Asynchronous computation and a data-driven model of execution

Scott B. Baden, Gregory T. Balls
Dept. of Computer Science and Engineering - UCSD

Tom Bartol and Terry Sejnowski
Salk Institute and UCSD
Requirements of asynchronous applications

- The prevailing bulk synchronous programming model is appropriate for applications that conveniently organize into separate phases of communication and computation.
- But what about applications that violate this assumption?
  - Asynchronous parallel simulations
  - Data driven algorithms that couple external data sources to ongoing computation
  - Latency tolerant formulations of synchronous algorithms
  - Coupled multiphysics applications
Motivating simulation: MCell

- Monte Carlo simulator of cellular microphysiology
- Biochemical reaction dynamics in realistic 3D microenvironments
- Brownian dynamics of individual molecules and their chemical interactions
- Developed at the Salk Institute and Pittsburgh Supercomputing Center by Tom Bartol and Joel Stiles
- 600+ users (First released in 1997)
- MCell-K: parallel variant implemented with KeLP
Animation

- Simulation of a chick ciliary ganglion synapse
  - A real-world problem
  - 400,000 polygons in the surface
  - Approximately 40,000 molecules diffusing
  - Approximately 500,000 surface receptors
Movie

Parallel Simulation of Cellular Microphysiology with MCell-K
Issues in parallelization

• Particles move over a sequence of timesteps
• React with embedded 2D surfaces - cell membranes
• We have to solve two problems
  – How do we know when the current timestep has completed?
  – How and when do we transmit particles among processors?
Asynchronous computation with Tarragon

- **ITR**: Asynchronous execution for scalable simulation of cell physiology
- **Tarragon**: asynchronous data driven model of execution
  - A non-bulk synchronous model
  - Communication and computation are coupled activities; they do not execute as distinct phases
  - Scheduling of tasks handled dynamically according to the flow of data and the progress of computation
- **Cleaner treatment of migrating particles**
  - change owners dynamically
  - avoid subtimestepping which exacerbates load imbalancing
- **Processor virtualization**
  - Many-to-one task assignments
  - Automated load balancing via workload migration
  - Finer grained intermittent communication
Tarragon API

- We express parallelism in an abstract form
- A **task graph** describes the partial ordering of tasks
  - Vertices $\rightarrow$ computation
  - Edges $\rightarrow$ dependences
- A background thread called the **mover-dispatcher** provides available tasks, processes completions, invokes a scheduler
Tarragon in context

- Data driven techniques used in DataFlow, databases and data intensive applications (Data Cutter, ADR)
- Charm++ [Kalé ’93]
  - Parallelism expressed across object collections by making remote method invocations (message passing)
  - Global name space
  - Task virtualization
- Tarragon
  - Functions operate on local data only: data motion is explicit
  - Tune performance by adjusting task granularity and by decorating the graph with performance metadata
Dynamic data driven applications

- Using Tarragon’s data driven programming model, we can couple external data sources into ongoing computation

- Work in progress, 2 applications

- Dynamic clamping of neurons (Bartol & Sejnowksi, Salk)
  - Feedback MCell simulations of neural microphysiology into living neurons \textit{in vitro} via patch clamping
  - Living and simulated neurons are (virtually) part of the same circuit

- Interactive ray-tracing of dynamic scenes (H. Jensen UCSD)
  - change the lighting, scene, camera angle..
  - Need interactive feel
Acknowledgements and support

• People
  – Urvashi Rao Venkata
  – Jake Sorensen
  – Pietro Cicotti
  – Paul Kelly (Imperial College)

• Support
  – NSF ACI0326013, ACI9619020, IBN9985964, Center for Theoretical Biological Physics PHY0225630
  – Howard Hughes Medical Institute
  – University of California, San Diego; National Partnership for Advanced Computational Infrastructure, San Diego Supercomputer Center (SDSC), Cal-(IT)²
  – ESPRC (visits to Imperial College UK)

• Papers and software:
  http://www-cse.ucsd.edu/groups/hpcl/scg/
  http://www.mcell.cnl.salk.edu/
A look inside the Run Time System

- E = Execution engines
- M = Mover/Dispatcher
Parallel efficiency

<table>
<thead>
<tr>
<th>Processors</th>
<th>Run Time</th>
<th>Eff. vs. 16</th>
<th>Eff. vs. 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4903 s</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>32</td>
<td>2652 s</td>
<td>92%</td>
<td>-</td>
</tr>
<tr>
<td>64</td>
<td>1450 s</td>
<td>85%</td>
<td>91%</td>
</tr>
</tbody>
</table>

• Communication costs for this algorithm are low on BlueHorizon:
  – Communicating a few thousand molecules: < 1 ms
  – All-reduce: ~ a few hundred µs for 64 procs
• Each time step requires ~ 1 s of computation
Load balancing - diffusion

Unbounded Diffusion

- 1 release site, 10,000 molecules
- 8 simulated processors
Load balancing - ganglion

Ciliary Ganglion

- 18 release sites, 1000 molecules each
- 8 simulated processors
Motivating application

• Cerebellar Glomerulus
  – 2 CPU-months on a single processor
  – 24 GB of RAM
  – 20 million Ca$^{2+}$ ions, 10 million polygons

• With serial MCell
  – Run 1/8 of the domain of the problem on a single processor
  – Reduced resolution

• Scalable KeLP version – MCell-K
  – Running on up to 128 processors on Blue Horizon
  – Collaboration involving Greg Balls (UCSD), Srinivas Turaga, Tilman Kispersky (UCSD/Salk), Tom Bartol (Salk), Terry Sejnowski (Salk)
Chick ciliary ganglion synapse
Chick ciliary ganglion synapse
Diffusion and interactions

- **Ligands**: neurotransmitter molecules
- Bind to sites under constraints
- Bounce off of surfaces
- Uneven distributions in space and time
MCell on a parallel computer

- Partition boundary splits up the problem over multiple processors
- As ligands cross a processor boundary, we color them yellow
Observations

• Avoids unnecessary sequencing constraints that inhibit overlap of communication
  – Applications are rich in potential parallelism
  – There is considerable latent parallelism among and within communication and computation tasks
  – Avoids the need to hard code the scheduling strategy into the application

• Optimal ordering of communication and computation varies across generational changes in technology
  – It can also vary dynamically
  – Hard for the user to get this right
  – Compare with Instruction Level Parallelism within the core of a central processor
Data driven execution

- Construct a task graph indicating the data dependences
- A task suspends until the required communication completes at which point the task is runnable
- Tarragon run time system schedules runnable tasks according to the flow of data in the task graph
Benefits

• Tolerate unpredictable or irregular latencies at different scales
  – Communication and computation are coupled activities rather than distinct phases
  – Tune slackness to improve communication pipelining

• Flexible Scheduling
  – Schedulers may be freely substituted, and may be application specific [AppLeS, Berman]
  – Performance meta-data enable us to alter the execution order without having to change the scheduler

• Run time system optimizes execution ordering without entailing heroic reprogramming
Slackness

- Multiple tasks per processing module
- Improve communication pipelining: communication occurs incrementally and in parallel with computation
- Tolerate irregular communication delays
- Treat load balancing as a scheduling activity (migration)
Summary

• **Asynchronous task graph execution model**
  – Non bulk synchronous execution model …
  – communication and computation are coupled activities rather than distinct phases
  – Tolerate unpredictable or irregular task and communication latencies
  – Performance meta-data decorate the graph to provide scheduling hints

• **Generalizations**
  – Very long latencies (on the order of milliseconds)
  – Application coupling
  – Incorporate dynamic data sources into ongoing computations
Parallelization strategy

• To detect termination we divide each timestep into sub-timesteps
  – We continue to the next time step only when there are no more ligands to update or communicate
  – Currently implemented with a barrier

• Aggregate communication of ligands to amortize message starts
  – Buffers and message lengths scaled automatically

• Uniform static decomposition
  – Work on dynamic load balancing is under way

• Implemented in KeLP
Computational results

- Chick ciliary ganglion
  - ~400k surface triangles
  - 192 release sites (max of 550)
  - Each site releases 5000 ligands at t=0 (960k total)
  - 2500 time steps
- “Persistent ligand” case
  - Enzymes that destroy ligands are made less effective
  - Most ligands are present at the end of the simulation
- Report summary statistics in *epochs* of 100 time steps
- Ran on NPACI BlueHorizon: 16, 32, & 64 processors
Performance on NPACI BlueHorizon

- Running times scale well
Performance prediction

Running Time per Epoch
persistent ligand case

- Running times are predicted well by maximum ligands per processor
Load imbalance

- Maximum load close to 2x average load
Uneven workload distributions

- Loads vary significantly, dynamically

![Graph showing ligands per processor over epochs for heavily loaded processors - persistent ligand case.](image)
The Future

- Ligands may bounce across processor boundaries
- Detecting termination is expensive
- Motivates asynchronous, execution, novel scheduling
- Molecule-molecule interactions
- Hybrid methods
  - Determine certain quantities at a coarser level using 3D tetrahedral mesh
  - Integrate the two representations

Wire frame view of rat diaphragm synapse courtesy Tom Bartol and Joel Stiles
Current and future work

- Asynchronous computation
- Large scale simulations
- Load balancing
- Predictive modeling (U. Rao Venkata)
- Parameter sweep
Conclusions

• We’ve looked at an asynchronous data driven programming model with motivating applications
  – Communication tolerance
  – Asynchronous simulations
  – Dynamic data driven applications that couple simulations with the real world

• Appropriate programming model and abstractions simplify the design (scheduling and partitioning)

• Generalizations
  – Very long latencies (on the order of milliseconds)
  – Application coupling
  – Incorporate dynamic data sources into ongoing computations