JCAS - IAA Simulation Efforts at Oak Ridge National Laboratory

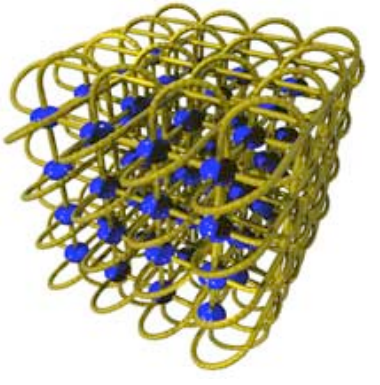
Christian Engelmann, Oak Ridge National Laboratory

Outline

- A look back
  - JCAS motivation in 2002
  - Cellular algorithms theory
  - Accomplishments and limitations
- IAA simulation efforts at ORNL
  - Motivation and goals
  - Technical approach
  - Closely related work (Pose/Charm++ by S. Kale et al.)
  - External contribution (μπ/μsik by KP@ORNL)
  - Future work
  - Discussion points for breakout groups

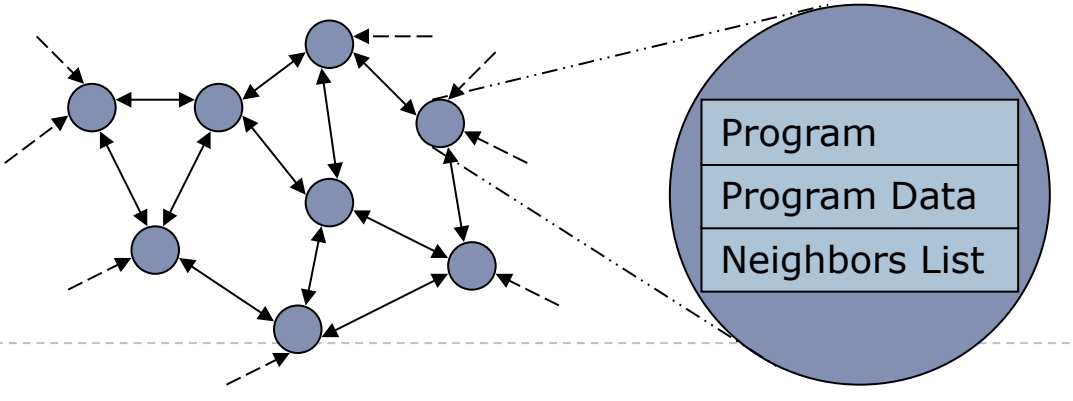
A Look Back

- JCAS was developed in 2002-2004
- It was motivated by the IBM Blue Gene/L effort
  - The already expected increase in scale to +100,000 cores
  - The anticipated growth in scale over time to +1,000,000
- How to scale algorithms efficiently *(Amdahl's law)*?
- How to deal with fault tolerance at extreme scale?
- The ORNL/IBM CRADA in cellular algorithms research was formed to explore and demonstrate:
  - Naturally fault-tolerant algorithms
  - Scale invariant algorithms



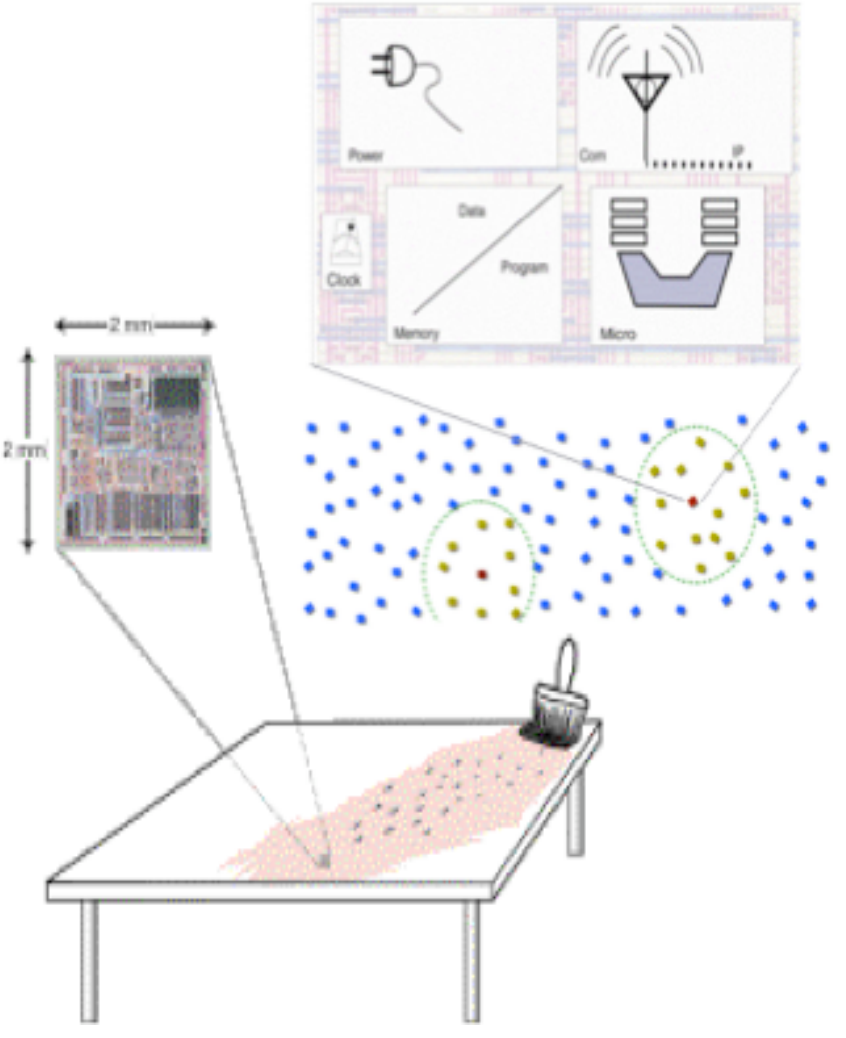
Cellular Algorithms Theory

- Processes have only limited knowledge mostly about other processes in their neighborhood
- Application is composed of local algorithms
- Less inter-process dependencies, e.g, not everyone needs to know when a process dies
- Peer-to-peer communication with overlapping neighborhoods promotes scalability
- MIT Media Lab. Research: Paintable Computing.



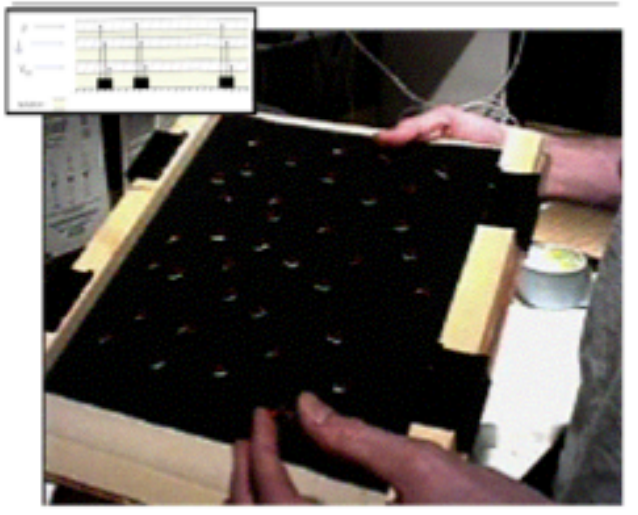
MIT Research: Paintable Computing

- In the future, embedded computers with a radio device will get as small as a paint pigment
- Supercomputers can be easily assembled by just painting a wall of embedded computers
- Applications are driven by cellular algorithms



MIT Research: Pushpin Computing

- 100 embedded nodes
- 1.25m x 1.25m pushpin board provides power
- Initial applications:
  - Distributed audio stream storage
  - Fault-tolerant holistic data (image) storage
- Ongoing research:
  - Sensor networks

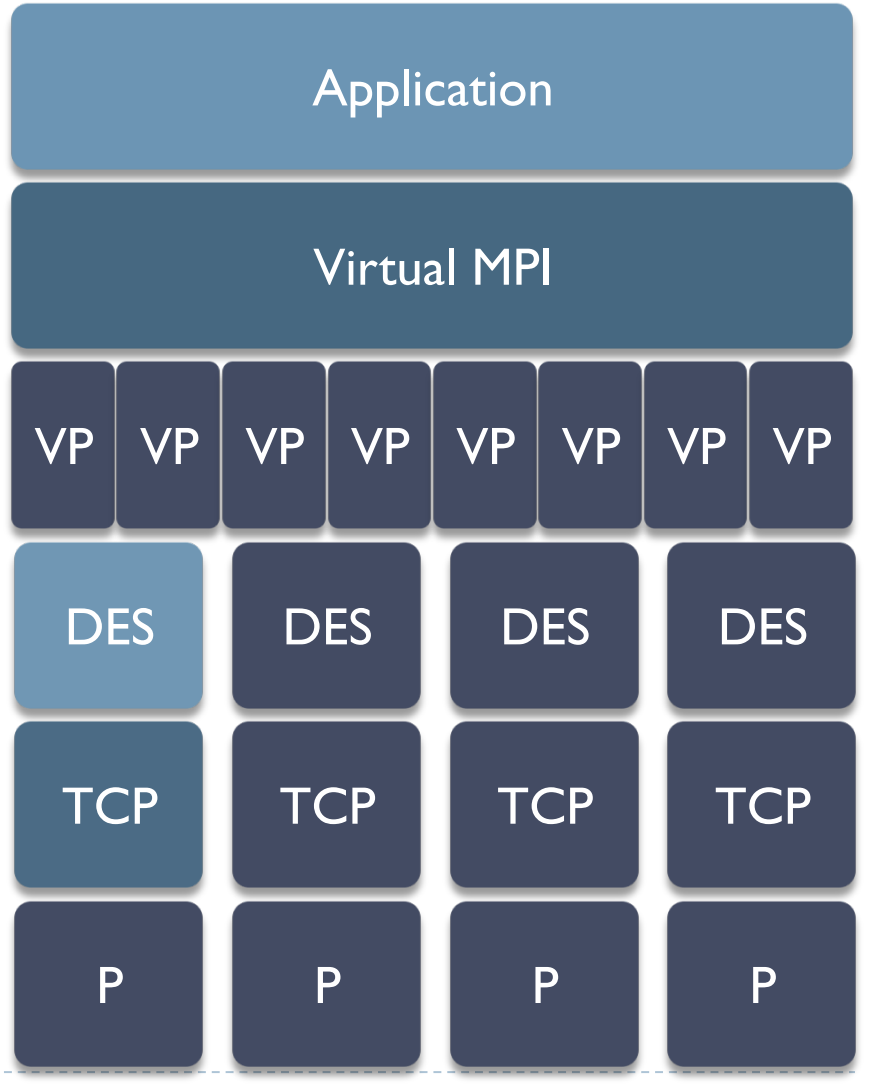


Java Cellular Architecture Simulator (JCAS)

- Developed at ORNL in Java
- Native C and Fortran application support using JNI
- Runs as standalone or distributed application
- Lightweight framework simulates up to 1,000,000 lightweight virtual processes on 9 real processors
- Standard and experimental network interconnects:
  - Multi-dimensional mesh/torus
  - Nearest/Random neighbors
- Message driven simulation without notion of time
  - Not in real-time, no time-accurate discrete event simulation
- Primitive fault-tolerant MPI support
  - No collectives, no MPI 2

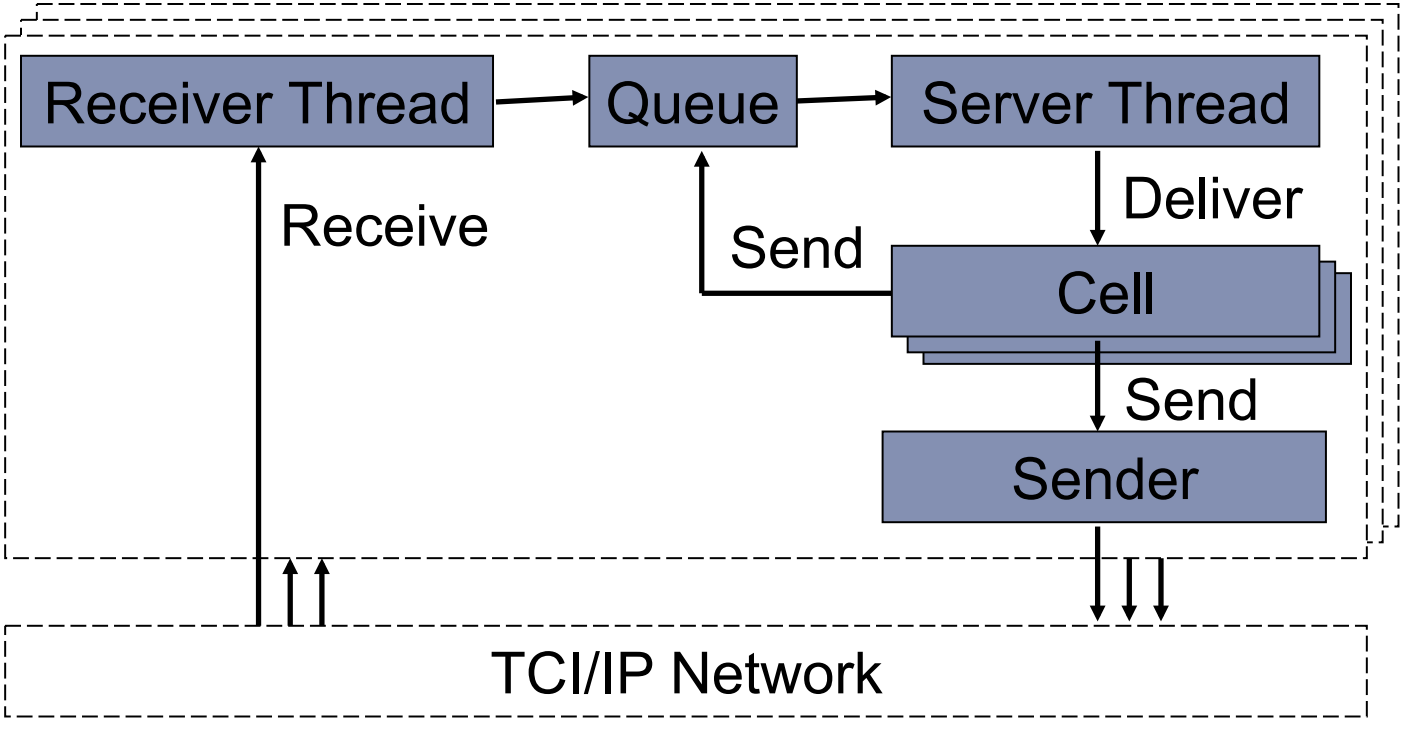
Technical Approach

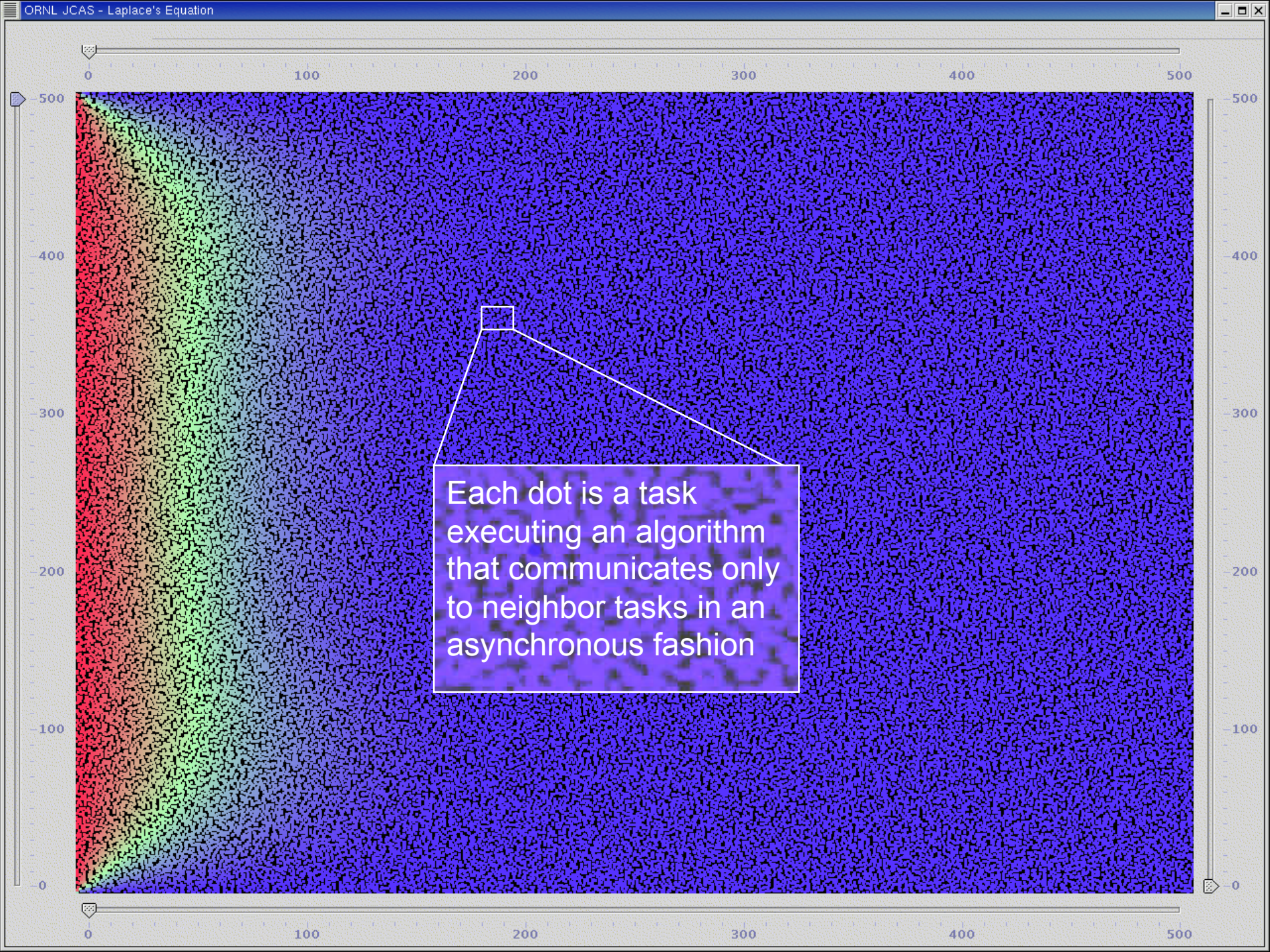
- Distributed set of discrete event simulators with node-local message queues
- Simulation of virtual MPI processes for parallel app.
- Virtual processes run on real hardware with virtual MPI
- No virtual process time
- Fault injection capability
- Interactive graphical user interface as front-end
- TCP/IP servers as back-ends

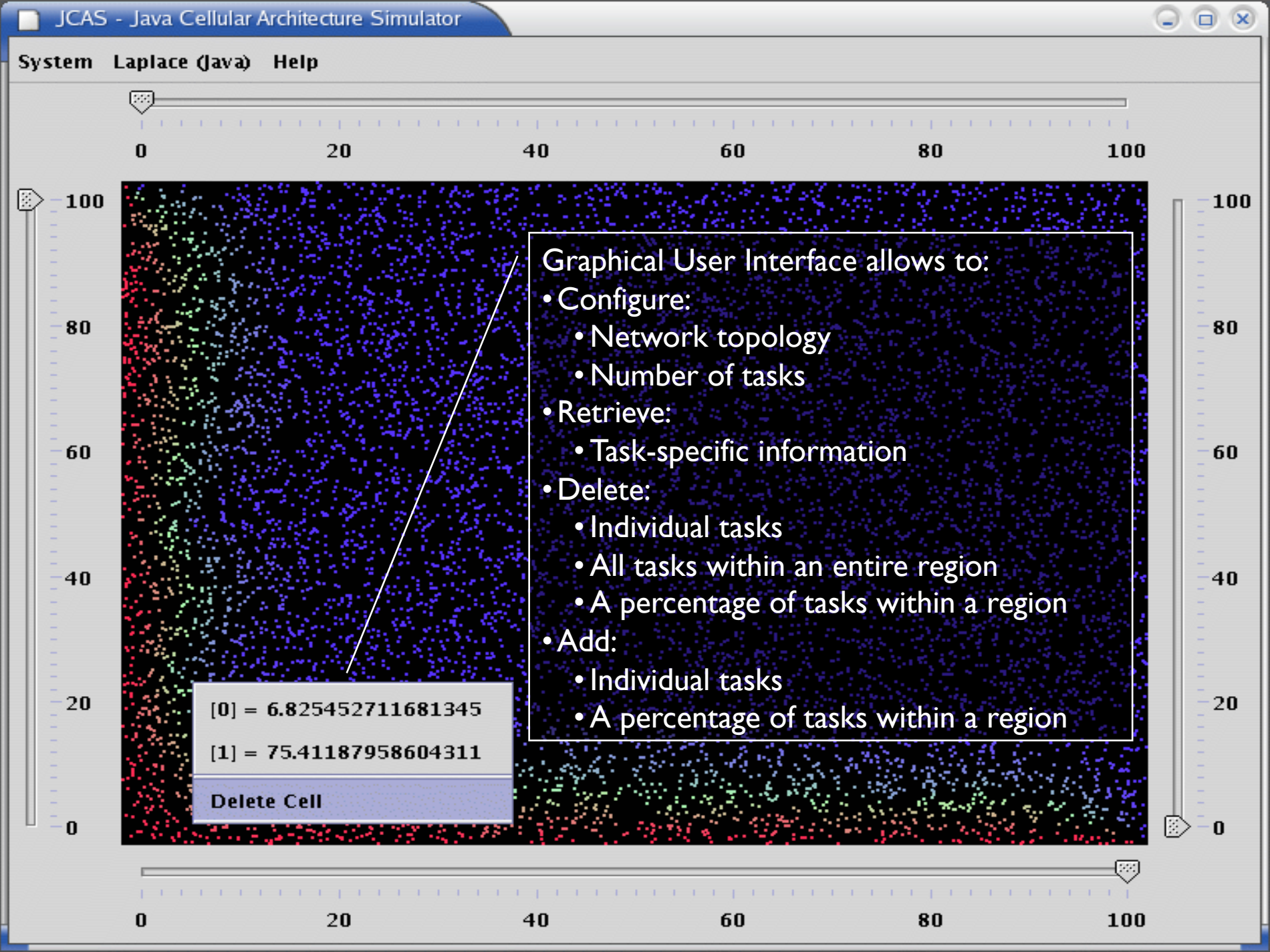


Implementation

- Every cell has its own code, memory and neighbors list
- Server hosts cells and initiates the context switch
- Cells communicate asynchronously using messages




ORNL JCAS - Laplace's Equation
0
100
200
300
400
500
Each dot is a task executing an algorithm that communicates only to neighbor tasks in an asynchronous fashion


JCAS - Java Cellular Architecture Simulator
System
Laplace (Java)
Help
0
20
40
60
80
100
[0] = 6.825452711681345
[1] = 75.41187958604311
Delete Cell
Graphical User Interface allows to:
• Configure:
• Network topology
• Number of tasks
• Retrieve:
• Task-specific information
• Delete:
• Individual tasks
• All tasks within an entire region
• A percentage of tasks within a region
• Add:
• Individual tasks
• A percentage of tasks within a region

Targeted Applications/Algorithms

- Local information exchange algorithms:
  - Mesh-free chaotic relaxation (Laplace/Poisson)
  - Finite difference/element methods
  - Dynamic adaptive refinement at runtime
  - Asynchronous multi-grid methods
  - Peer-to-peer diskless checkpointing
- Global information exchange algorithms:
  - Global peer-to-peer broadcasts of values
  - Global maximum/optimum search
- Applications:
  - Locally self-consistent multiple scattering (LSMS) method
  - Molecular dynamics simulation for computational biology

Limitations

- Simulator scalability
  - Simulation capability is limited to simple algorithms with ~1,000,000 virtual processors on ~10 real processors
  - Larger-scale simulations and/or running more complex codes requires to scale the simulator to 100-1,000 real processors
- MPI virtualization
  - Only a very basic (but fault tolerant) MPI layer is provided to parallel applications running on the simulator
- Virtual time
  - There is no simulation of virtual system time, resource accounting and network interconnect timing (latency)

Other Related Research Efforts

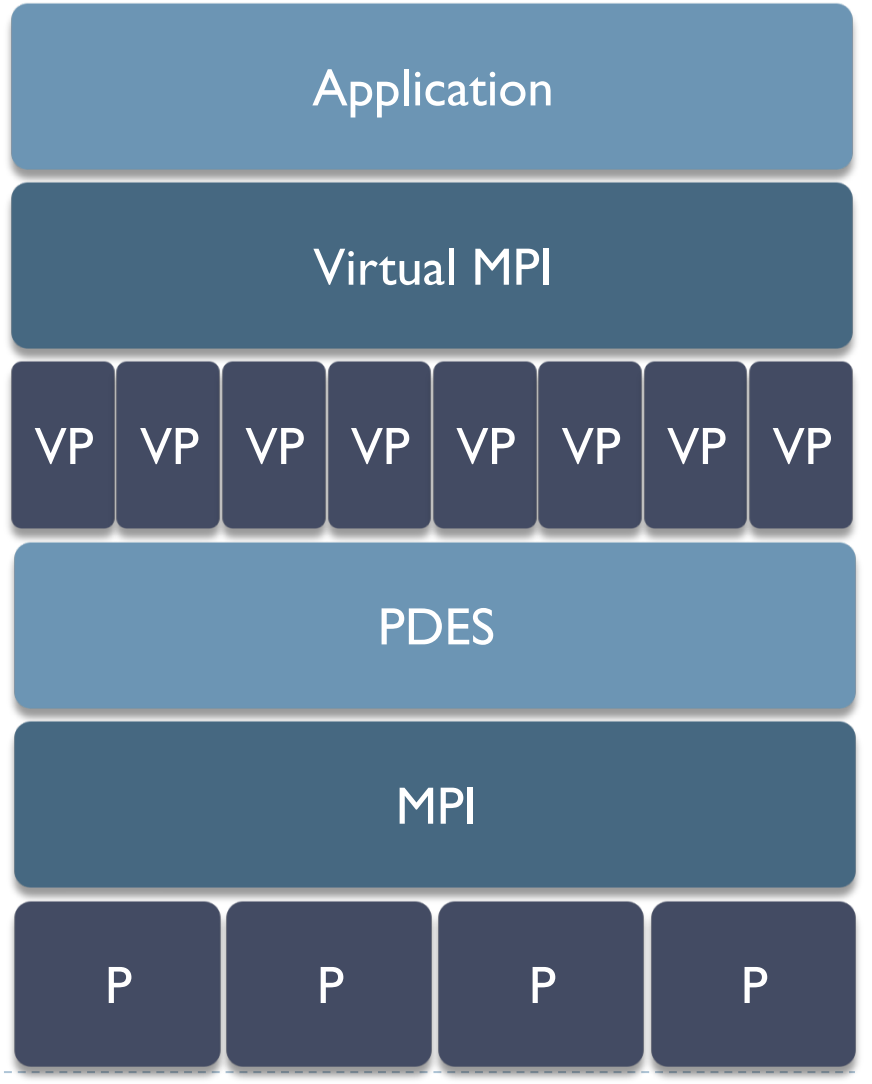
- IBM Research:
  - IBM Power PC processor emulation as a Linux process
  - Slow and resource hungry (full OS per emulated node)
- Caltech:
  - MPI trace file analysis for performance prediction
- UIUC (BigSim):
  - Message driven simulation of low level machine API running the Charm++ programming model
  - Adaptive MPI running on top of Charm++
  - Post-mortem mode for performance prediction
  - Scalability and general performance issues
  - Fixed to Blue Gene/L architecture

IAA Simulation Efforts at ORNL

- Investigate scalability, performance and fault tolerance of algorithms at extreme scale through simulation
- Extending the JCAS simulation capabilities
  - Simulating more processes (~10,000,000)
  - Running more complex and resource-hungry algorithms
  - Support for unmodified MPI applications
- Evaluation of algorithms at extreme scale
  - Notion of global virtual time and virtual process clocks
  - Accounting for resource usage, such as processor and network
  - Gathering of scalability, performance & fault tolerance metrics
  - Parameter studies at scale

Technical Approach

- Parallel discrete event simulation (PDES) atop MPI
- Simulation of virtual MPI processes for parallel app.
- Virtual processes run on real hardware with virtual MPI
- Consistent virtual process clock from PDES
- Virtual process clock can be scaled by PDES via model
- Virtual interconnect latency is set by PDES via model



Needed JCAS Modifications*

1. Port JCAS to C/C++ to improve scalability/performance
2. Replace TCP/IP with (native) MPI communication
3. Replace Distributed set of DESs with PDES
    1. Conservative, optimistic and time-warp synchronization
4. Extend virtual MPI capabilities
    1. Asynchronous, collectives, process control (spawn), …
5. Extend fault injection and notification mechanisms
    1. Injection based on failure distributions and application state
6. Add simulated machine model (for network)
7. Gather scalability, performance & fault tolerance metrics

* easy (days/weeks), difficult (weeks), challenge (months)

Leveraging Existing Work

- Drastic changes in JCAS are required
- Before we start, let's try not to reinvent the wheel
- Some research has already been done in this area
- Existing PDES cores, e.g.:
  - Pose by Sanjay Kale et al. (UIUC)
  - μsik by Kalyan Perumalla (ORNL)
- Existing MPI virtualization layers, e.g.:
  - Adaptive MPI by Sanjay Kale et al. (UIUC)
  - μπ by Kalyan Perumalla (ORNL)
- Our recent work focused on identifying existing solutions for integration & enhancement to replace JCAS

Pose Simulator and Adaptive MPI

- Pose was developed by Terry Wilmarth (UIUC) for the BigSim effort lead by Sanjay Kale (UIUC)
  - PDES engine with conservative and optimistic synchroniza-ion support (global virtual time)
  - Runs atop Charm++ on many systems, e.g., BG/P, Cray XT
- Possible future work within ORNL's IAA simulation effort
  - Run MPI virtualization layer, e.g., Adaptive MPI, atop Prose
  - Adaptive MPI runs atop Charm++ and has already full MPI support and load balancing
  - Pose and AMPI would need to overcome scaling challenges:
    - A more lean implementation
    - Time-warp synchronization

μπ (MUPI) Simulator Prototype

- Developed by Kalyan Perumalla (ORNL) outside of IAA
- μπ (micro parallel performance investigator)
  - PDES for MPI applications (MPI virtualization)
  - Support for basic MPI communication primitives
  - MPI application is executed on the real hardware
  - Execution and communication timing can be adjusted according to simulated machine description
- Based on μsik (micro simulator kernel)
  - Scalable PDES engine with conservative, optimistic and time-warp synchronization support (global virtual time)
  - TCP- or MPI-connected simulation kernels
  - Support for many systems, e.g., BG/L and Cray XT

How to Run an MPI Application on μπ*

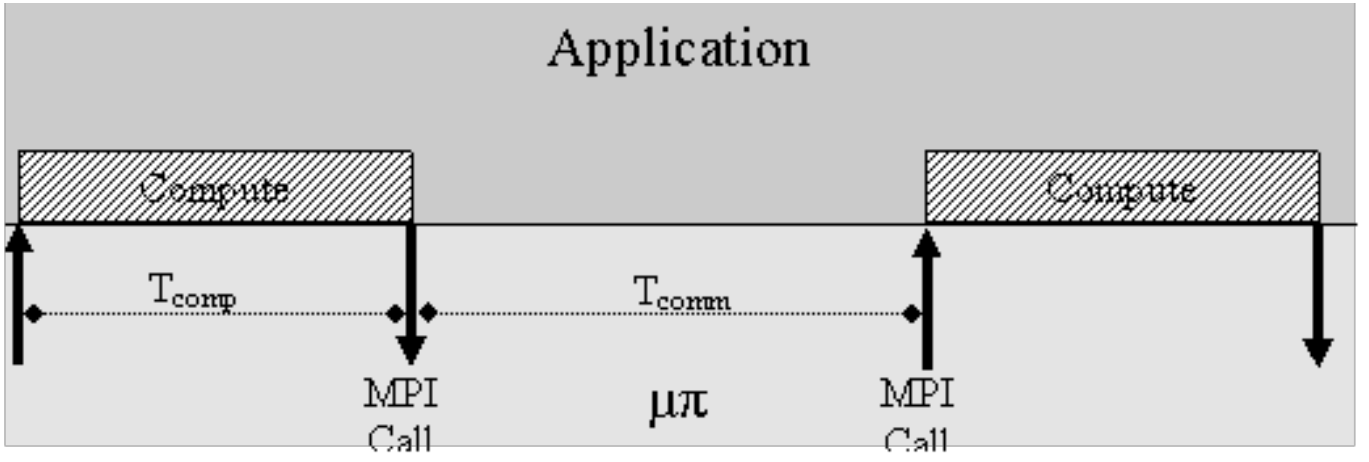
- Change MPI include and recompile
  - #include <mpi.h> to #include <mupi.h>
- Add linker flag and relink
  - -lmupi
- Execte MPI application (now $\mu\pi$ simulation)
  - mpirun –np 4 myprog -np 32

    runs myprog on 32 virtual cores,
    simulated by $\mu\pi$ on 4 real cores

* From μπ documentation

Current μπ Capabilities*

- Support for FORTRAN and C applications
- FORTRAN applications can be object-only or source-code
- C applications need source-code for all object code that calls MPI



* From μπ documentation

Current μπ Capabilities* (continued)

- Compiles and runs on several platforms
  - Desktops
  - Clusters
  - Supercomputers
- Tested on
  - Linux
  - Mac OS X
  - Windows (Native, as well as Cygwin)
  - Cray XT4/XT5, Blue Gene

* From μπ documentation

Current μπ Status and Future Work

- μπ is a first, very early prototype
  - Basic MPI communication support only
  - No simulated machine model yet
  - However, a big step in the right direction
- Possible future work within IAA
  - Extend MPI communication support in μπ (collectives)
  - Add fault injection and notification mechanisms
  - Add simulated machine model (for network)
- Possible future integration with Sandia effort
  - Execute models in μπ instead of MPI applications
  - Input for models come from cycle-accurate simulations

Summary and Future Work

- JCAS can simulate up to 1,000,000 virtual processors on 10 real processors, but is limited in scale and usability
- ORNL's IAA simulation efforts targets a new type of simulator based on a scalable PDES engine that can
  - Simulate more processes (~10,000,000)
  - Run more complex and resource-hungry algorithms
  - Support unmodified MPI applications
  - Keep track of global virtual time and virtual process time
  - Accounts for resource usage, such as processor and network
  - Gathers of scalability, performance & fault tolerance metrics
  - Perform parameter studies at scale
- Future work focuses on improving μπ to meet IAA goals

Discussion Points for Breakout Groups

- HPC simulation is a big area with various goals and approaches
  - Architectural properties (processor, memory, network) and application properties (scaling and fault tolerance)
  - Time slice and discrete event simulation
  - Machine cycle and programming model granularity
  - Single-processor, multi-processor and extreme-scale simulation
- How can these efforts interface with each other to
  - Avoid reinventing the wheel all over again (reuse of code !!!)
  - Reuse (benefit from) each other's results, e.g., simulators feed their output into each other (small-to-large scale & back, free & commercial)
- What are the true challenges for simulation efforts in the HPC community (apart from funding)?
  - Scalable simulation cores, standard models and interfaces, …?

Questions?

IAA Simulation Efforts at Oak Ridge National Laboratory

Christian Engelmann, Oak Ridge National Laboratory