

# *Scalable Interaction with Parallel Applications*


Filippo Gioachin  
Chee Wai Lee  
Laxmikant V. Kalé

Department of Computer Science  
University of Illinois at Urbana-Champaign

# Outline

- Overview
  - Charm++ RTS
  - Converse Client Server (CCS)
- Case Studies
  - CharmDebug (parallel debugger)
  - Projections (performance analysis tool)
  - Salsa (particle analysis tool)
- Conclusions

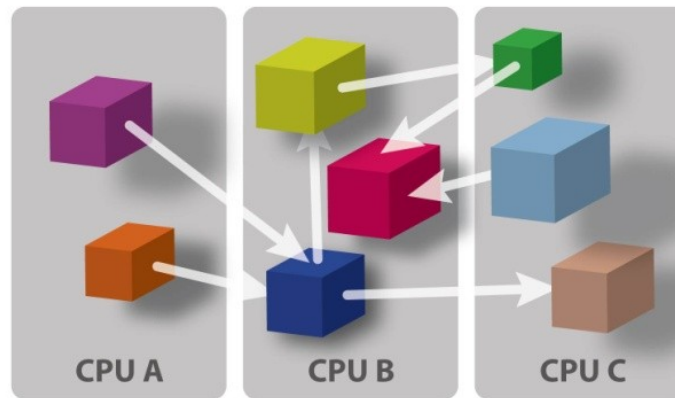
# Overview

- Need for real-time communication with parallel applications
  - Steering computation
  - Visualizing/Analyzing data
  - Debugging problems
- Long running applications
  - Time consuming to recompile the code (if at all available)
  - Need to wait for application to re-execute
- Communication requirements:
  - **Fast** (low user waiting time)  **Scalable**
  - **Uniform method of connection**

# Charm++ Overview

- Middleware written in C++
  - Message passing paradigm (asynchronous comm.)
- User decomposes work among objects (*chares*)
  - The objects can be virtual MPI processors
- System maps chares to processors
  - automatic load balancing
  - communication optimizations

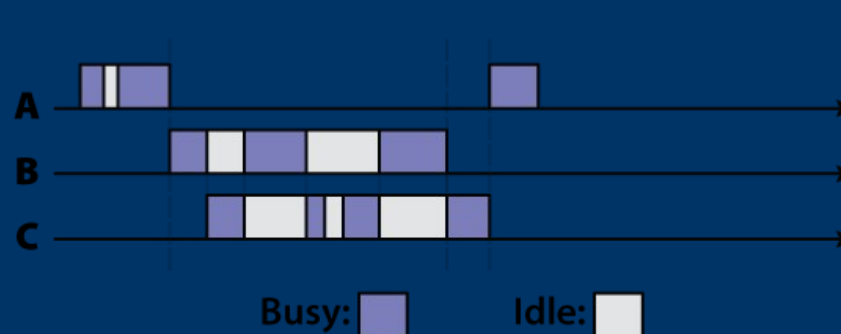
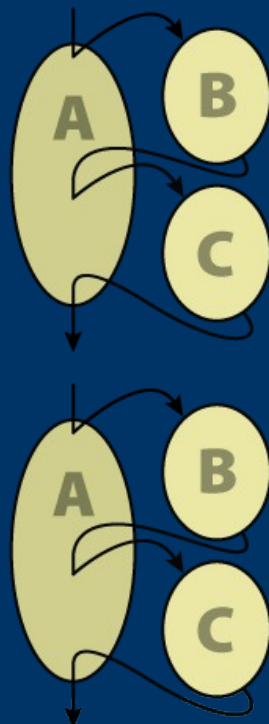
User view



System view

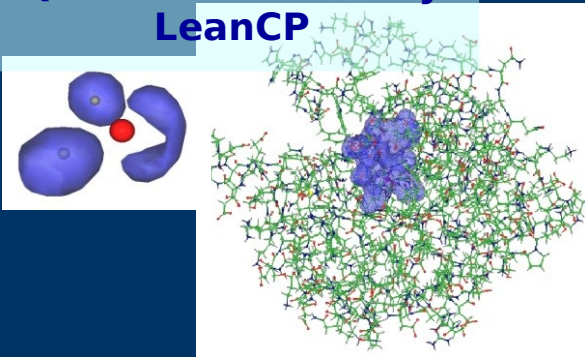
# Adaptive overlap and modules

- Allow easy integration of different modules
- Automatic overlap of communication and computation

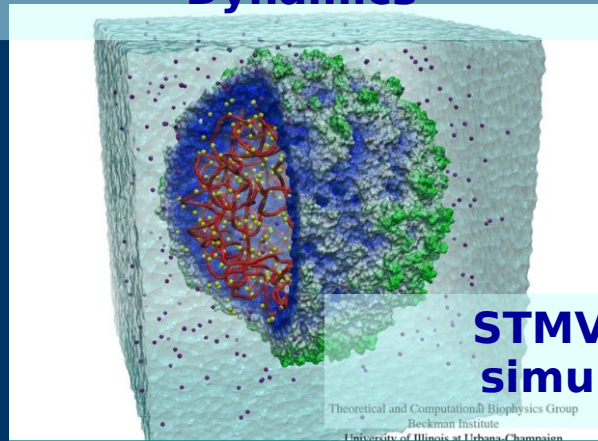


# Develop Abstractions in Context of Full-Scale Applications

Quantum Chemistry  
LeanCP

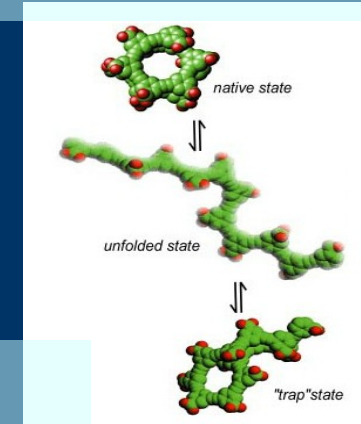


NAMD: Molecular Dynamics

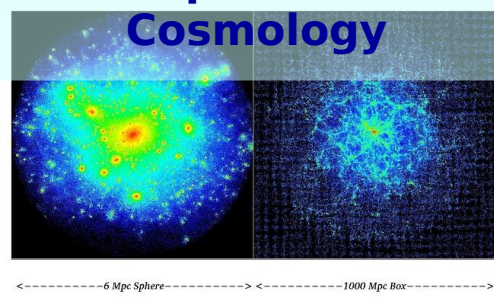


STMV virus simulation

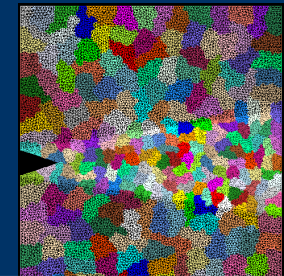
Protein Folding



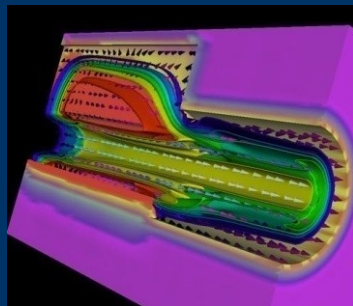
Computational Cosmology



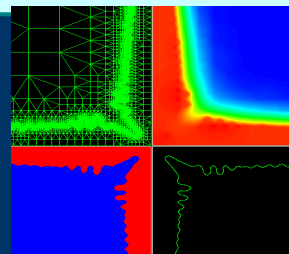
Parallel Objects,  
Adaptive Runtime System  
Libraries and Tools



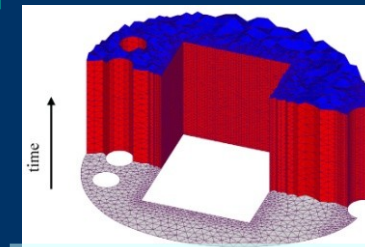
Crack Propagation



Rocket Simulation

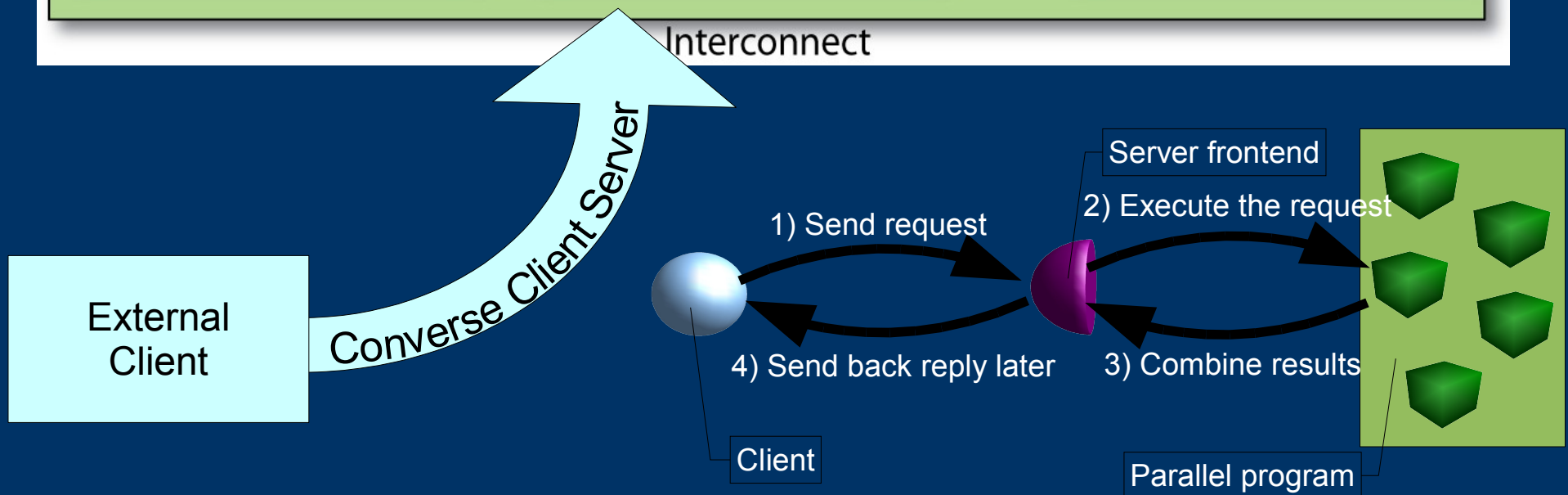
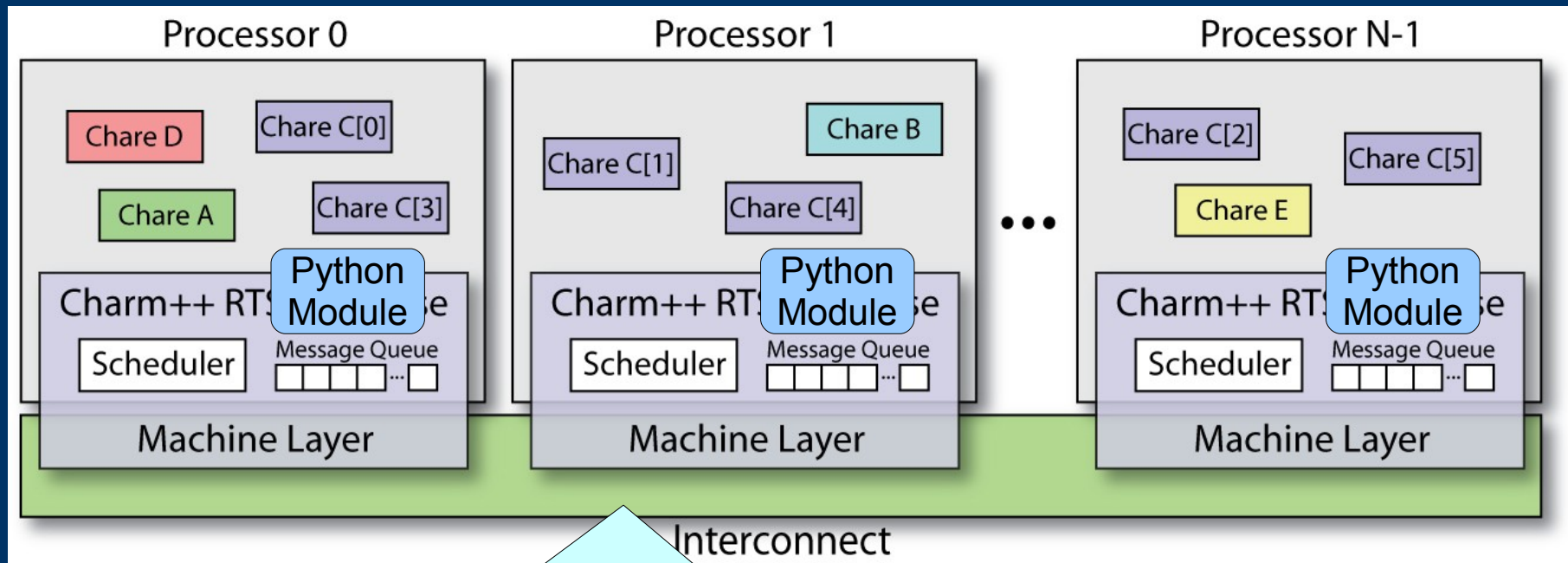


Dendritic Growth



Space-time meshes

# Charm++ RTS



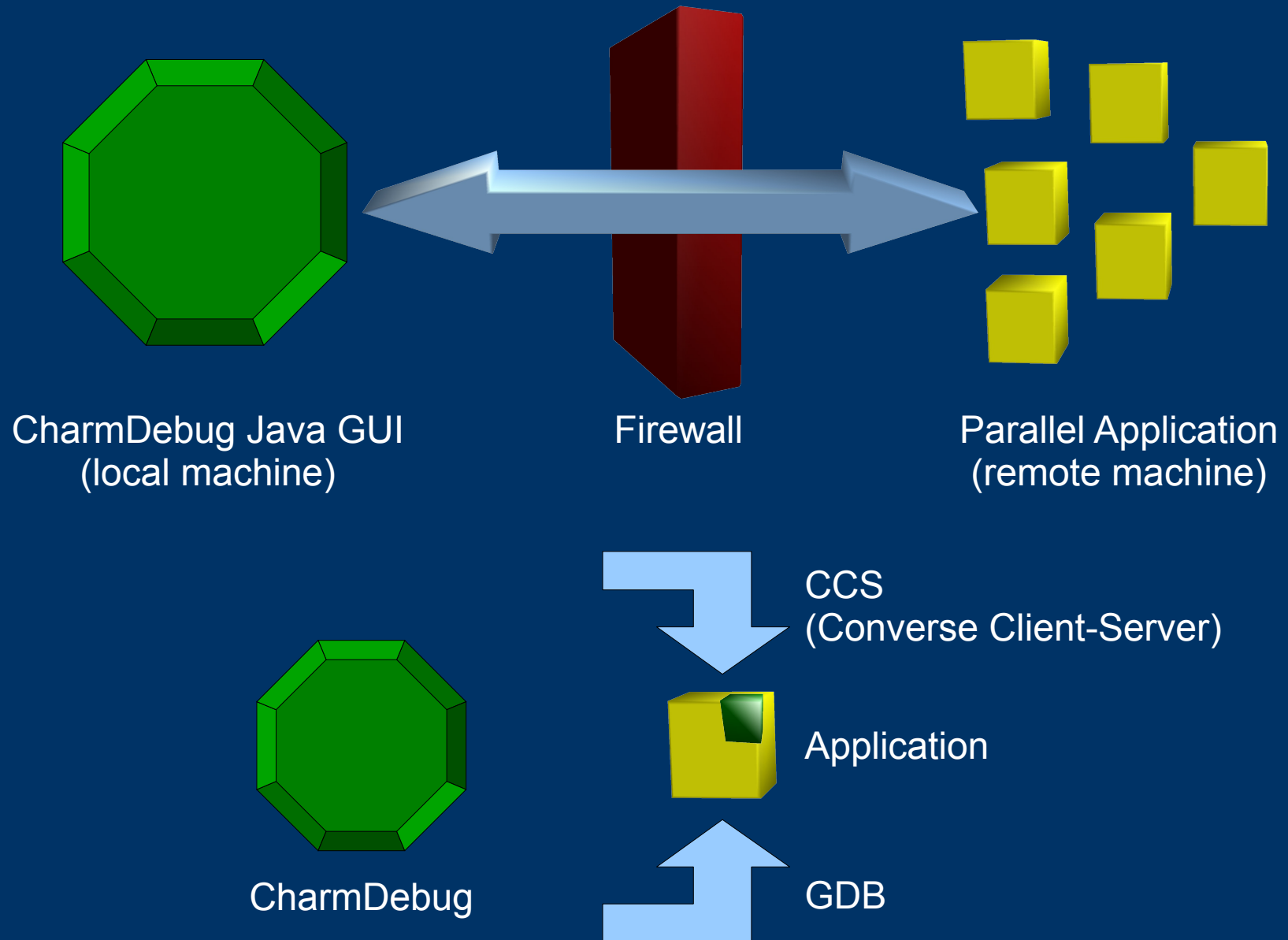
# *Case study: Parallel Debugging*



# Large Scale Debugging: Motivations

- Bugs on sequential programs
  - Buffer overflow, memory leaks, pointers, ...
  - More than 50% programming time spent debugging
  - GDB and others
- Bugs on parallel programs
  - Race conditions, non-determinism, ...
  - Much harder to find
    - Effects not only happen later in time, but also on different processors
  - Bugs may appear only on thousands of processors
    - Network latencies delaying messages
    - Data decomposition algorithm
  - TotalView, Alinea DDT

# CharmDebug Overview



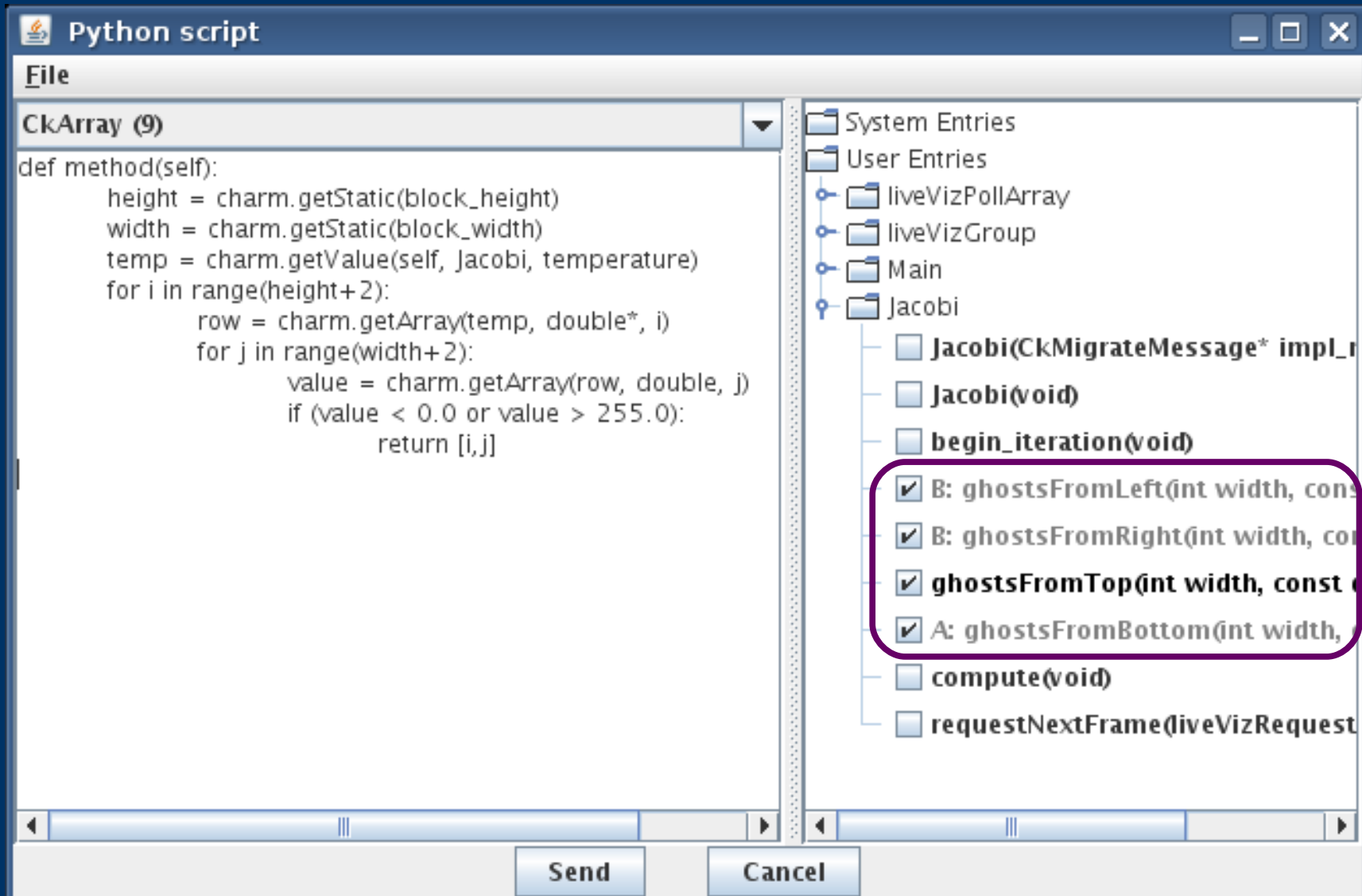
# Main Program View

The screenshot shows the Charm Parallel Debugger interface. On the left, the 'Set Break Points' tree is annotated with a red bracket and the text 'entry methods'. The tree includes folders for 'System Entries', 'User Entries', 'Main', 'Hello', 'HelloGroup', 'HelloNode', 'HelloChare', and 'SecondArray'. Under 'Hello', methods like 'Hello(CkMigrate)', 'Hello(void)', and 'SayHi(int hiNo)' (checked) are listed. The 'Control Buttons' section contains 'Start', 'Step', 'Continue', 'Freeze', 'Quit', and 'Start GDB'. The 'Program Output' window is annotated with 'output' and shows a list of messages such as 'Hello 1 created', 'Hello 7 created', etc. To its right, the 'Pes' window is annotated with 'processor subsets' and shows 'all' and 'even' processors. Below these is the 'View Entities on PE' section with a 'Messages in Queue' dropdown set to '0'. The bottom section is split into 'Entities' (annotated with 'messages queued') and 'Details' (annotated with 'message details'). The 'Entities' list shows 'Hello::SayHi(int hiNo)' selected. The 'Details' pane shows: 'Sender processor: 0', 'Destination: Hello::SayHi(int hiNo) (type 16)', 'Size: 16', and 'User data: data= {hiNo=27}'. The status bar at the bottom indicates 'Frozen processor 0'.

# CharmDebug at scale

- Current parallel debuggers don't work
  - Direct connection to every processor
  - STAT (MRNet) not a full debugger
- Kraken: Cray XT4 at NICS
  - Parallel operation collecting total allocated memory
  - Time at the client 16~20 ms
  - Up to 4K processors
  - Other tests to come
- Attaching to the running program took also very little (few seconds)

# CharmDebug: Introspection



Memory Processor 0

Action Info

Number of lines: 40  
Horizontal pixels: 1400  
Line size: 16  
Bytes per pixel: 233

Update

Information

\*\*\* LEAKING \*\*\*

Memory type: message

Slot at position 0x1007db8 of size 912 bytes. Belonging to chare 0. Backtrace:

```

function CmiAlloc (0x4efc1c) at ??:0
function CkAllocMsg (0x4951da) at ??:0
function CMessage_Ghost::alloc(int, unsigned long, int*, int) (0x45d262) at jacobi2d.def.h:250
function CMessage_Ghost::operator new(unsigned long, int) (0x45d2ea) at jacobi2d.def.h:237
function Jacobi::begin_iteration() (0x46006a) at jacobi2d.C:202
function CkIndex_Jacobi::_call_begin_iteration_void(void*, Jacobi*) (0x45d30e) at jacobi2d.def.h:443
function CkDeliverMessageReadonly (0x4904a2) at ??:0
function CkLocRec_local::invokeEntry(CkMigratable*, void*, int, bool) (0x4a9413) at ??:0
function CkArrayBroadcaster::deliver(CkArrayMessage*, ArrayElement*) (0x4adac7) at ??:0
function CkArray::rcvBroadcast(CkMessage*) (0x4b0c96) at ??:0
function CkDeliverMessageFree (0x48e181) at ??:0
function _processHandler(void*, CkCoreState*) (0x493c46) at ??:0
function CmiHandleMessage (0x4f0e3c) at ??:0

```

**Severe leak:  
ghost layer messages  
leaked every iteration**

# *Case study: Performance Analysis*

# Online, Interactive Access to Parallel Performance Data: Motivations

- Observation of time-varying performance of long-running applications through streaming
  - Re-use of local performance data buffers
- Interactive manipulation of performance data when parameters are difficult to define a priori
  - Perform data-volume reduction before application shutdown
    - k-clustering parameters (like number of seeds to use)
    - Write only one processor per cluster



# *Projections: Online Streaming of Performance Data*

- Parallel Application records performance data on local processor buffers
- Performance data is periodically processed and collected to a root processor
- Charm++ runtime adaptively co-schedules the data collection's computation and messages with the host parallel application's
- Performance data buffers can now be re-used
- Remote tool collects data through CCS

# Impact of Online Performance Data Streaming

## Simple Charm++ Parallel Application

(Iterations of Work + Barriers)

# Cores	Exec Time in seconds (no Data Collection and Streaming)	Exec Time in seconds (with Data Collection and Streaming*)
4095	21.44s	21.46s
8191	37.84s	37.71s

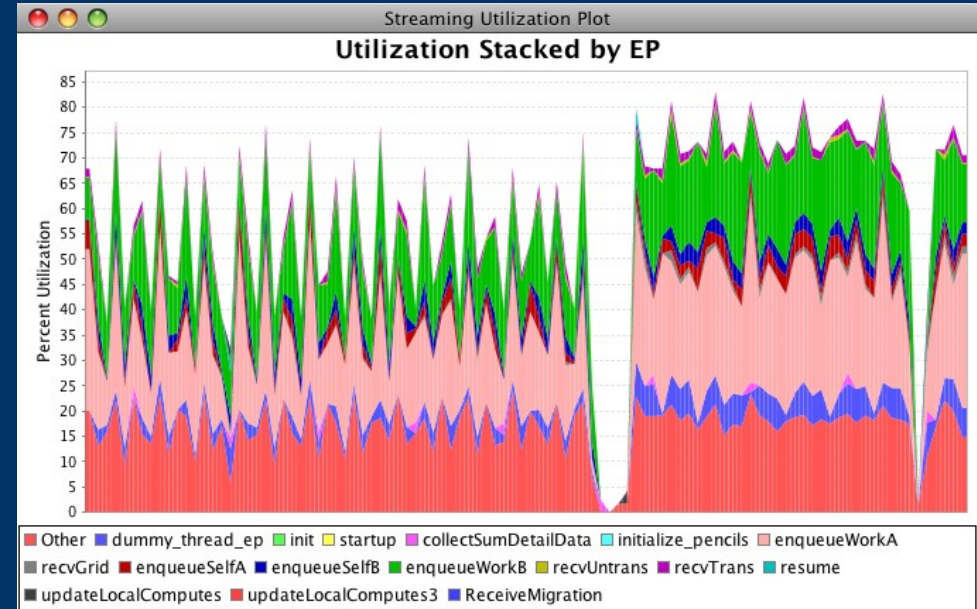
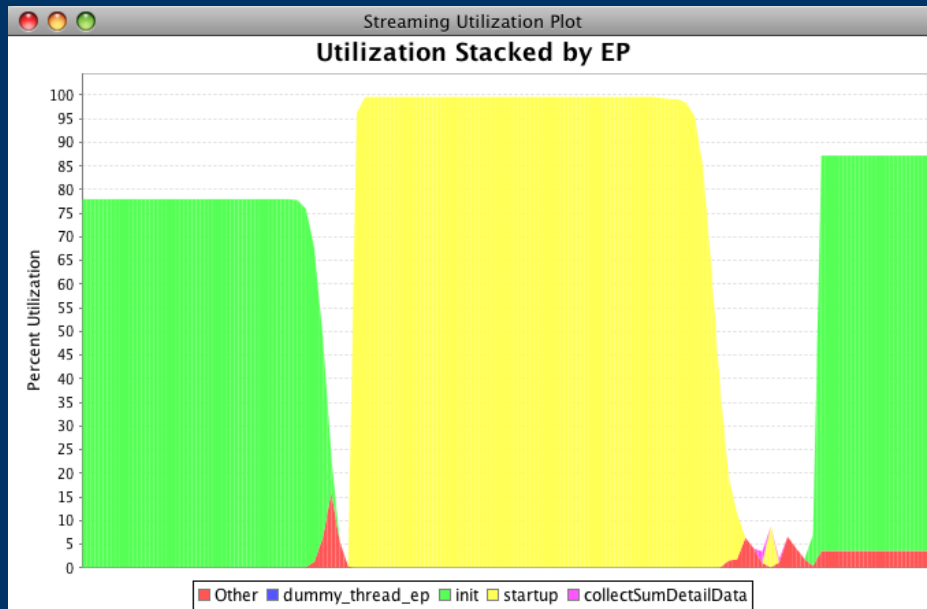
\* Global Reduction of 8 kilobyte messages from each processor every second.

## NAMD 1-million atom simulation (STMV)

# Cores	512	1024	2048	4096	8192
Overhead (%) no Data Collection and Streaming to visualization client.	0.69%	0.55%	-3.44%	1.56%	1.29%
Overhead (%) with Data Collection and Streaming@	0.30%	0.43%	-3.94%	3.47%	6.63%

@ Global Reductions per second of between 3.5 to 11 kilobyte messages from each processor. The visualization client receives 12 kilobytes/second.

# Online Visualization of Streamed Performance Data



- Pictures show 10-second snapshots of live NAMD detailed performance profiles from start-up (left) to the first major load-balancing phase (right) on 1024 Cray XT5 processors
- Ssh tunnel between client and compute node through head-node

# *Case study: Cosmological Data Analysis*

# *Comsological Data Analysis: Motivations*

- Astronomical simulations/observations generate huge amount of data
- This data cannot be loaded into a single machine
- Even if loaded, interaction with user too slow



- Need to parallel analyzer tools capable of
  - Scaling well to large number of processors
  - Provide flexibility to the user

# Salsa

*Write your own piece of Python script*

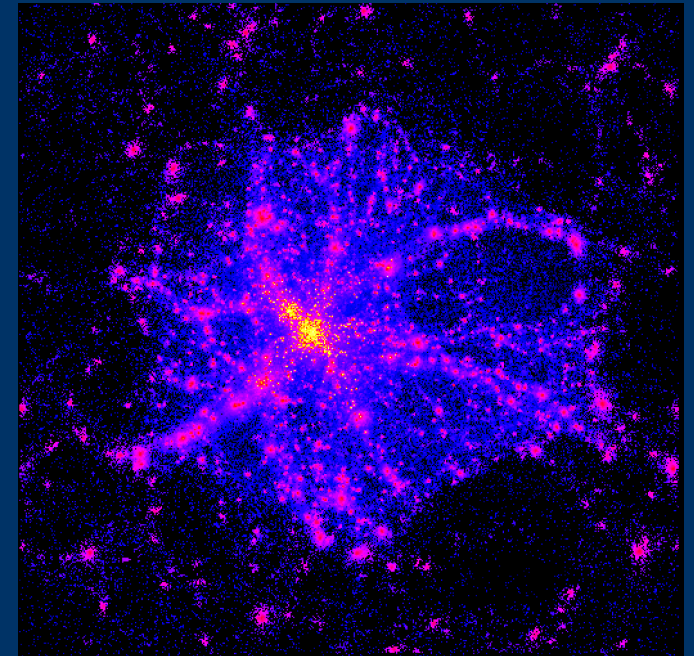
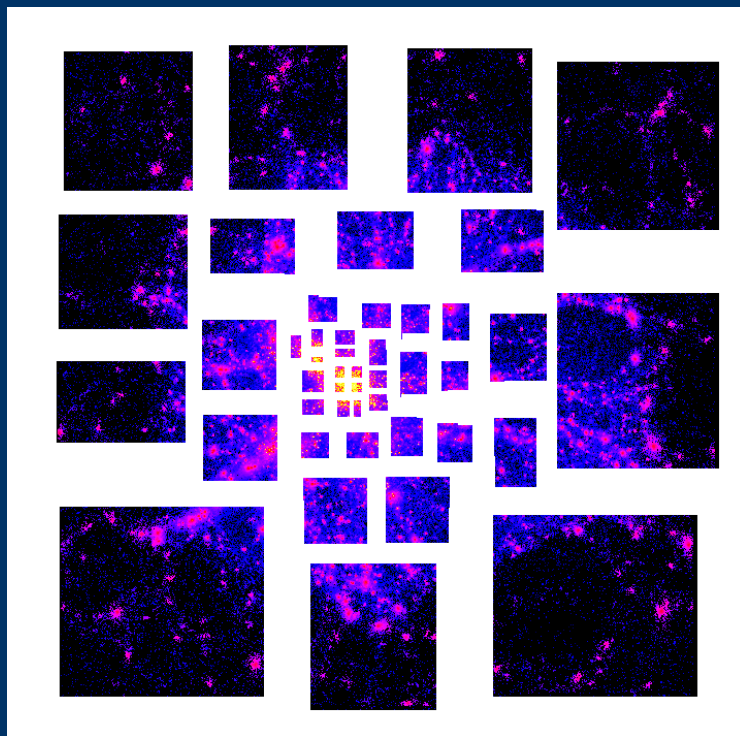
The screenshot displays the Salsa software interface with three main windows:

- Salsa: Code:** Contains Python code for counting particles and printing the result. The output shows "number of particles: 1235400". A button "Execute code on server" is at the bottom.
- Salsa: Simulation View:** Shows a visualization of a galaxy cluster with a color bar at the top. Below the visualization are controls for coloring (Density), group (Potentia...), and various sliders (Down, Up, Left, Right, Cntr, Clock). A checkbox for "Splatter Visual" is also present.
- Salsa: Group Manager:** A dialog box for defining groups. It shows a list of groups with "Potential group" selected. The "Group definition" section includes fields for "Group name" (Potential group), "Attribute" (potential), "Minimum value" (-2.32508E0), and "Maximum value" (-3E-2). Buttons for "Create new group" and "Apply changes" are at the bottom.

Collaboration with  
Prof. Quinn,  
(U. Washington)  
and Prof. Lawlor  
(U. Alaska)

# LiveViz

- Every piece is represented by a chare



- Under integration in ChaNGa (simulator)

# How well are we doing?

Salsa application framerate

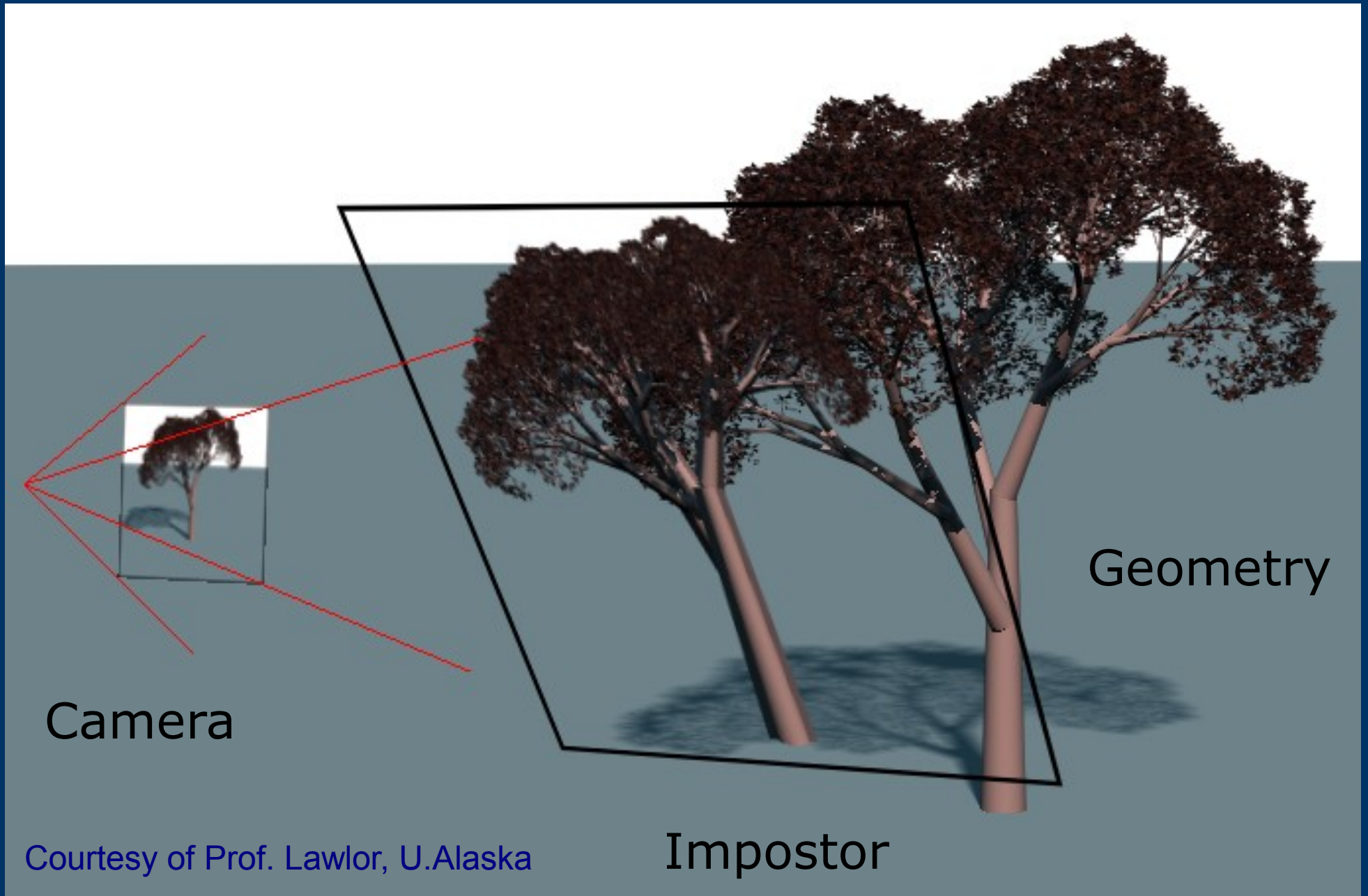
Window size	Gigabit network		2MB/s wireless	
	Bitmap	JPEG	Bitmap	JPEG
256x256	333	25	6	25
512x512	166	24	2	15
1024x1024	50	15	< 1	13
2048x2048	13	4	<< 1	4

Courtesy of Prof. Lawlor, U.Alaska

- JPEG is CPU bound
  - Inefficient on high bandwidth networks
- Bitmap is network bound
  - Bad on slow networks
- The bottleneck is on the client (network or processor)
  - Parallel application: use enough processors



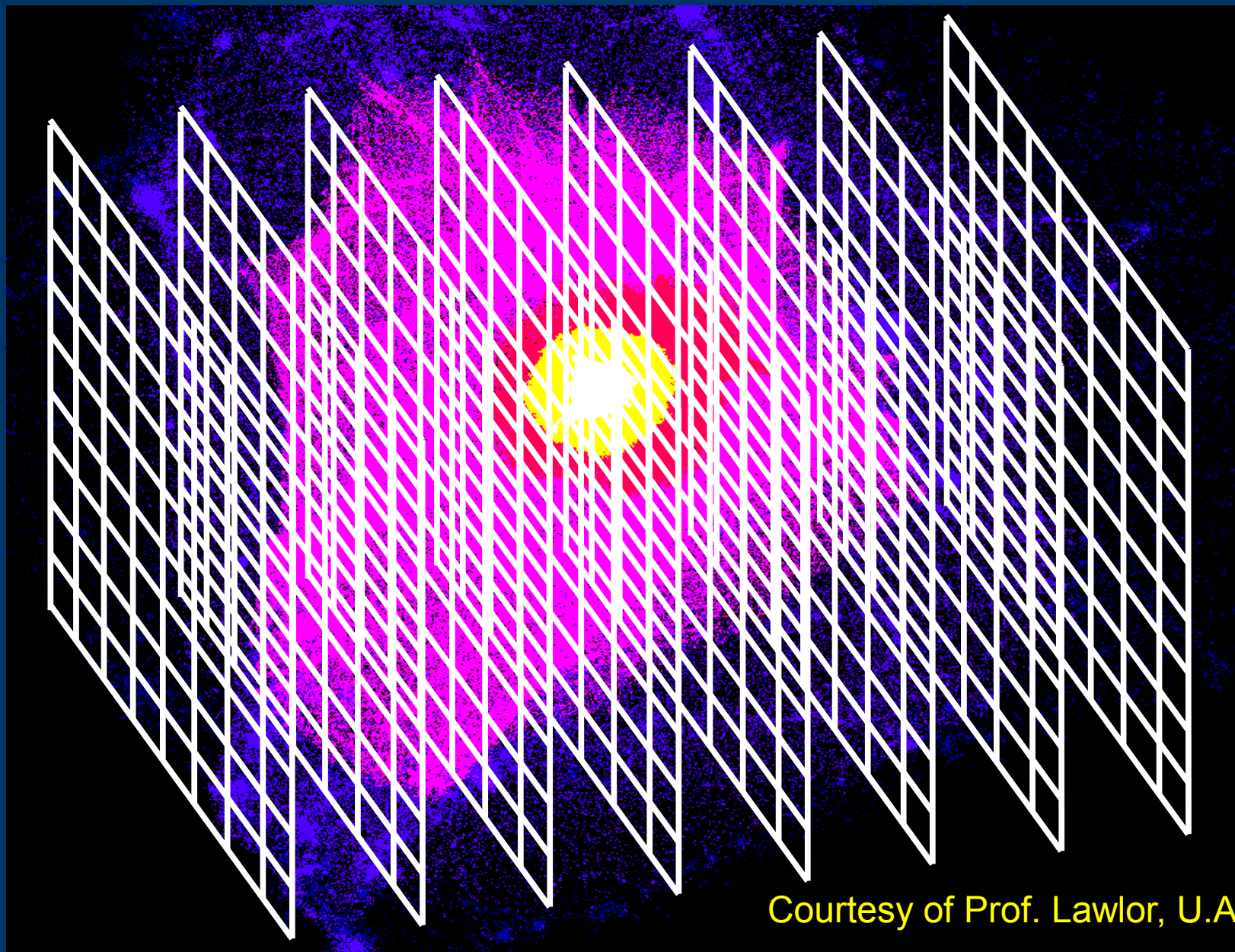
# Impostors: Basic Idea



Courtesy of Prof. Lawlor, U.Alaska

Impostor

# *Particle Set to Volume Impostors*



Courtesy of Prof. Lawlor, U.Alaska

# Summary

- Generic framework (CCS) to connect to a running parallel application, and interact with it
- Demonstration in different scenarios:
  - Parallel debugging
    - Low response time
  - Performance analysis
    - Low runtime overhead
  - Application (cosmological) data analysis
    - High frame rate
- All code is open source and available on our website

# Questions?

Thank you

<http://charm.cs.uiuc.edu/>