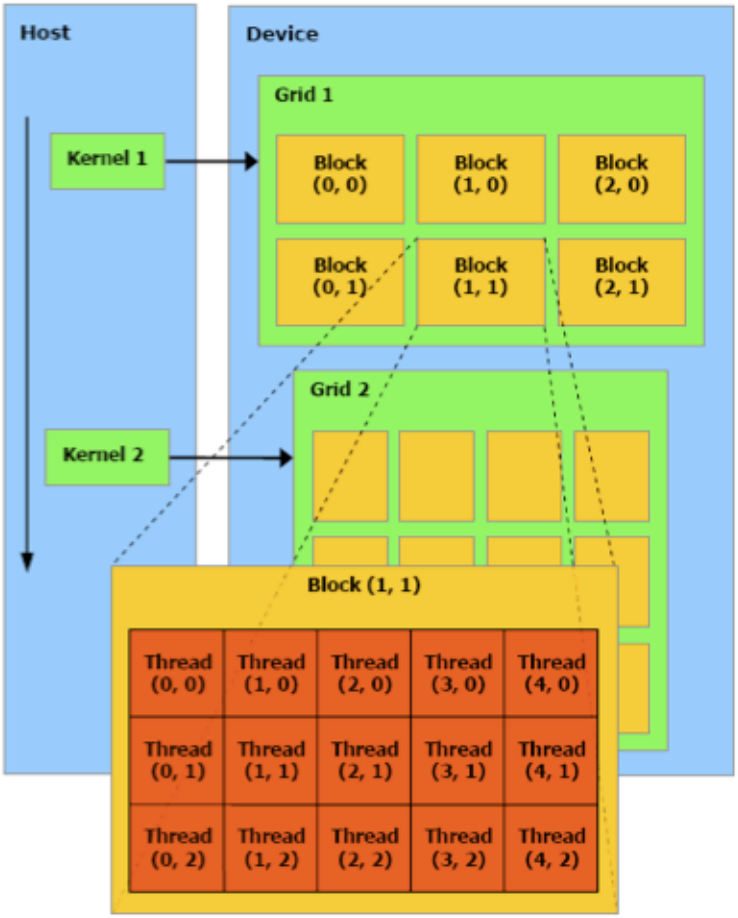
- Each GPU device has N multiprocessors
- Each multiprocessor has M processors
- Each processor has its own registers
- Processors in the same multiprocessor have access to shared memory

Memory Hierarchy



- A grid is a collection of blocks
- A block is a collection of threads on the same multiprocessor
- Each thread on the same processor has its own register and local memory
- Threads on the same processor share “Shared Memory”
- Threads on the same device share “Global Memory”

Host/Device Program Mapping

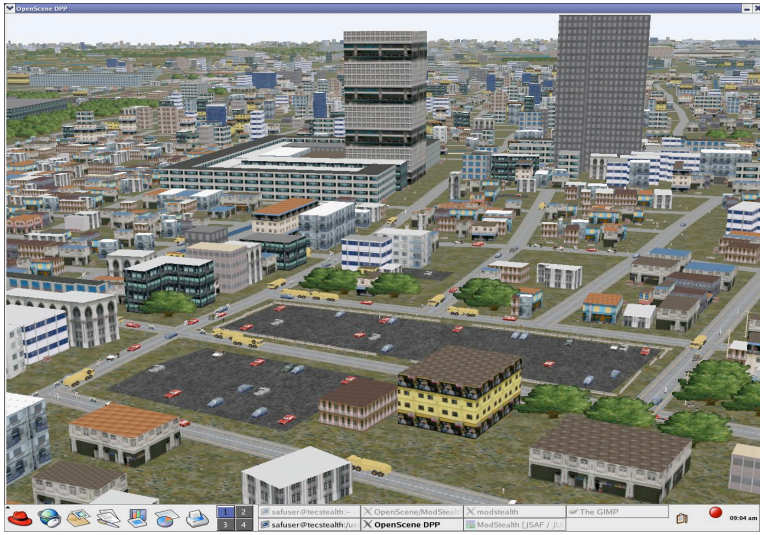


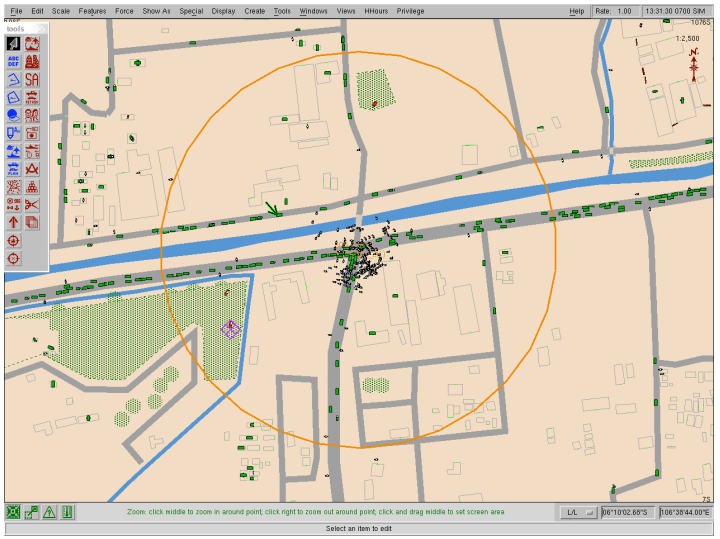
- Each "kernel" is a GPU "program"
- A program on a host can have multiple kernels
- Cannot have multiple programs on the host sharing GPU at the same time
- All programs access GPU by: (1) copy data from host to GPU (2) execute program on GPU, (3) copy data back to host

CUDA

- Programming on the GPU = SIMD programming
  - Same instruction on massively parallel data
- High level language supported by NVIDIA for current and future architectures
- No need to hand code low level language and rewrite every few years
- C language with GPU specific extensions
- Don't use OpenGL

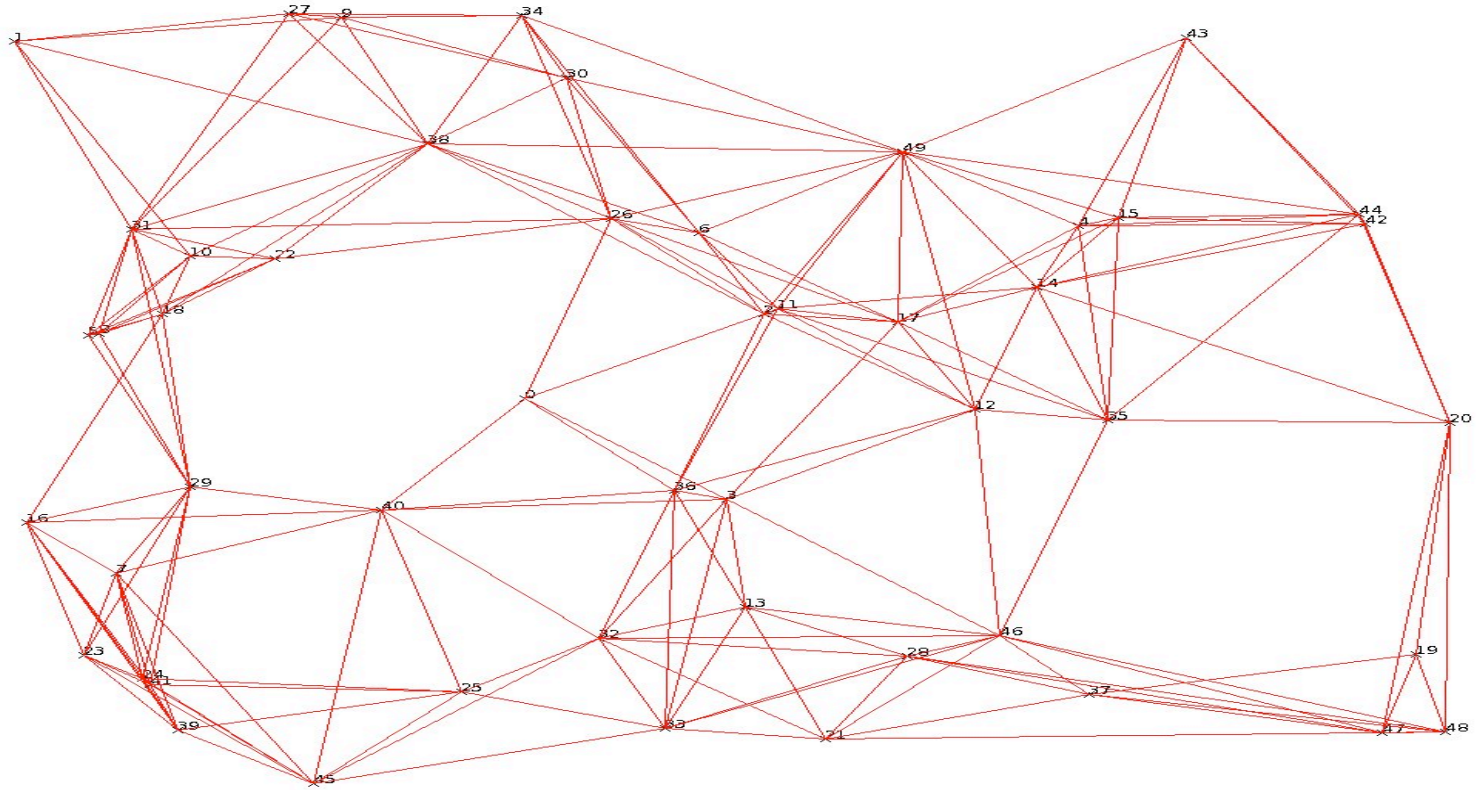
Case Study: Road Networks





- Complex urban setting
- Dense road networks

Map Problem to Graph Theory

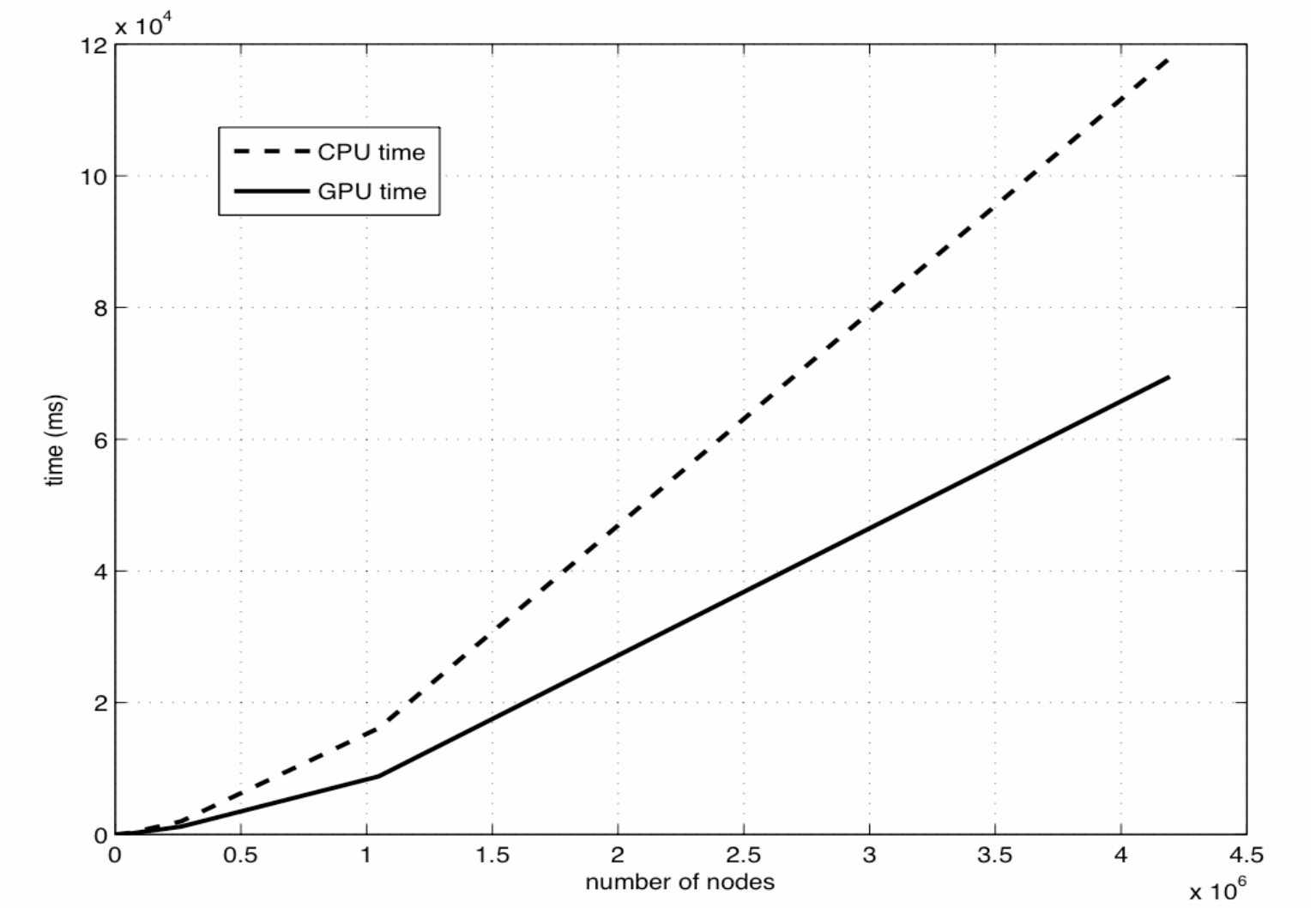


Network of roads = nodes and edges

Graph Problem Classification

Algorithm Models	Properties			
	Implementation Model	Connectivity Graph	Storage Size	O(Time Complexity)
A*	Serial	Priority Queue	N^2	N log N
MM	Serial & Parallel	Adjacency Matrix	N^2	N^3 log N
FW	Serial & Parallel	Adjacency Matrix	N^2	N^3
SSSP	Parallel	Adjacency List	N^2	N log N
ASSP	Parallel	Adjacency List	N^2 * M	N^2 log N

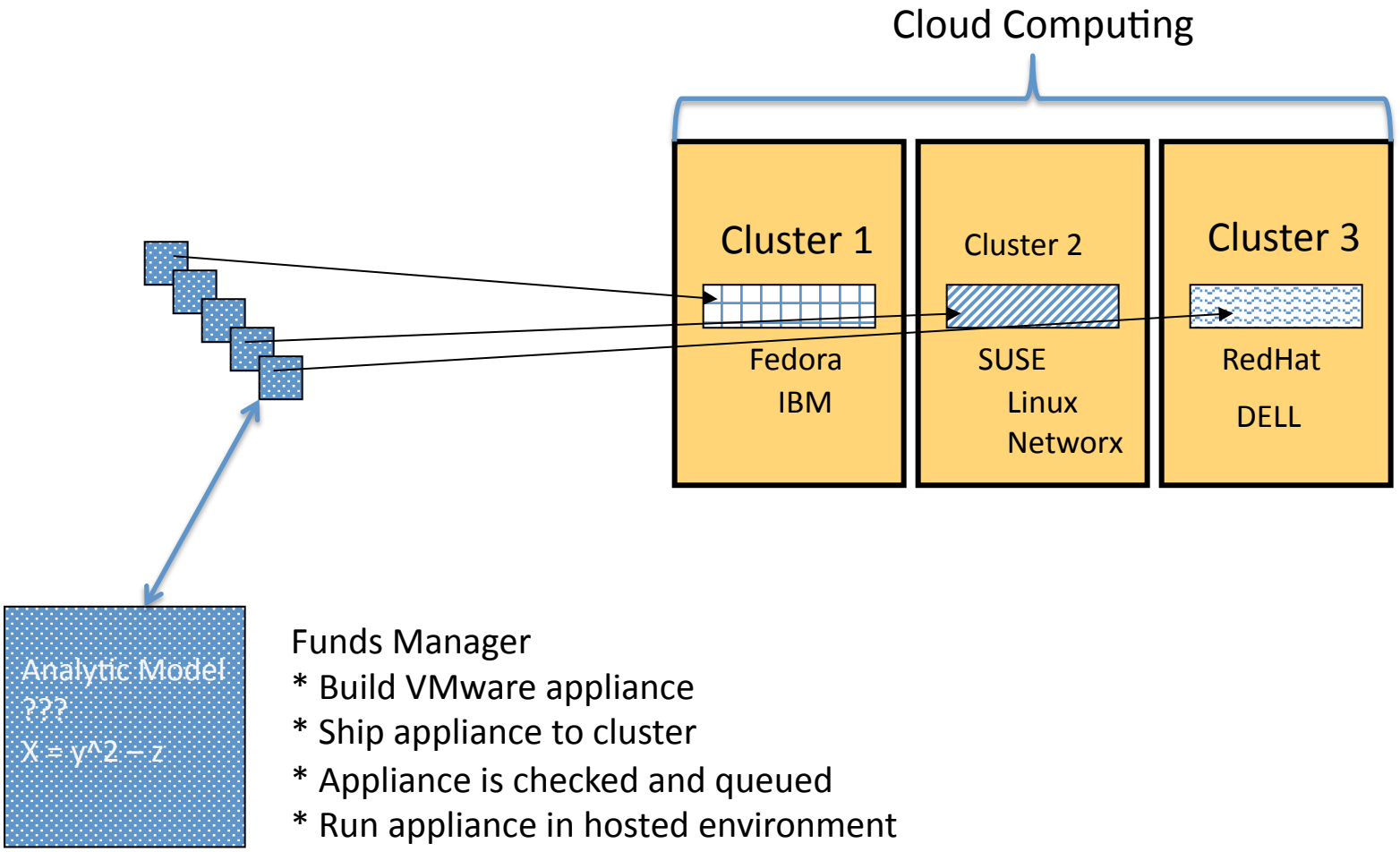
GPU vs. Non-GPU Timing



Lessons Learned

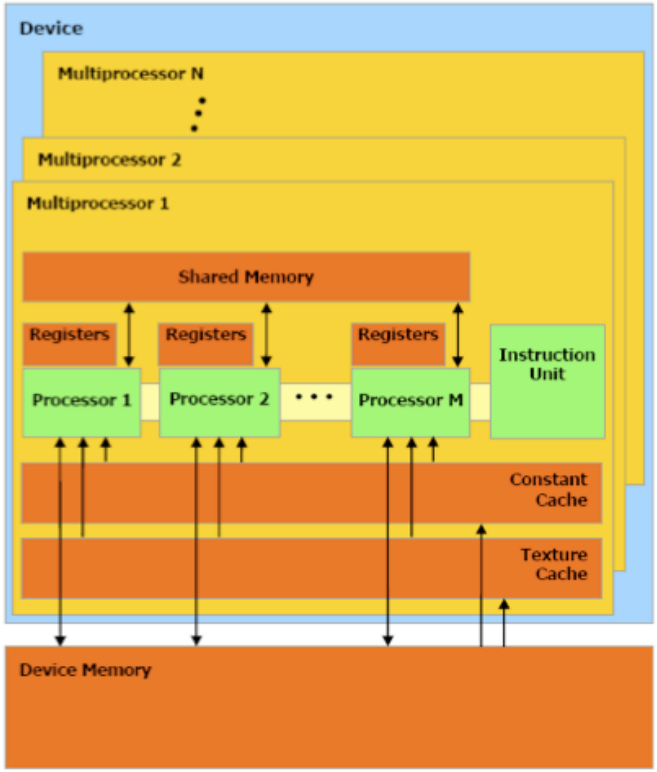
	Practical	**Performance**
All-to-All (MM)	GPU – Not practical N is capped at 20k	GPU – O(N^3)/C C = 128
	CPU – Not practical N is capped at 20k	CPU – (N^3)/C C = 4
One-to-All (SSSP)	GPU – Practical N is 1 Million	N log(N)/C C = 128
	CPU – Practical N is 1 Million	N log(N)/C C = 4
All-to-All (ASSP)	GPU = yes practical N = 1M * # GPU	GPU - N^2 log(N)/C C = number of cores * 128
	CPU = yes not efficient however	CPU – N^2 log(N)/C C = number of cores

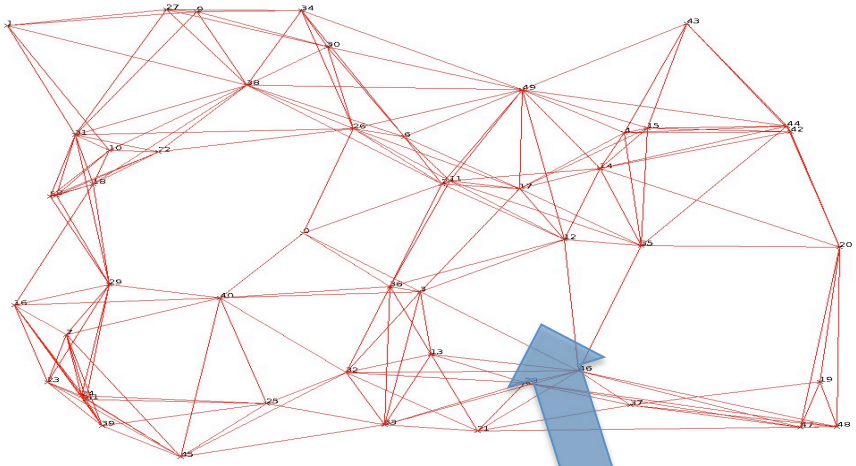
Application to CF



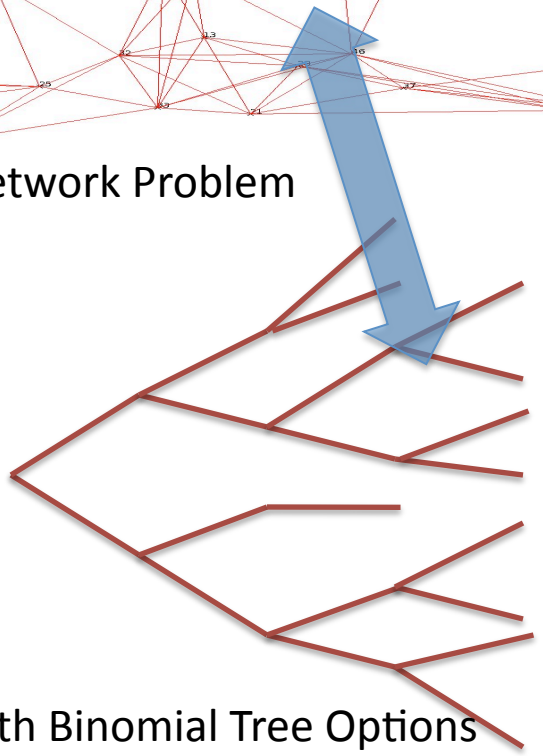
Funds Manager
* Build VMware appliance
* Ship appliance to cluster
* Appliance is checked and queued
* Run appliance in hosted environment

Application to CF





Graph Network Problem



n Levels
2^(n+1) - 1

N-path Binomial Tree Options

Uncharted Areas (future work)

- Approach #1
  - Xen virtualization
  - Security: defining policy and implications
  - Policy setting
    - Rules of engagement
  - Metrics of success
    - Transition from utilization & performance to broader customer base
  - Licensing issues – the case of Microsoft (or possibly Apple)
- Approach #2
  - Port CF algorithms to GPU
    - BM, BS and other Options Model
  - Provide generic programming interface accessible for all programming platforms

Acknowledgments

- Information Sciences Institute (ISI)
  - Gene Wagenbreth, Bob Lucas, Dan Davis, & Ke-Thia Yao
- University of Southern California (USC)
  - Chris Mattmann
- Jet Propulsion Laboratory (JPL)
  - Dan Crichton & Dana Freeborn
- IBM
  - John Doppke