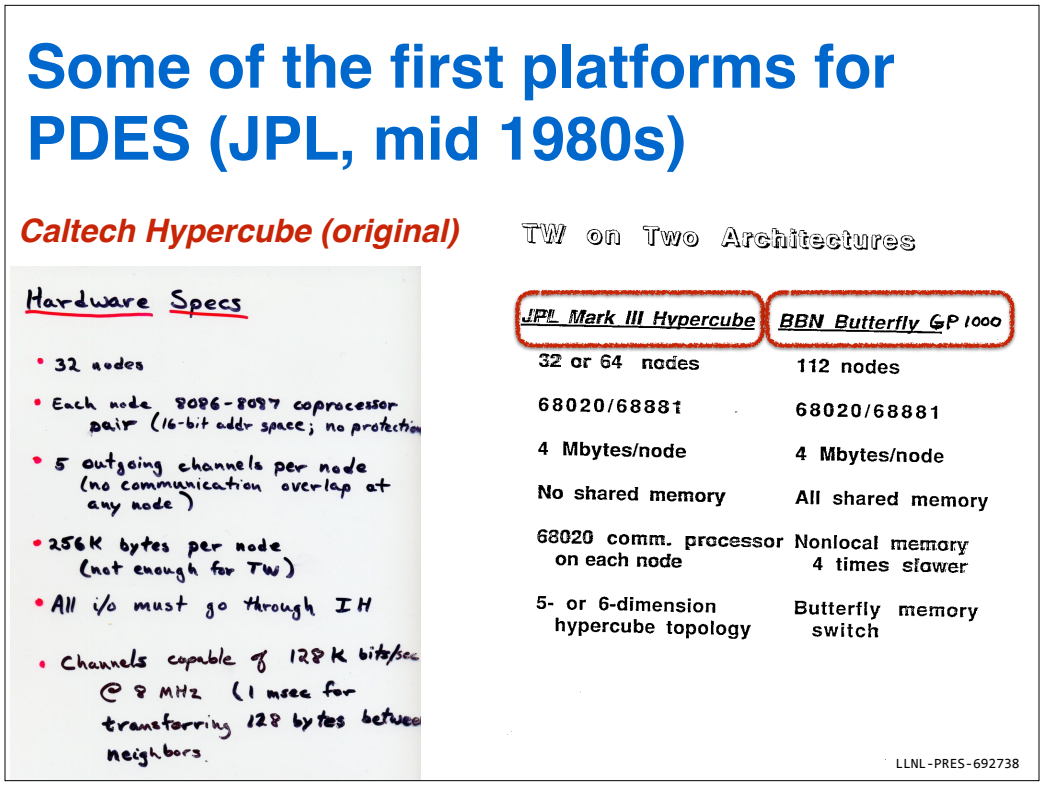
Past, Present, and Future of Parallel Discrete Event Simulation

SIGSIM - PADS Keynote
Banff Centre, Alberta
May 2016

David Jefferson
Lawrence Livermore National Laboratory

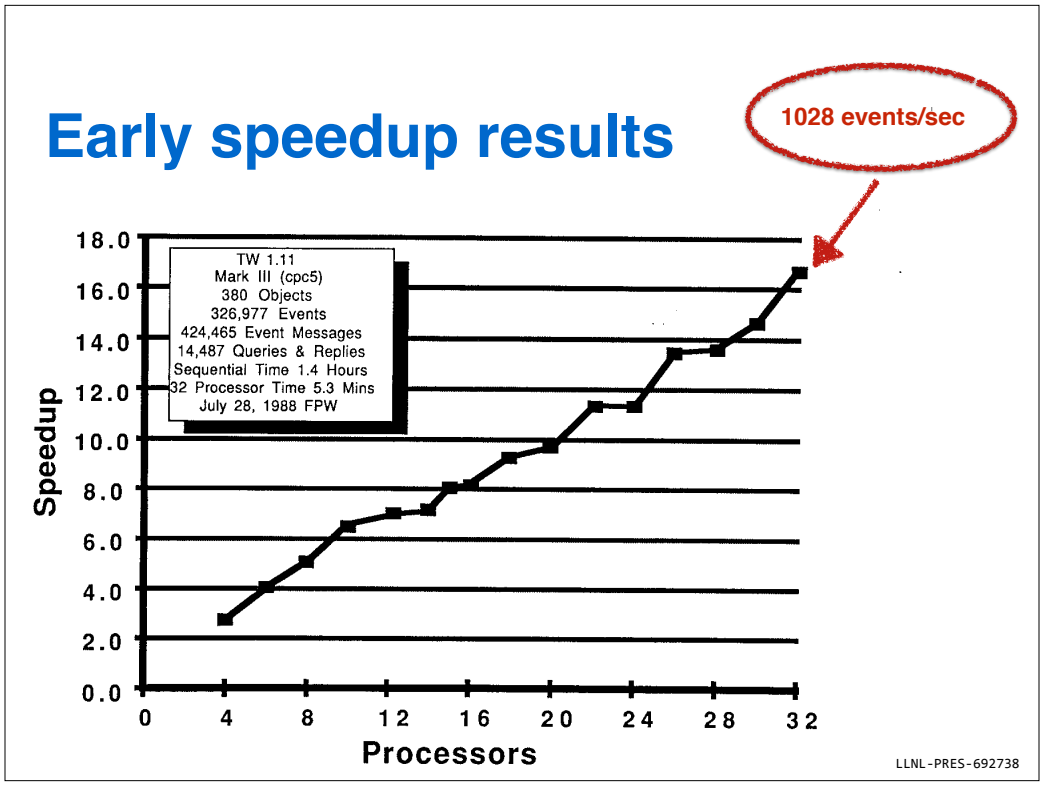
LLNL-PRES-692738

The Past



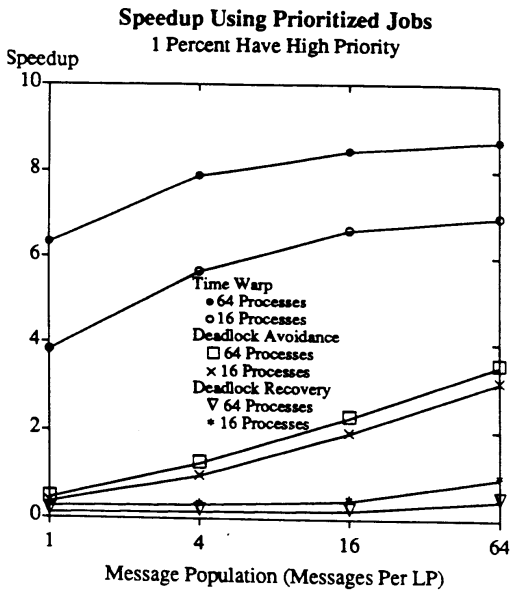
These are ancient slides from the 1980s that I used to describe the hardware specs of the machines we used in the JPL Time Warp project to do the first performance studies of optimistic simulation using TWOS, the Time Warp Operating System.

The slide on the left was hand written on plastic, before the era of PCs, and describes the original Caltech Hypercube.
The slide on the right is from a couple of years later, and was printed by a first gen Macintosh. It describes the JPL Mark III hypercube, and the BBN Butterfly.



This slide shows a speedup curve for a string scaling study done in 1988 of a military combat model executing on the Mark III Hypercube under TWOS. It shows that we achieved a speedup of over 16x using 32 processors, compared to a sequential execution of the same model. (The sequential simulation was done on a special compatible sequential discrete event simulator, not using TWOS on a single node.) The peak performance on 32 processors was about 1028 events/sec.

Fujimoto: First fair comparison between conservative and optimistic synchronization



LLNL-PRES-692738

This slide reproduces a graph that Fujimoto published in the late ‘80s using GTW (Georgia Tech Time Warp). (Sorry - I don’t have the reference handy.) It was the first fair comparison between conservative and optimistic methods of synchronization on the same problem and platform. It was particularly well done because the conservative and optimistic runs were done on the same simulator, sharing almost all of the code, so that the only difference was the synchronization. This study clearly showed that at least on this one model of a job scheduler Time Warp dramatically outperformed the two conservative methods proposed by Chandy and Misra. Of course there were other models in which the reverse was true. But this study showed that optimistic methods could be competitive with and sometimes superior to conservative methods.

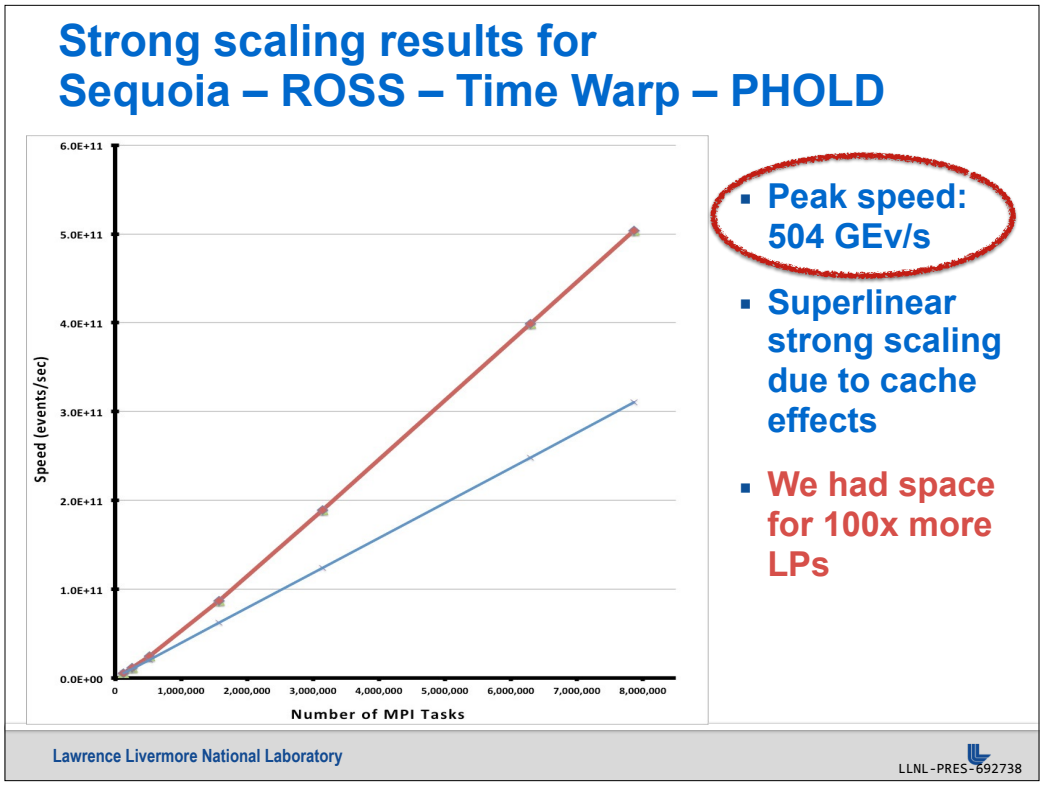
LLNL's *Sequoia* Blue Gene/Q

"Super Sequoia"

1,966,080 A2 cores

5-D Torus: 20x16x16x12x2

This is the *Sequoia* Blue Gene/Q supercomputer at LLNL. At the time of the PHOLD test described on the next slide it was configured at 120 racks containing almost 2 million IBM A2 cores running at 1.6 GHz each.



This study, published at PADS 2013 shows a strong scaling study of a PHold benchmark running on configurations ranging from 2 racks to 120 racks on *Sequoia*. It reached a sustained performance at the high end of *504 billion* events/sec. The blue curve shows what the performance would have been if the scaling had been perfectly linear, i.e. with the 60-fold increase in parallel threads from 131,072 to 7,864,320 we would expect a 60-fold increase in speed. The red curve shows the performance actually achieved, which was much better than that. The superlinear performance was primarily due to cache effects, i.e. when the simulation was spread over many more processors much more of the model code and data would fit in cache, boosting performance beyond that achievable with just parallelism. This was a world record speed for PDES, and will probably stand for several more years because supercomputer architectures are evolving in a direction that is not favorable to PDES.

Billions and billions of LPs

"*Sequoia* "scale

- Room for 25 billion PHold LPs

"Planetary scale"

- 7 billion people
- 1 billion motor vehicles
- 2 billion cattle
- 4 billion IPv4 addresses

"Material Scale"

10^{18} atoms — visible crystal

"Biological scale"

- **Neurons in brain**
  - ~86 billion neurons in human brain, 10^{14}-10^{15} synapses
  - ~70 million neurons in house mouse brain, 10^{11} synapses
- **T-cells in immune system**
  - ~5 billion T-cells in human immune system ($1{,}000/mm^3$ * $5{,}000{,}000\ mm^{3)}$
  - ~1.5 million T-cells in mouse immune system
- **Protein molecules in cell**
  - ~10 billion protein molecules in a human cell ($3e6/\mu m^3$ x $3000\mu m^3$ for HeLa cell)
  - <1 million in a typical bacterium

If you can build the model, chances are we can execute it.

Further parallelism needed more for ensemble of executions than internal parallelism.

LLNL-PRES-692738

This slide indicates the scales that we need to simulate at for biological models (Biological Scale), models whose size is proportional to the number of people on Earth (Planetary Scale), and models of macroscopic materials represented at the atomic level ("Material Scale"). The Sequoia supercomputer has enough RAM for 25 billion PHold LPs, so it is at approximately Planetary Scale already. It is also a Biological Scale already, at least for a mouse. It is still some distance away from material scale defined by human visibility.

The point is that we are now limited more by our ability to build and validate complex models than we are by our ability to execute them.

But still more parallelism will be needed in the future because we don't want to run a model just once — we have to run it thousands or more times to do parametric and statistical studies of the *ensemble* of behaviors that the model can display.

The Present

Vast improvements in scale and speed. But…

Can we keep that scale increase going? GPUs?

Even after 35 years, PDES simulations are still very fragile, especially optimistic synchronization.

They still not easily portable to new platforms.

They regularly fail, or get terrible performance, often for reasons difficult to diagnose.

Still not ready for prime time.

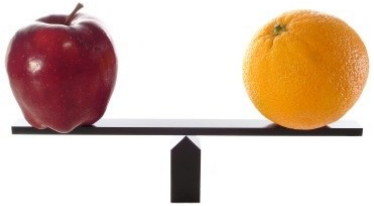
Problems we still have after 35 years

- Dynamic ~~load balancing~~ configuration management
- Throttling optimism
- Memory management (optimistic)
- Rollback overhead
- Many mechanical issues

Dynamic ~~load balancing~~ configuration management

Fundamental problem in all parallel computations, but especially irregular ones

- **measure performance (instrumentation)**
- **decide on config changes**
- **migrate load**
- **do it all concurrently if possible**



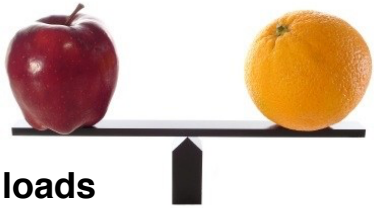
LLNL-PRES-692738

I prefer the term “dynamic config management” to “dynamic load balancing” because there are many reasons to move load around the platform besides “balancing” core usage, including balancing channel usage, reducing communication latency, decreasing memory pressure, optimizing I/O performance, etc.

Dynamic ~~load balancing~~ configuration management

Fundamental problem in all parallel computations, but especially irregular ones

- **measure performance (instrumentation)**
- **decide on config changes**
- **migrate load**
- **do it all concurrently if possible**



Not just concerned with processor loads

- **channel bandwidth loads**
- **power use and processor speed**
- **memory pressure**
- **I/O traffic**
- **comm latency**

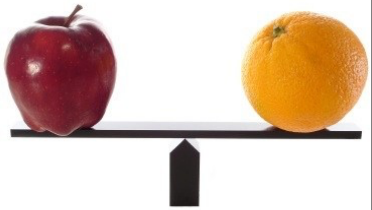
LLNL-PRES-692738

I prefer the term “dynamic config management” to “dynamic load balancing” because there are many reasons to move load around the platform besides “balancing” core usage, including balancing channel usage, reducing communication latency, decreasing memory pressure, optimizing I/O performance, etc.

Dynamic config management

It is different for simulations!

- **We have more information to guide us:**
  - **global temporal coordinate system**
  - **global measure of progress.**
- **We can calculate instantaneous critical paths**
- **What is the goal?**
  - **To advance global simtime as fast as possible**
  - **Progress along *critical path* as fast as possible**

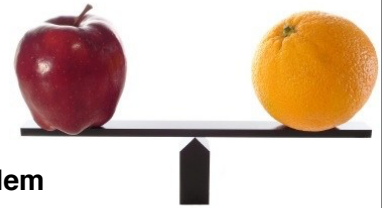


Simulations are different from other computation in that there is a built-in measure of global progress and temporal coordinate system: simulation time. The value of GVT and/or LVT are natural and valuable inputs available for migration decision-making.

Dynamic config management

It is different for simulations!

- **We have more information to guide us:**
  - **global temporal coordinate system**
  - **global measure of progress.**
- **We can calculate instantaneous critical paths**
- **What is the goal?**
  - **To advance global simtime as fast as possible**
  - **Progress along *critical path* as fast as possible**



We need *theory*!

- **This is an classic *online optimization* problem**
- **Config management must be *portable***
- **Which is best: *global or local?***
- **Scalable load management must be *hierarchical* because of federation**
- ***Where is the complexity theory*?**

LLNL-PRES-692738

There are a lot of papers that report empirical experience with one or more “load balancing” algorithms, sometimes on multiple benchmarks and multiple platforms. But there seems to be no real theory behind the subject. It seems to me that there is a golden opportunity waiting for someone to explicate a really solid and portable first principles theory of online dynamic configuration management, its costs and benefits. A good analogy would be the working set theory of demand paging that put all virtual memory systems on a firm footing in the 1970s and still guides thinking today.

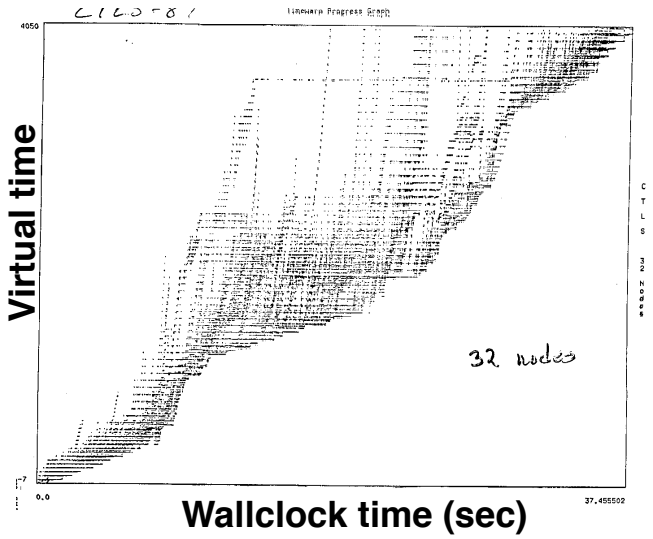
inc

- **Fills up memory**
- **Causes rollback cascades**
- **Must throttle optimism — to make it more *conservative*.**
- **Currently we have literature describing *ad hoc* mechanisms and some (forgotten) theory**
- **Again, we need more *theory***

LLNL-PRES-692738

Once again, as with dynamic config management, there are lots of papers hat have tested one or more algorithms for throttling and reported the results on one or more benchmarks. But what we need is a robust, portable *theory* to guide the development of throttling algorithms.

Throttling excess optimism



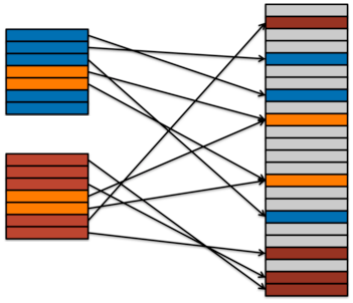
Battle simulation showing excess optimism and the need for throttling [JPL, 1987 (?)]

LLNL-PRES-692738

This is a slide from about 1987 depicting events executed in a war-game simulation on the JPL Mark III hypercube. Each dot represents an event executed, including both committed and rolled back. There is a nice clean lower envelope that tracks GVT, but there is an enormous amount of activity far ahead of GVT that represents excess optimism that should have been throttled. This demonstrates that the issue was apparent 30 years ago, and is still with us.

Memory management

- **Running out of memory is one of the most common failure modes with optimistic synch.**
  - **Conservative synch *can* require much more memory than optimistic.**
  - **But not normally — AFIK only for specially constructed examples.**
- **Fossil collection and throttling help, but are definitely not the full answer.**
- **We have had a good global protocol for memory management (Cancelback) for 25 years.**
  - **Provably good properties. (Space-optimal in shared memory!)**
  - **Implemented (I think) only once, at JPL.**
  - **Never properly studied.**



LLNL-PRES-692738

Optimistic simulations require good memory management. All of the memory used by the simulator has to be managed with a common system, including events that have been executed but not committed, events enqueued in the future, anti messages saved in case of rollback, and state date saved to in case of rollback.

We have had a provably memory-optimal protocol for this called Cancelback for about 30 years, but it is rarely if ever implemented. We need to revive it and test it and make it more robust, or create a better alternative.

(It is not well known, but conservative methods are not, and cannot be, space optimal in all cases. In fact, it is possible to construct cases where they must take arbitrarily more space than would be required by an optimistic method using Cancelback.)

Rollback Overhead

- **Reversible computation: the best current idea in reducing rollback overhead**
  - Carothers, Perumalla, Fujimoto 1999
  - Quaglia, Pellegrini, et al
  - Perumalla book (2013)
- ***Backstroke* is a recent approach directly inspired by the Carothers, *et al* paper**

Backstroke: P → P^+

P can be essentially *any* C++/11 (or C) program

multiple source
    and header files
assignments
conditionals
switches
loops with
    continue
    break
gotos
return
try-catch
iterators
functions
    any signature
recursion

all primitive types
arrays
classes
methods
constructors / destructors
operators
inheritance
pointers (undisciplined)
smart pointers
new
delete
STL container classes
    vectors, maps, sets, etc.
templates

LLNL-PRES-692738

Backstroke is a program that transforms irreversible programs P into reversible form P+. It is a source-to-source transformation that works on any program written in C or C++. The entire C and C++ languages are supported, including the not-so-clean or disciplined features and key parts of the STL.

Much more to do with various PDES Mechanisms

- **various rollback and cancellation mechanisms**
- **various lookahead algorithms**
- **dynamic LP creation/destruction**
- **transparent LP replication (cloning)**
- **LP arrays and collections with collective events**
- **optimistic file systems and databases**

LLNL-PRES-692738

Various other simulation issues remain to be addressed properly. There is still plenty of room for research.

The Future

Research challenges for the next decade

- Unification of time-stepped, conservative, and optimistic synchronization
- Unification of discrete and continuous simulation
- General coupling, federation, nesting and componentization of simulations
- Ensembles as the top level unit of simulation

LLNL-PRES-692738

Unification of time-stepped, conservative, and optimistic synchronization

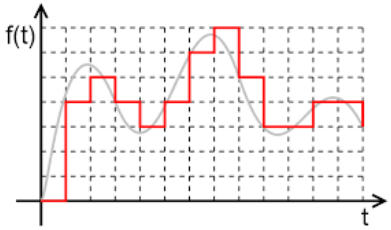
- **Unification of conservative and optimistic synchronization:**
  - **Execute conservatively using lookahead information**
  - **Continue execution optimistically until throttled**
  - **Rollback, fossil collection, and commitment modified accordingly**
- **Unification with time-stepped simulation actually harder**
  - **Time-stepped models don't use event messages**
  - **Algorithms not designed for state changes between time steps**
  - **Interpolation code must be added before they can be coupled, and they must be capable of accepting arbitrary events.**

LLNL-PRES-692738

Unification of conservative, optimistic, and throttled synchronization. We need a single scheme that executes faster and conservatively when there is good lookahead information, and optimistically when there is not, and where the transition from one to the other is seamless and transparent.

Time stepped simulation makes harder static assumptions than either conservative or optimistic simulation. We need a methodology that makes time stepped simulations comparable with, and able to be coupled to, even driven simulation, or even to another time-stepped simulation. This is not easy, and I suspect the best answer will be to just stop building time stepped simulations and standardize on event driven.

Unification of continuous and discrete simulation



- **Complex simulations often have both *continuous* and *discrete* components**
  - **controlled systems (aircraft, auto engine, power grid)**
  - **materials continuous at coarse scale, discrete at fine scale**
- **Time-stepped continuous simulators for ODE and PDE integration must be (re)designed for coupling to discrete event simulations.**
- **Discretize variable rate integrators into discrete event form, rather than time-stepped.**

LLNL-PRES-692738

Continuous models are generally built as time-stepped simulations. From the last slide I argued that we may need to move from time stepped simulations to event driven in all cases, which means we need efficient event driven integration algorithms for continuous models defined by ODEs or PDEs.

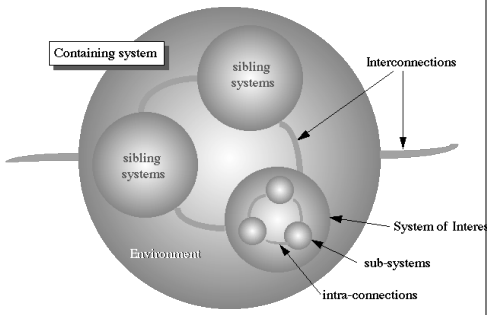
Coupling of Simulations

Vital to be able to *couple*, *federate*, or *embed separately developed parallel* simulations

- Fairly straightforward for conventional processes and componentized applications
- …but not with simulations yet.

Coupling modes:

- boundary-boundary (peers)
- subcomponent
- scale coupling (coarse-fine)
- state space coupling



What we want

Coupling of Simulations

- **To couple *different kinds of separately developed simulation* the coupling mechanism must incorporate exchange of**
  - **namespace & coordinate system data**
  - **event message routing data (when there is load migration)**
  - **GVT data**
  - **throttling data**
  - **lookahead data**
  - **dynamic performance data**
  - **error & interpolation data (couple continuous models)**
- **HLA was a noble experiment, but needs to be reconsidered, re-attempted**
- **Coupling standards: an activity for SIGSIM?**

What we've got

LLNL-PRES-692738

The ability to couple simulations to one another is an absolutely critical requirement. It is the only way we con construct large, complex simulations from smaller ones the way we construct large, complex physical systems from smaller ones. We need to be able couple two simulations so each is the boundary of the other, and we need to be able to nest simulations dynamically, so that one operates at course scale and others operate in the same parts of simulation space, but at finer scale.

Simulation Ensembles

- **Useful simulations usually have *n* inputs, *p* parameters and *r* random variables**
- **Forms an (*n+p+r*)-dimensional space of model behaviors**
- **The goal of a simulation study is to answer questions about that *space* or *ensemble* of behaviors.**

No one executes simulation only once. You have to execute it many times, maybe thousands or more times, to explore the parameter space of behaviors it can exhibit, and explore the multivariate distributions of random variables. A structured set of executions of the same model is called an *ensemble.*

Ensemble Studies

- **parameter optimization**
- **parameter sensitivity**
- **exhibit rare combinations of events**
- **estimate means, variances, other moments and distribution parameters, correlation coefficients, etc.**
- **estimate frequencies of rare events**
- **uncertainty quantification**
- **validation against mathematical models, other simulations, or experiment**

LLNL-PRES-692738

There are many reasons to do ensemble studies, not just one. In fact, we should consider the ensemble as the primary unit of simulation. We learn very little from one simulation run, but we answer our research questions by studying the results of an ensemble structures as and optimization study, or a parameter sensitivity study, etc.

.

Ensemble Studies

- **The goal should be to conduct an entire ensemble study *in a single job.***
- **Process must incorporate a *job control API* into the OS**
  - **Allocate groups of nodes**
  - **Launch whole simulation (sub)jobs in parallel on those groups of nodes (with time limits)**
  - **Be notified when each job completes, normally or abnormally**
  - **Reclaim nodes, and launch more ensemble runs as needed**
- **Current cluster OS's often do not allow remote process creation, let alone job initiation**

LLNL-PRES-692738

We really need software support for the notion of an ensemble. This requires the OS to allow a running simulation to allocated nodes and launch another simulation..And the parent simulation must be able to set time and resource limits, and must be notified when the child simulation finishes, normally or abnormally.

35 years of research in PDES

Broy Lamport Bagrodia
Jefferson Fujimoto Misra Chandy Bryant

Unger Jefferson Riley Nicol Fujimoto

It has been a pleasure to be associated with so many significant simulation researchers over the last few decades.