System and Programming Overview

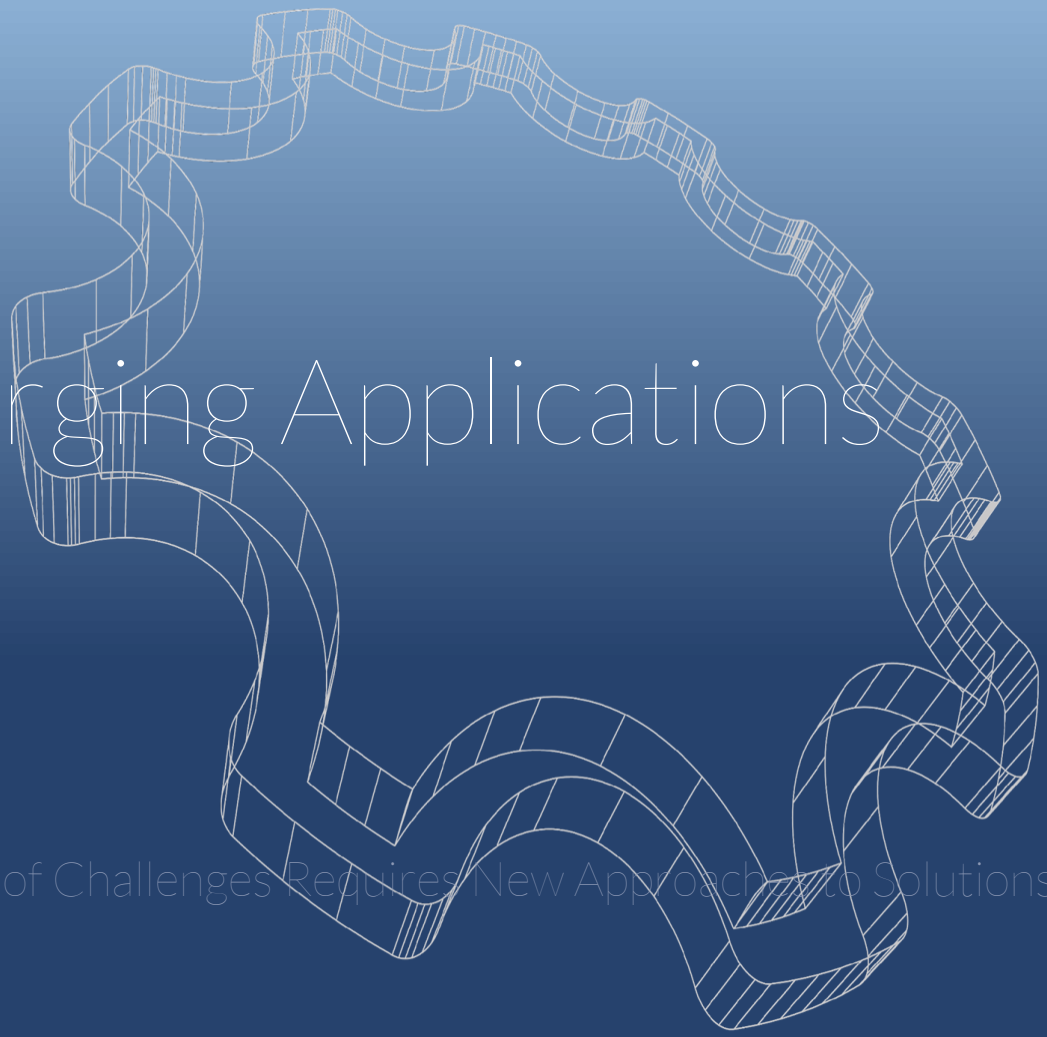
Janice McMahon

September 2022

Outline

- Emerging Applications
- Emu System Architecture
- Programming and Execution Model
- Software Support

Emerging Applications



Evolution of Challenges Requires New Approaches to Solutions

LUCATA

Lucata Mission

A fundamentally new solution to identify relationships within large, unstructured datasets without sacrificing programmer productivity

Large Graph Database Problems

- Distributed over many memories
- Data movement dominates performance
- Memory accesses are irregular, remote, & unpredictable

Traditional System Failure

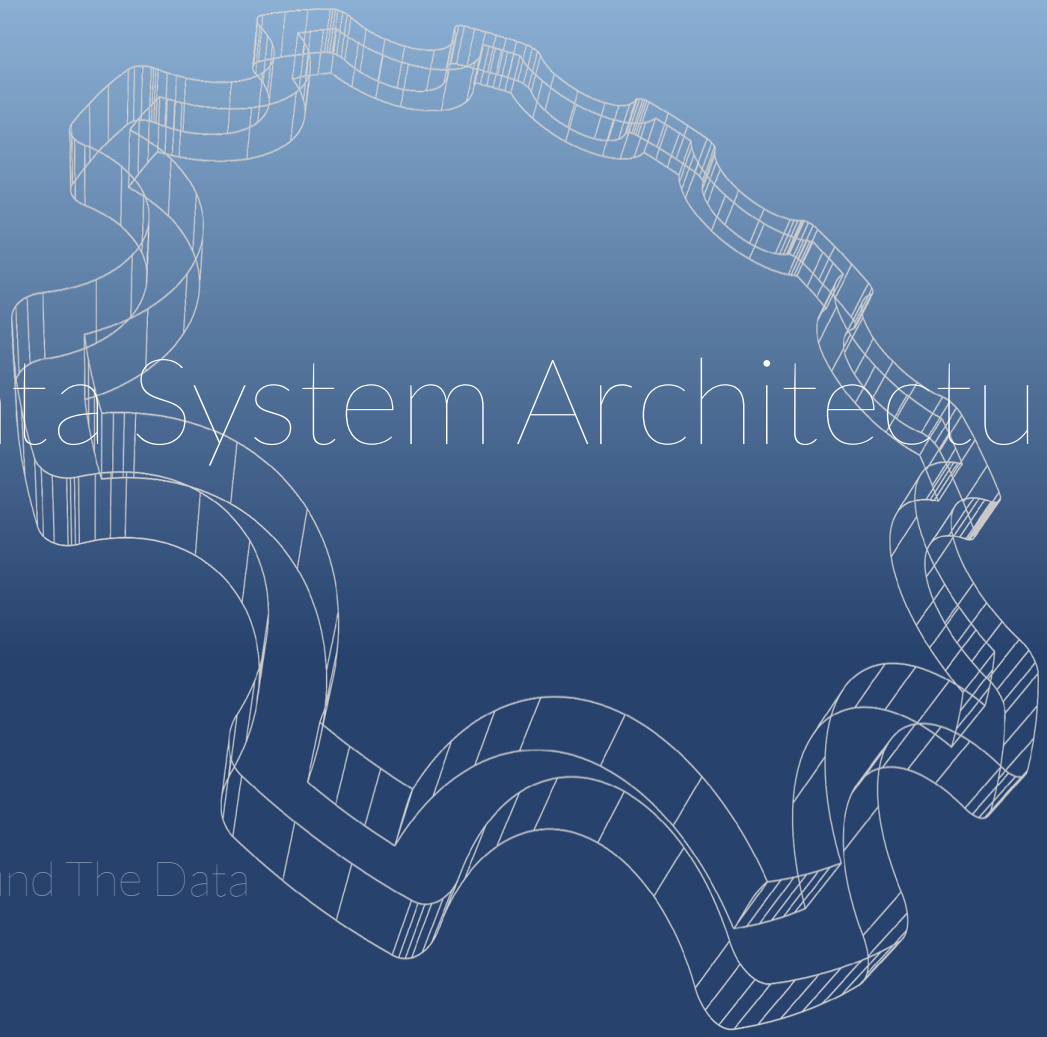
- Memory caches inefficient
- Interconnect bandwidth insufficient
- Power consumption unaffordable

Lucata *Context-Flow Architecture* designed to meet the needs of today's large graph database applications

Evolving Applications

Benchmark	Function	System Efficiency (% of peak)	
		Conventional	Lucata
LINPACK	Solve Ax=b, A is dense	>90%	>90%
GUPS	Random updates	~10%	90%
HPCG: High Performance Conjugate Gradient	Ax=b, A sparse but regular	~2%	50%
SpMV: Sparse Matrix Vector	AB; A sparse and irregular	~2% of peak	80%
BFS: Breadth-First Search (Graph500)	Find all reachable vertices from root	~2% of peak	60%
Firehose	Find “events” in streams of data	~1% of peak	95%
CC: Connected Components	Find disjoint subgraphs	~25% of peak	95%

Lucata System Architecture

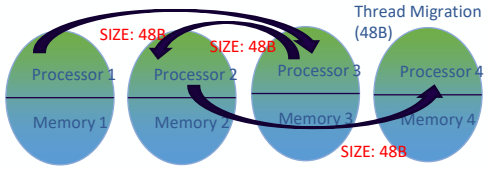


Built Around The Data

LUCATA

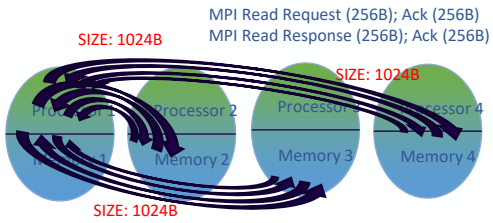
Lucata Innovation: Context Flow

Context flow with thread migration



VS.

Conventional computing with message passing



Remote memory access triggers movement (migration) of thread context to destination

- **Less data moved shorter distances**
- **Managed in hardware and invisible to programmer**
- **Improved processor utilization and simplified network design**
- **Lower energy cost and higher efficiency**

Enables fine-grain parallelism and high scalability for data analytics

Graph Processing / Random Memory Access on Lucata

Traditional Architectures

processors weren't designed for this!

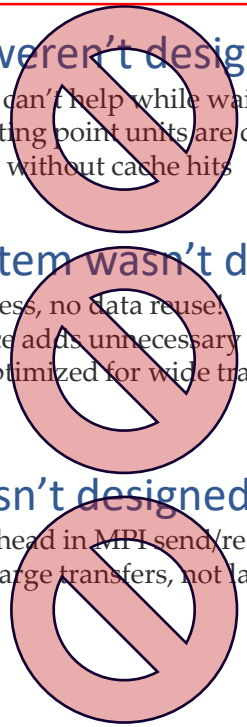
- High clock rate can't help while waiting on memory
- Vector and floating point units are dead weight!
- Can't stay busy without cache hits

memory system wasn't designed for this!

- Caches are useless, no data reuse!
- Cache coherence adds unnecessary complexity
- Memory bus optimized for wide transfers, wasteful!

network wasn't designed for this!

- Too much overhead in MPI send/receive
- Optimized for large transfers, not latency



Lucata Architecture

Hundreds of simple, multi-threaded cores execute thousands of threads to perform massively concurrent near-memory processing.

Cache-less shared-memory architecture. Multiple parallel DRAM with channels that perform advanced atomic memory operations at each memory controller providing nearly linear scalability

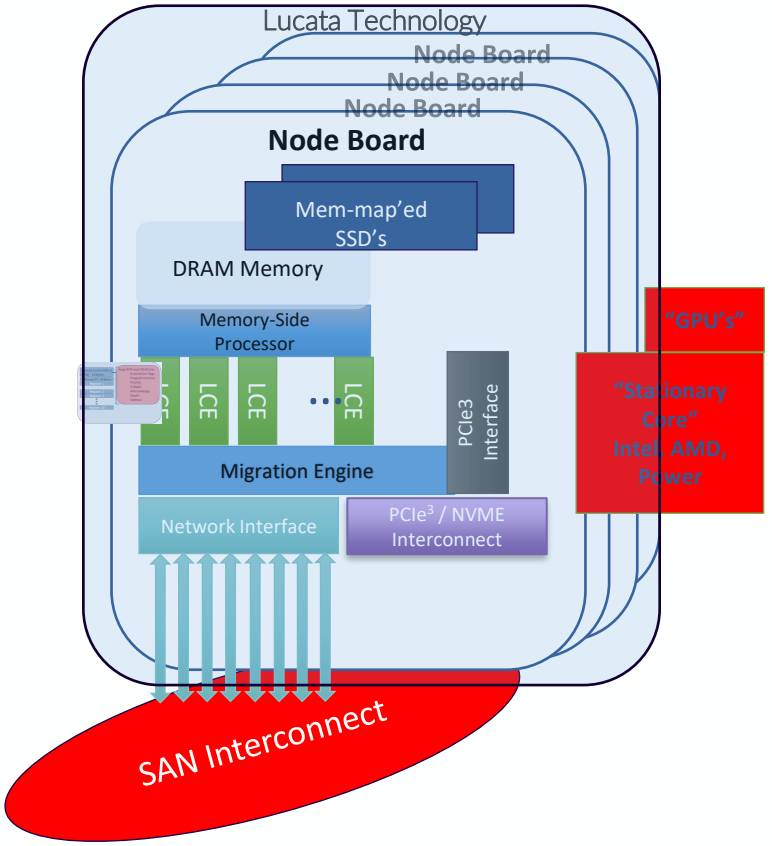
High bandwidth network: Threads migrate between nodes in the system. Lucata moves small thread contexts instead of large data transfers, reducing network bandwidth needs by over an order of magnitude.

Lucata Technology: Heterogenous System

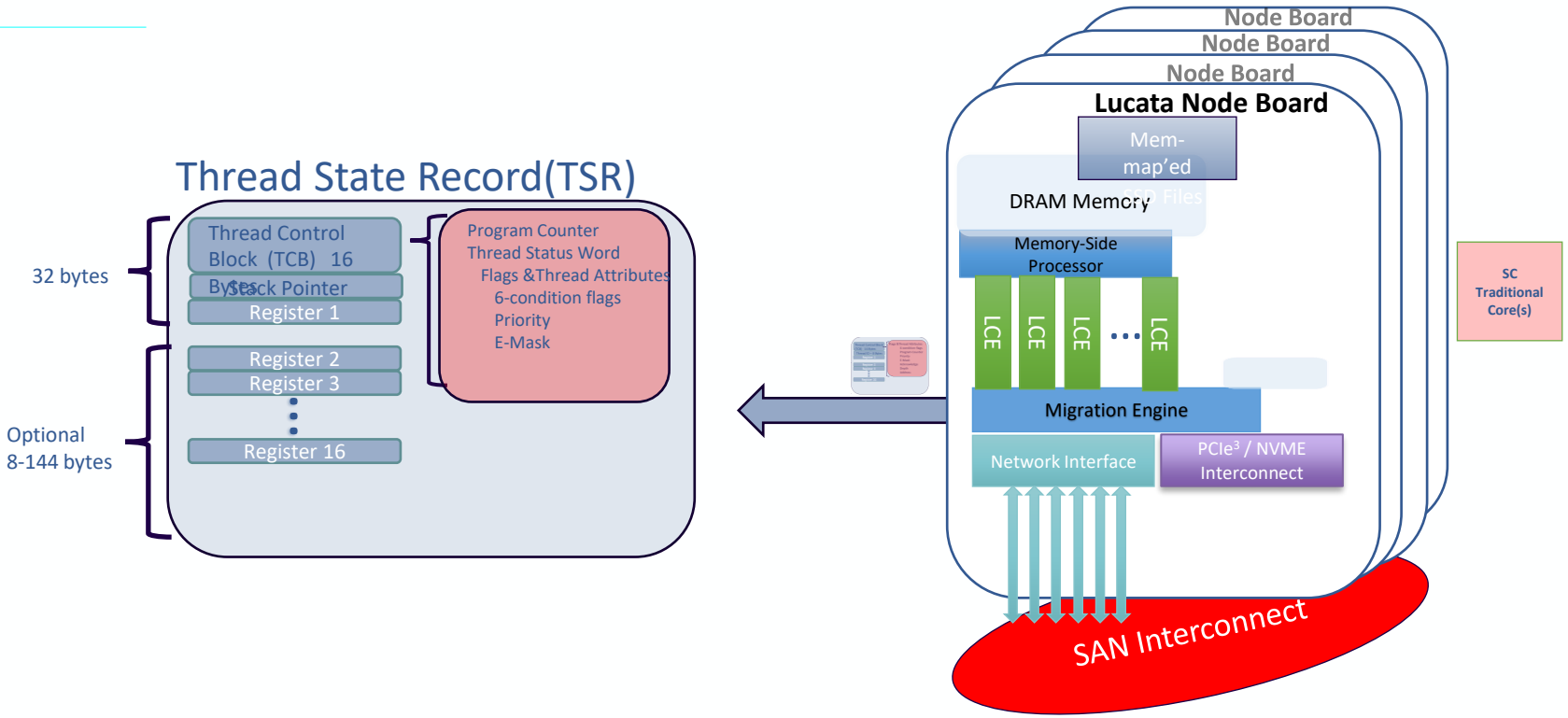
- Stationary Core (SC) runs Linux, performs I/O
- Database distributed across all shared memories

- Every memory read is a local access
- Thread context (TSR) migrates to data, leaving data in place
- Bandwidth only consumed by thread context movement and remote writes / atomics

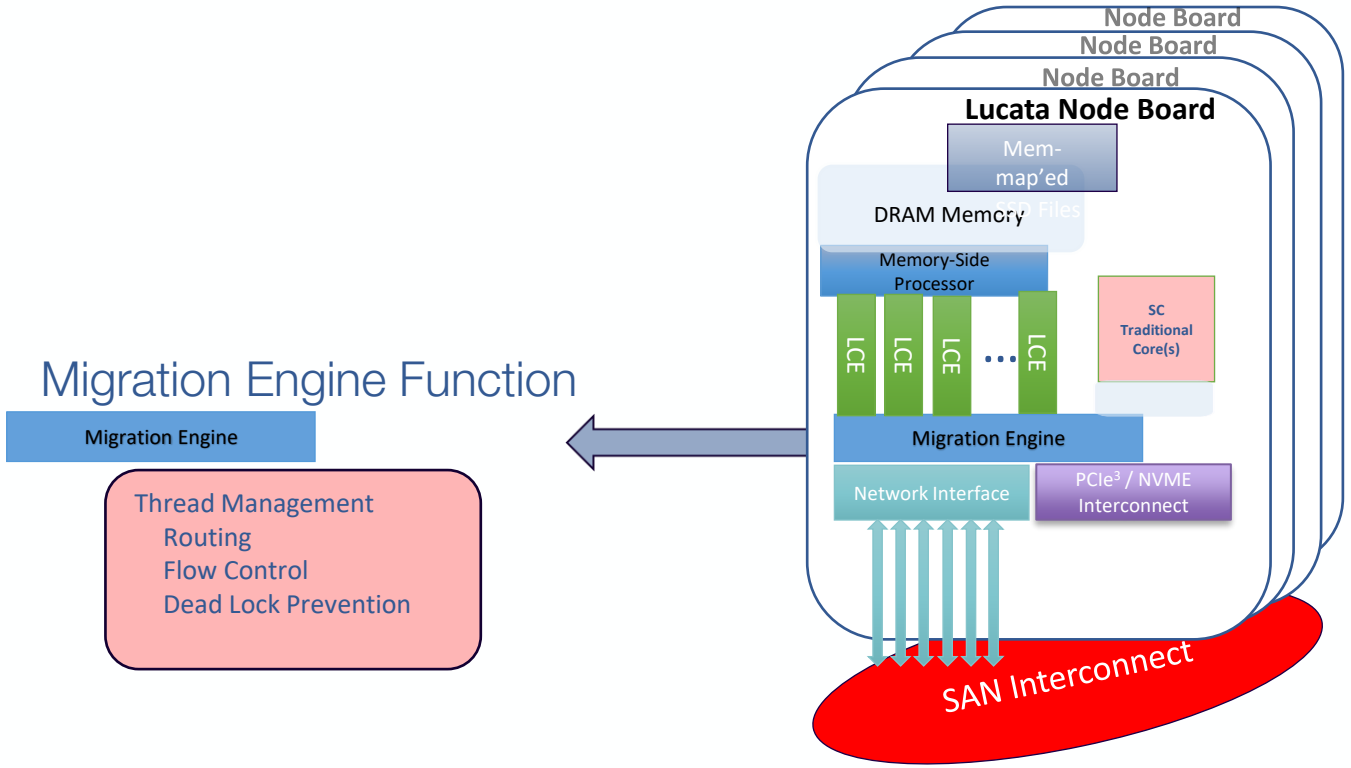
- Node Boards interconnected with dual plane 100Gb/s SAN



Lucata Technology: What migrates instead of data



Lucata Technology: Simultaneously execute 100's of thousands of threads



Lucata Technology: Narrow Channel Memory Access

Memory Side Processor

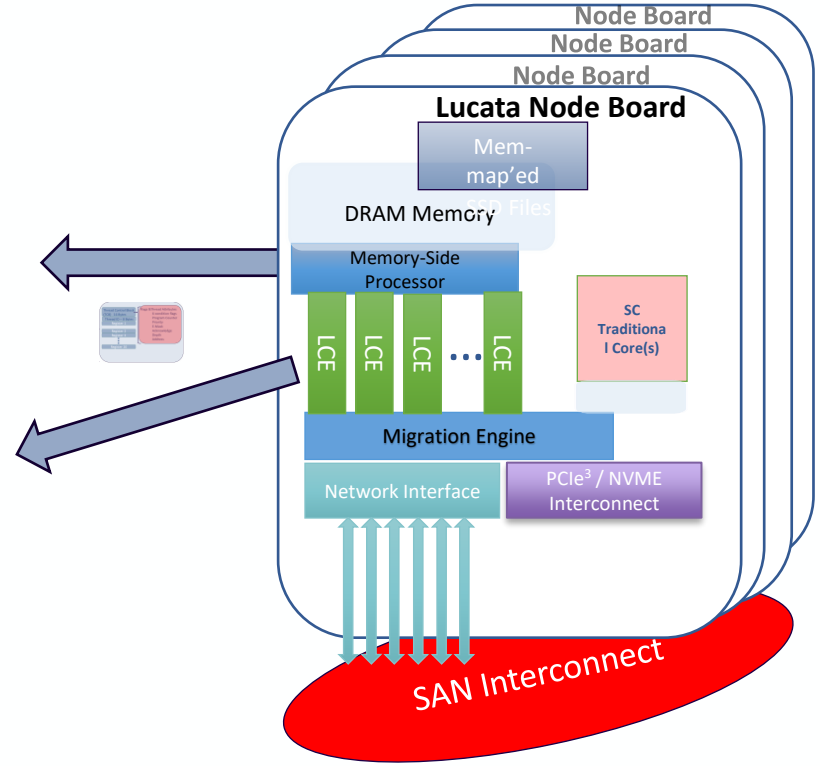
Memory-Side Processor

Memory Transactions
- Order control
- Atomic Operations

Lucata Compute Element

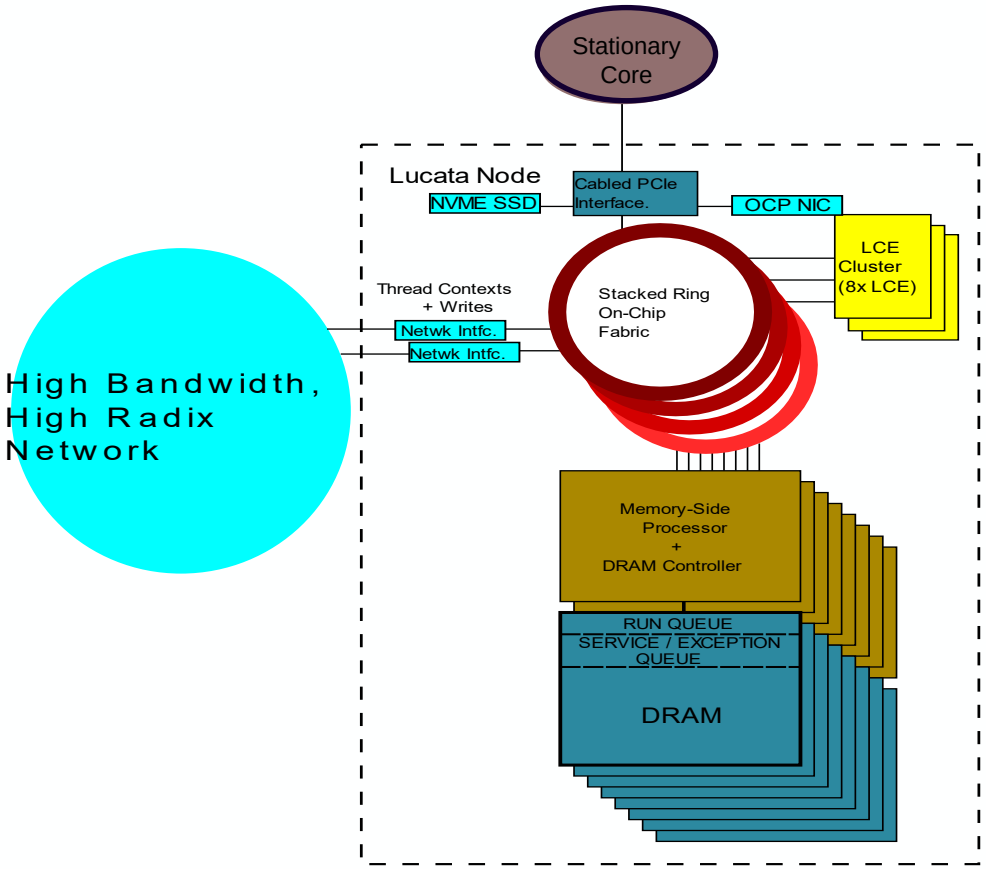
LCE LCE LCE LCE

Local Parallel Compute
- Logical Operations
- Arithmetic Operations
- Massive multithreading hides latency



Lucata Pathfinder System Architecture

- GT hosts a four chassis Pathfinder-S installation
- Uses a PowerPC Stationary Core instead of x86 host for upcoming systems
- 8 Nodes per Chassis
- 8 Chassis per Rack
- RapidIO Network with multi-level switch
  - Contexts for migrating threads
  - Write packets for remote memory operations



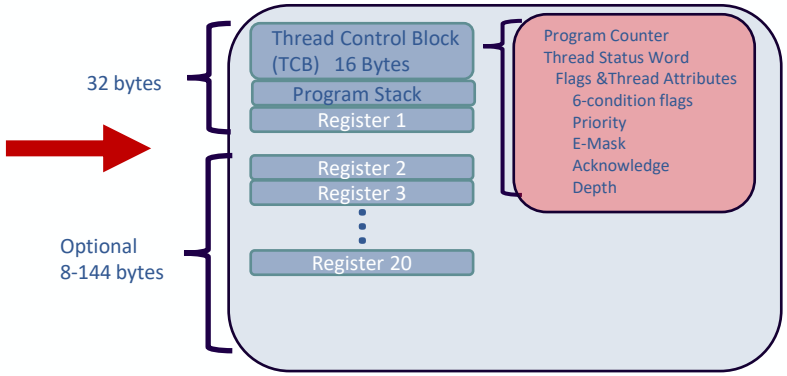
Node Architecture

- 24 Lucata Compute Elements (LCE)
- 4 Memory Side Processors (MSP)
- 64GB DRAM
  - 4 banks of 16GB dual-port DDR4
- Stacked Ring Fabric for on-chip communication
- 6 RapidIO 2.3 4-lane network ports
- Stationary Core (SC)
  - DualCore 64-bit Power E5500
  - 2GB DRAM
  - 1 TB SSD
  - PCIe Gen 3
  - Runs Linux

Gossamer Core Architecture

- Deeply pipelined, multithreaded core
  - Custom, accumulator-based ISA
  - Support for 64 active hardware threads
  - Thread Context
    - Program Counter
    - Registers
    - Thread status words
- Multithreading hides instruction latency, including local memory operations

Thread context contained in Thread State Record (TSR)



Hardware Thread Management

- Thread scheduling in GCs automatically performed by hardware
- SPAWN instruction
  - Creates new thread and places it in Run Queue
- RELEASE instruction
  - Places thread in Service Queue for processing by SC
- Non-local memory reference causes a migration
  - Thread context packaged by hardware and sent over system interconnect to destination node
  - Arriving thread context is placed in Run Queue at destination node

09/2022

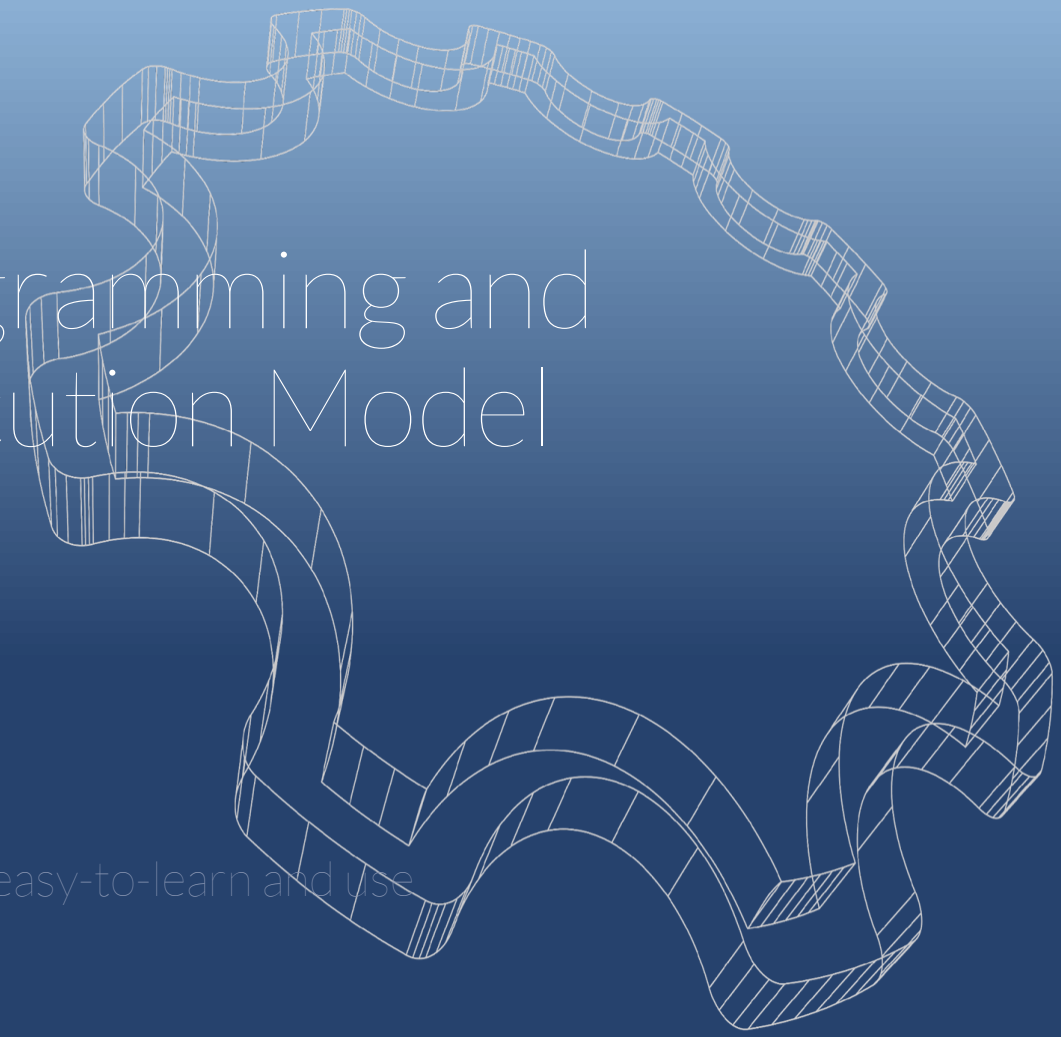
System Level Spawn Control

- Threads do not inherently know how many spawns other threads have executed
- Credit-based hardware/software scheme under development to
  - Limit the total number of threads to only what the system can handle
  - Handle hotspots where large numbers of threads converge on a single nodelet
  - Identify and avoid hotspots when possible

System Software

- LINUX runs on the Stationary Cores (SCs)
- OS launches main() user program on a Gossamer Core (GC)
  - main() then spawns descendants that execute in parallel and migrate throughout system as needed
- Runtime executes primarily on the SCs
  - Handles service requests from threads running on the GCs including: memory allocation, I/O, exception handling, and performance monitoring
- Threads return to main()upon completion, which then returns to the OS

Programming and Execution Model

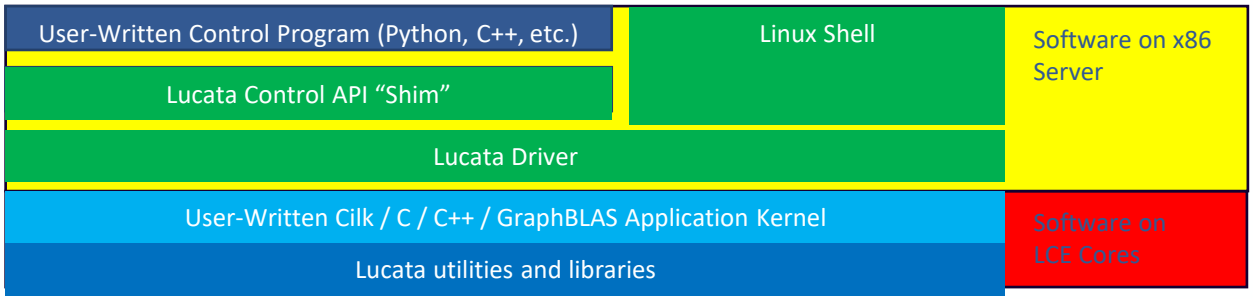


Flexible, easy-to-learn and use

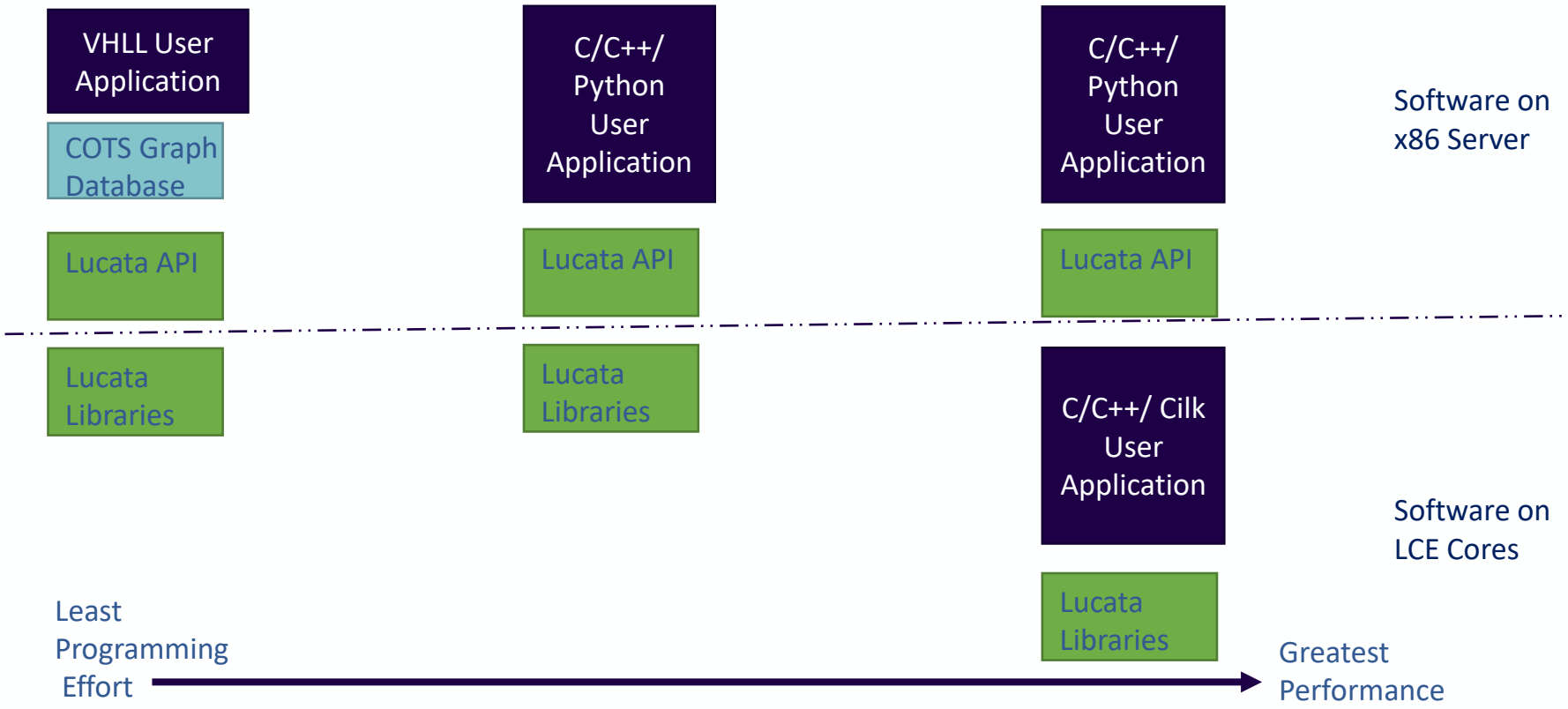
LUCATA

Software Stack

- Native shared-memory programming model for maximum performance and flexibility
  - C/C++ with standard libraries
  - Support for parallelism, concurrency & data distribution
- Higher level software
  - Runs on x86 server, uses Lucata driver to execute on cores
  - Python, C/C++ interfaces



Programming Models



Lucata Programming Environment

- Dynamic parallelism via Cilk / C / C++
- memoryWeb and C/C++ utilities libraries for data distribution
- Intrinsic functions for architecture-specific operations
- Replicated variables to avoid unnecessary migrations
- GraphBLAS, BeeDrill, and LAGraph libraries

Lucata GraphBLAS library

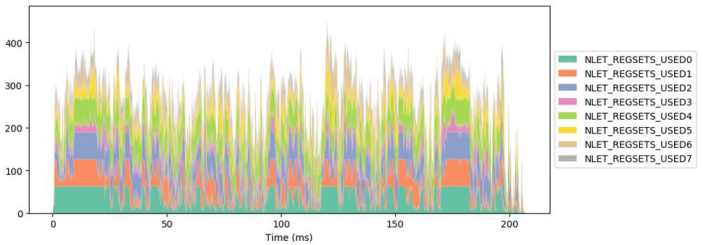
- Implements full GraphBLAS API
- Greatly reduces development time / improves productivity
- Achievable performance ~50% of custom-written graph codes with 10-25% of coding effort
- Open Source; written in OpenCilk

Lucata GraphBLAS library

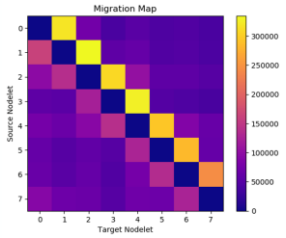
- Implements full GraphBLAS API
- Greatly reduces development time / improves productivity
- Achievable performance ~50% of custom-written graph codes with 10-25% of coding effort
- Open Source; written in OpenCilk

Performance Counters & API

- Hardware Counters to measure numerous performance parameters, all snapshot simultaneously throughout system
  - IPC
  - Memory transactions
  - Network Transactions
  - Stall Cycles
  - Peak active threads



- Simulator and Hardware have identical Counters
- System Calls to start and stop counting
- Profiling and visualization tool



What have we not covered here?

- Low-level compiler and custom code generator details for Lucata Cilk
- Stdlib support, User libraries, profiler and other tool details
- The rest of the tutorial will cover basic programming of the Lucata system and applications