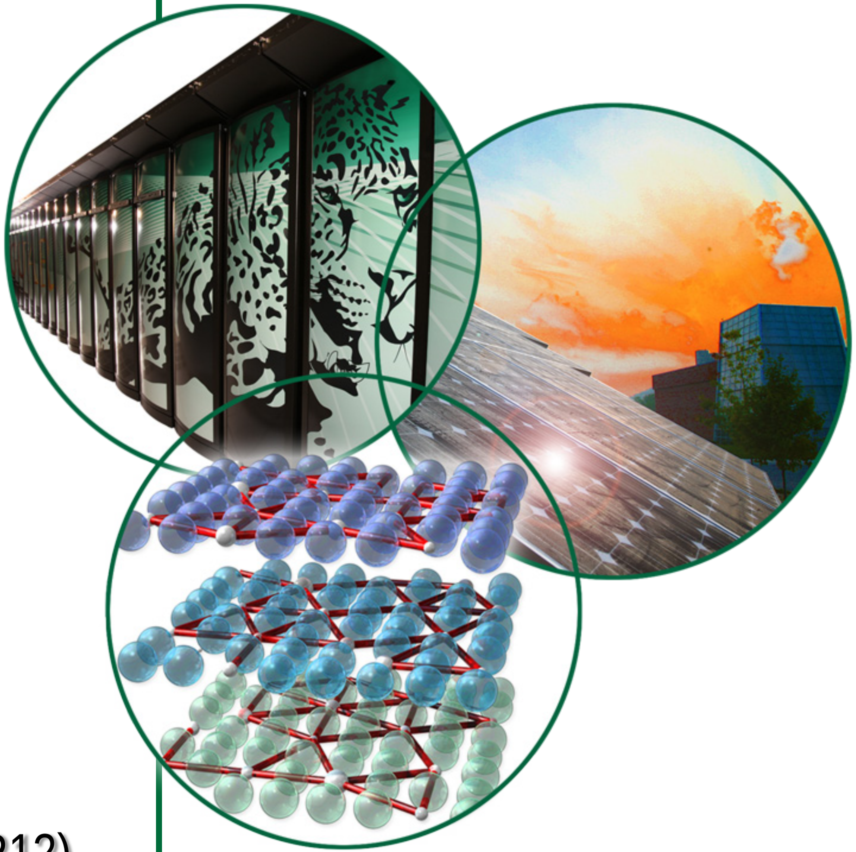
ReveR-SES: Reversible Software Execution Systems

November 2, 2011

Kalyan S. Perumalla
Oak Ridge National Laboratory



Advanced Scientific Computing Advisory Committee (ASCAC) Meeting, Washington, DC

Research funded by DOE Career Award (ERKJR12)

U.S. DEPARTMENT OF ENERGY

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Project Personnel

- **Kalyan S. Perumalla (PI)**
- **Alfred J. Park (Post-doc)**
- **Vladimir A. Protopopescu**
- **Vinod Tipparaju**

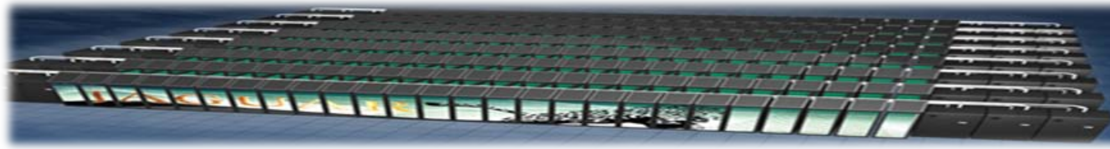
Other Collaborators *no cost*

- Christopher Carothers (RPI)
- David Jefferson (LLNL)

Outline

- **Introduction**
  - Problem and Solution Approach
  - Illustrations
  - Related Work
  - Research Challenges
- **Research Components**
  - Automation
  - Runtime
  - Theory
  - Experimentation
- **Summary**

Problem



- **Very large scale concurrency is burdened by inefficiency at best, and by infeasibility at worst**

  Underlying (software) challenges include synchronization, fault tolerance, and debugging

- **Need to explore paradigm alternative(s) to overcome challenges**

Simulation execution is forward-only

- Almost all current simulations, models, software are inherently forward-only

Paranoid runtime

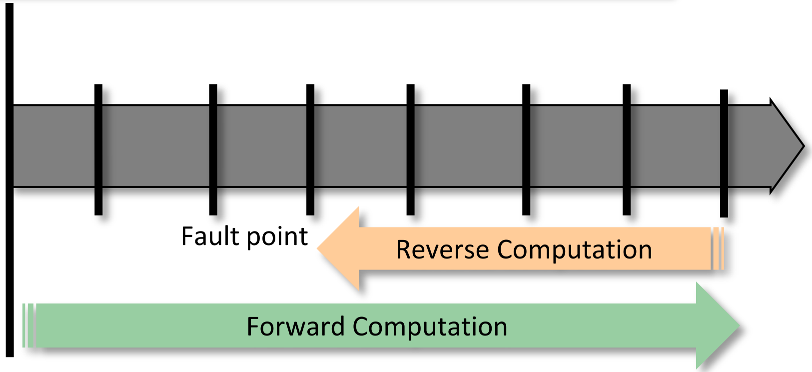
- Needs all components to be always correct

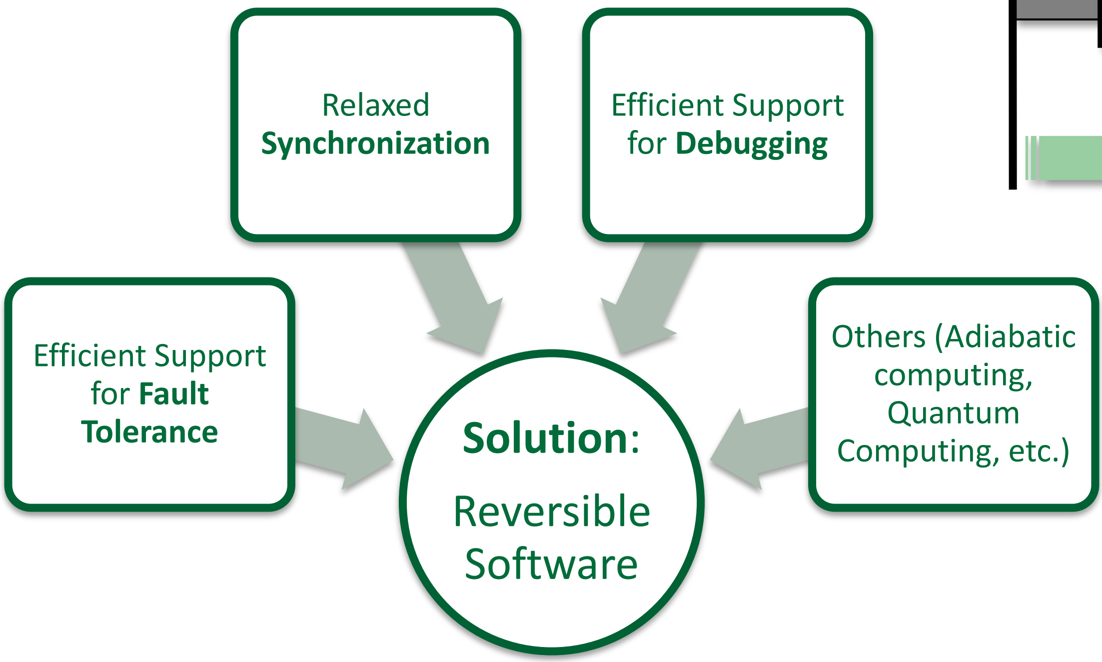
Non-scalable execution

- High synchro-nization cost
- Untenable system-wide fail-safety

Our Solution Approach

- **Relieve runtime paranoia via reversibility**
  Enable software-level reversible execution at scale
- **Perform basic research to enable efficient reversibility**
  Explore, define, refine, implement, test, experiment, study, and **advance the paradigm** of reversible execution for efficient large-scale concurrency





Simplified Illustration of Reversible Software Execution

Traditional Checkpointing	Reversible Software
Undo by saving and restoring e.g. → `{save(x);x = x+1}` ← `{restore(x)}`	• Undo by executing in reverse e.g. → `{ x = x+1 }` ← `{ x = x-1 }`
Disadvantages	Advantages
• Large state memory size	• Reduced state memory size
• Memory copying overheads slow down forward execution	• Reduced overheads; moved from forward to reverse
• Reliance on memory increases energy costs	• Reliance on computation can be more energy-efficient

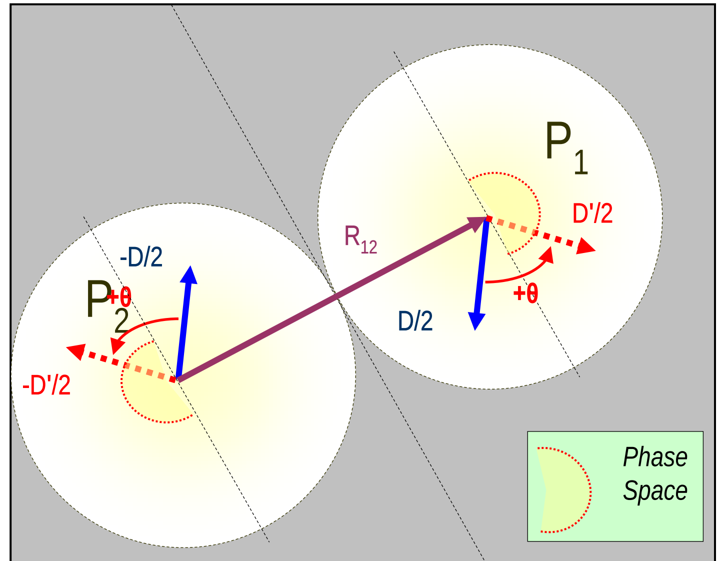
Illustration of a More Complex Reversible Model Execution

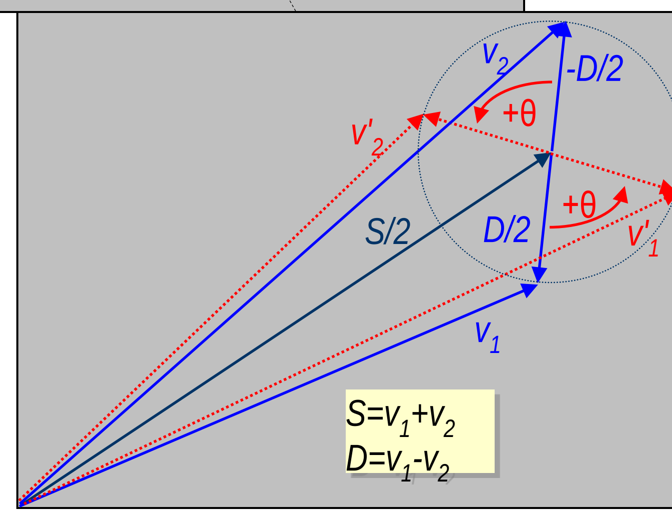
- Example: Simulate elastic collisions reversibly
  - n-particle collision in d dimensions, conserving momentum and energy
  - Incoming velocities X‘, outgoing velocities X
- Traditional, inefficient solution
  - In forward execution, checkpoint X'
  - In reverse execution, restore X‘ from checkpoint
  - Memory M proportional to n, d, and #collisions N_c

    $M=n\times d\times 8\times N_c$ bytes
- New, reversible software solution
  - Generate new reverse code
  - In forward execution, no checkpoint of X'
  - In reverse execution, invoke reversal code to recover X‘ from X
  - Memory dramatically reduced

    E.g., for n=2, 1 bit per collision
  - In fact, zero-memory can also be achieved!

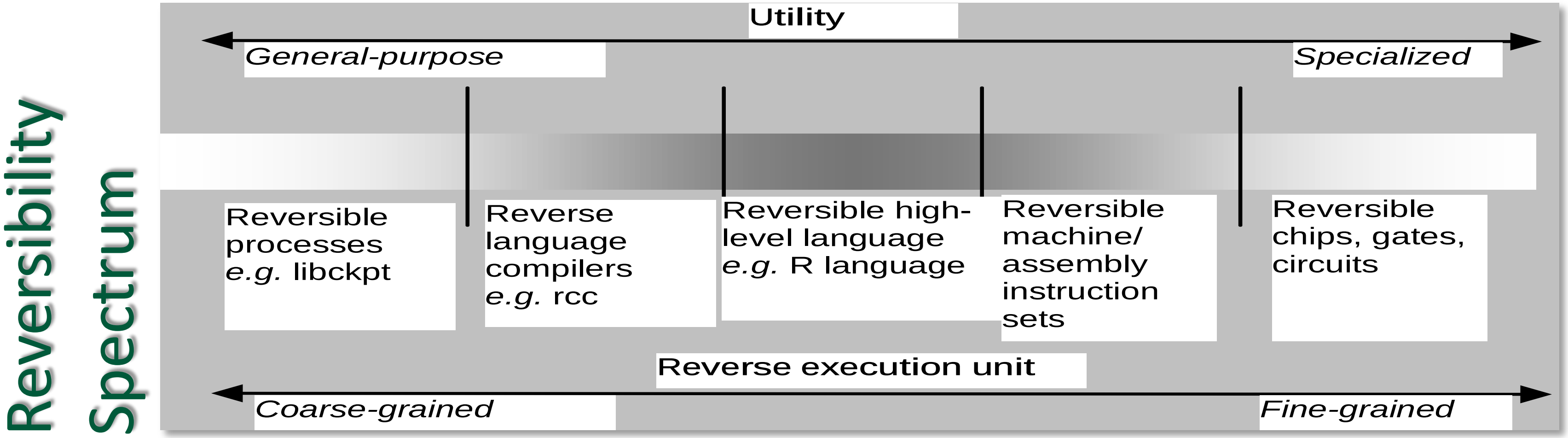
    We have now solved it for n=2, 1≤d≤ 3, and n=3, d=1





Related Work

Time warp, VLIW, Trace scheduling, Adiabatic circuits, Speculative execution, Database Transact., Undo-redo, Functional programming, Cellular automata, Thermodynamics, Quantum computing, …



	Forward-only	Reversible
Synchronous	Old paradigm of all traditional parallel computing	A sub-optimal variant, but improved scalability than forward-only
Asynchronous	Other emerging paradigms (Non-blocking collectives, new languages, programming models, etc.)	**Our proposed new paradigm of fully generalized, staggered, reversible execution by all processors**

Reversible Software: Challenge

- **Conceptually appealing at a high level, but very hard to realize in practice**
- **With very few exceptions, most existing simulations are irreversible**

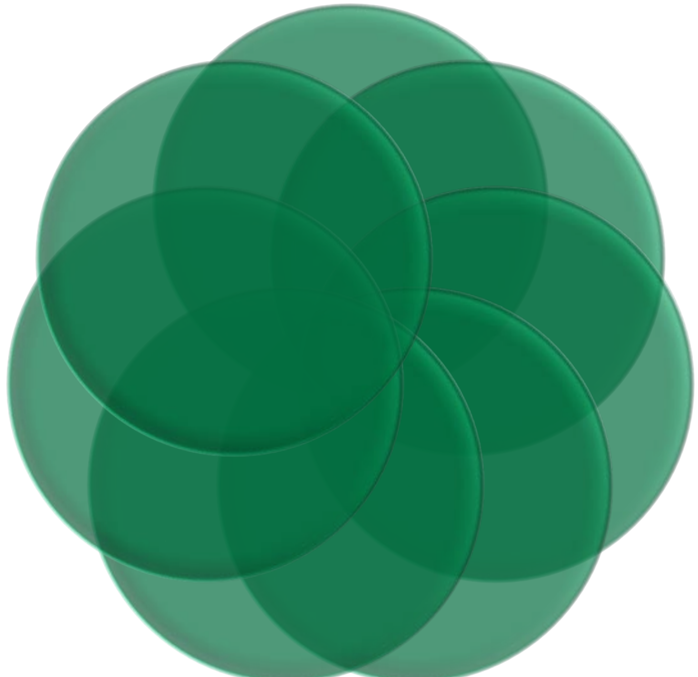
  Very complex control flows, data structures, inter-processor dependencies
- **Almost all existing approaches for undoing computation rely on *memory*, *not* true (reverse) *computation* software**
- **Need theory, methodology, frameworks, proofs-of-concept, and scaling demonstrations to advance reversible software**

Selected Research Challenges in Reversible Software

Generate efficient reversible models of physical systems

(Semi-) Automatically make existing models/code reversible (reverse compiler)

Implement, establish feasibility, and optimize reversibility at scale



Develop backward-compatible interfaces for reversibility

Develop efficient runtime engines for reversible execution at scale

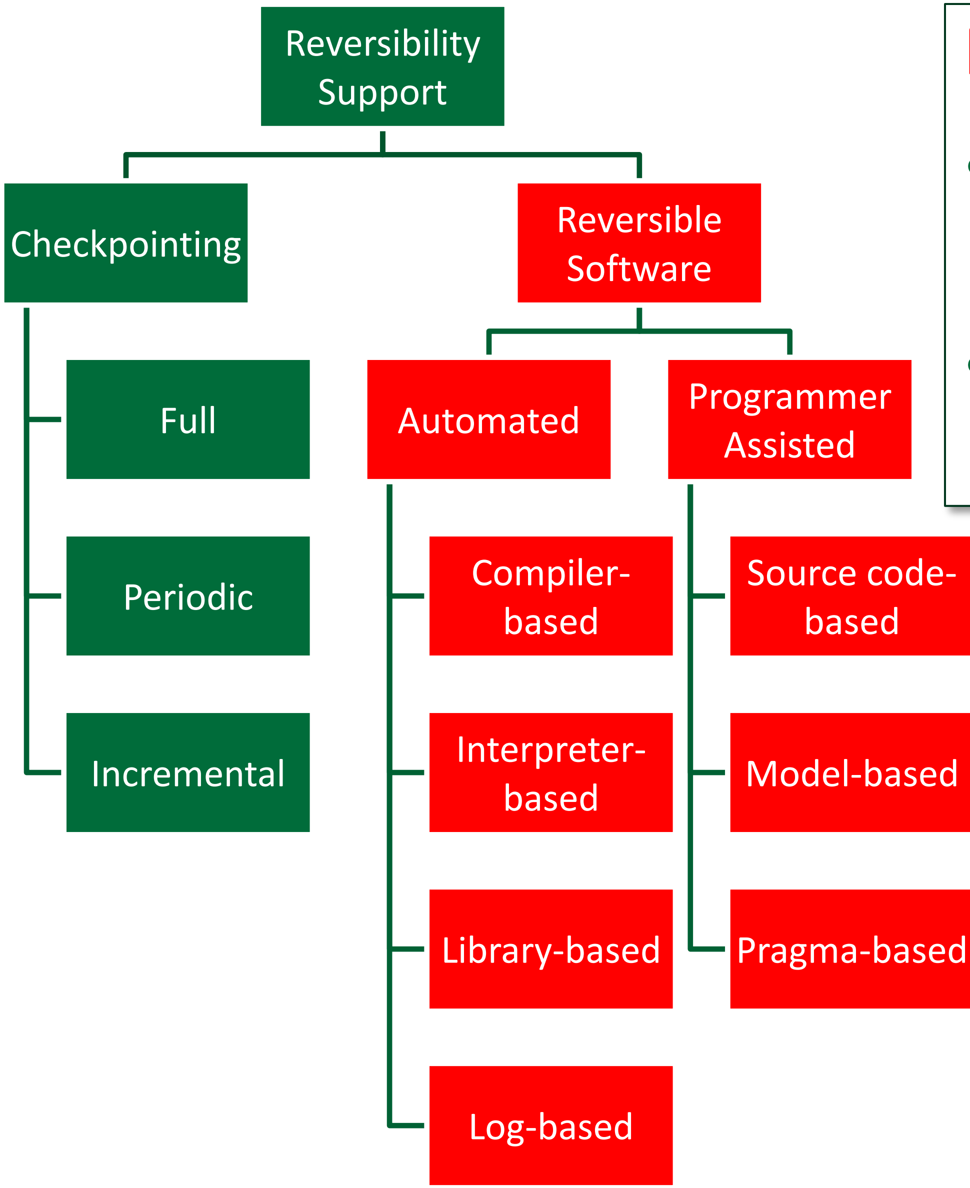
Determine trade-off between reversal, re-computation, and memory costs

Determine, and achieve, memory lower bound for efficient reversibility

Principal Research Components

Automation	Reverse compilers, reversible libraries, …
Runtime	Execution supervisor, extensions to standards, …
Theory	Memory limits, novel modeling approaches, methodologies, …
Experiments	Experimentation system, benchmarks, mini-apps, scaling, …

Our Automation Approach

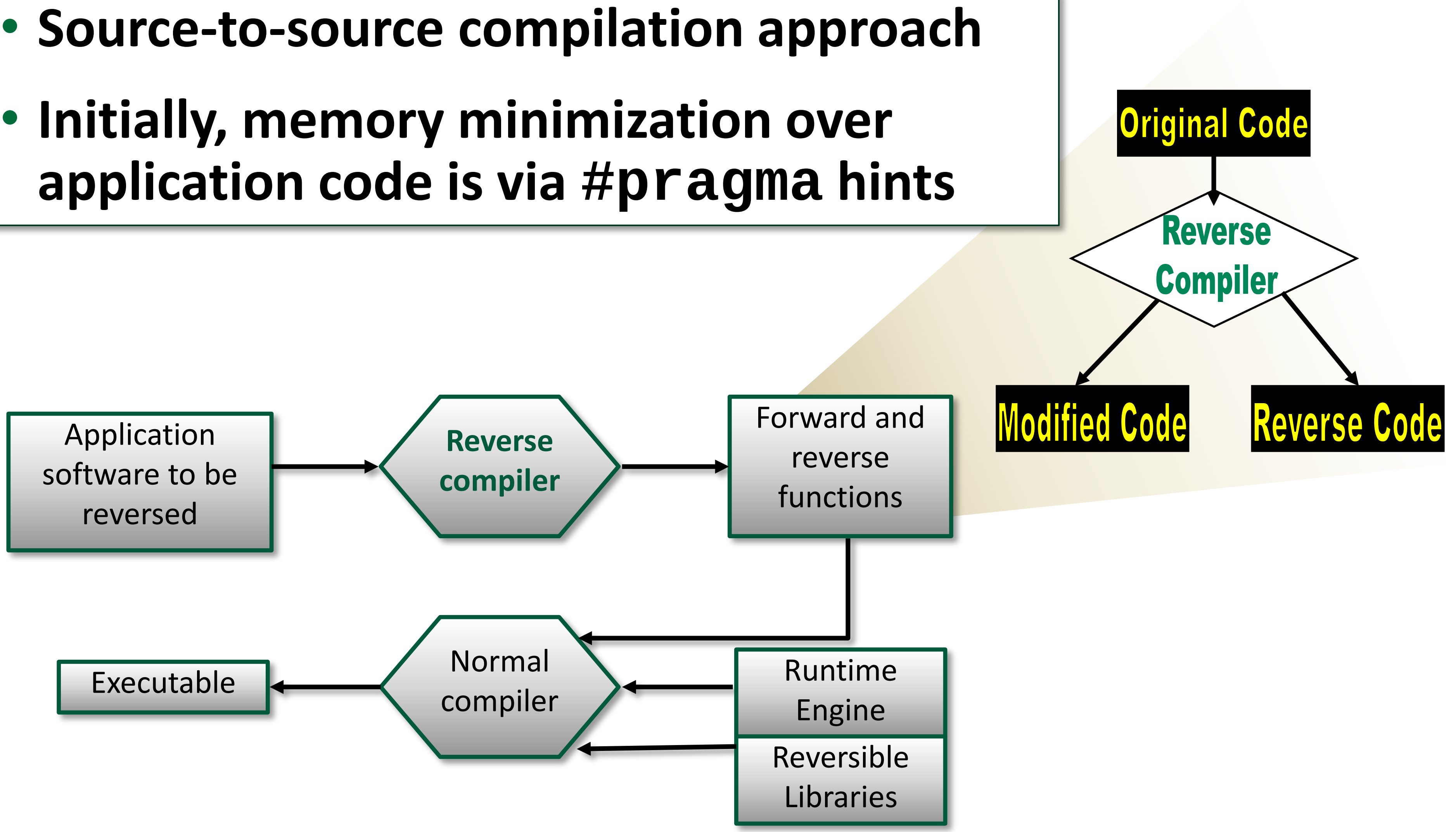


Reversible Software

- **Fundamentally distinct from checkpointing**
- **Theoretically, a strict superset of checkpointing**

Automation (cont.) – Compiler-based

- **Source-to-source compilation approach**
- **Initially, memory minimization over application code is via `#pragma` hints**



Automation (cont.) – Basic Reversal Methodology for Source-to-Source

- Simple generation rules as starting points for reverse source generation, and *upper-bounds* on bit requirements for various statement types
- Much optimization necessary at a higher-level, beyond these rules

Type	Description	Application Code			Bit Requirements		
		Original	Translated	Reverse	Self	Child	Total
T0	simple choice	if() s1	if() {s1; b=1;}	if(b==1){inv(s1);}	1	x1,	1+
		else s2	else {s2; b=0;}	else{inv(s2);}		x2	max(x1,x2)
T1	compound choice	if () s1;	if() {s1; b=1;}	if(b==1) {inv(s1);}	lg(n)	x1,	lg(n) +
	(n-way)	elseif() s2;	elseif() {s2; b=2;}	elseif(b==2) {inv(s2);}		x2,	max(x1….xn)
		elseif() s3;	elseif() {s3; b=3;}	elseif(b==3) {inv(s3);}		….,	
		else() sn;	else {sn; b=n;}	else {inv(sn);}		xn	
T2	fixed iterations (n)	for(n)s;	for(n) s;	for(n) inv(s);	0	x	n*x
T3	variable iterations	while() s;	b=0;	for(b) inv(s);	lg(n)	x	lg(n) +n*x
	(maximum n)		while() {s; b++;}				
T4	function call	foo();	foo();	inv(foo)();	0	x	x
T5	constructive	v@ = w;	v@ = w;	v = @w;	0	0	0
	assignment						
T6	k-byte destructive	v = w;	{b =v; v = w;}	v = b;	8k	0	8k
	assignment						
T7	sequence	s1;	s1;	inv(sn);	0	x1+	x1+…+xn
		s2;	s2;	inv(s2);		….+	
		sn;	sn;	inv(s1);		xn	
T8	Nesting of T0-T7	Recursively apply the above			Recursively apply the above		

Automation (cont.) – Model-based Reversal Example

Diffusion Equation

$$\frac{\partial F}{\partial t} = k\frac{\partial^2 F}{\partial x^2} + \alpha$$

Forward

$$a_i^{j+1} = k\Delta t\frac{(a_{i+1}^j + a_{i-1}^j) + (2 - \frac{\Delta x^2}{k\Delta t})a_i^j}{(\Delta x^2)} + \alpha\Delta t$$

Reverse

$$a_i^j = \frac{k\Delta t(a_{i+1}^j + a_{i-1}^j) - (\Delta x^2)a_i^{j+1} + \alpha(\Delta x^2)\Delta t}{(\Delta x^2) - 2k\Delta t}$$

Discretization

$$\frac{a_i^{j+1} - a_i^j}{\Delta t} = k\frac{a_{i+1}^j - 2a_i^j + a_{i-1}^j}{(\Delta x)^2} + \alpha$$

Reversible Execution

- Space discretized into cells
- Each cell *i* at time increment *j* computes a_i^j
- Can go forward & reverse in time
  - Forward code computes a_i^{j+1}
  - Reverse code recovers a_i^j
- Note that $a_{i+1}^{j+1} = a_{i+1}^j$ due to discretization across cells

Automation (cont.) – Linear Codes

Compute n^th^ and n+1^th^ Fibonacci number: f(n)=f(n-1)+f(n-2)

int a = 0, b = 1

Forward

for *i* from *2* to n:
    Invoke f()

```
f()
{
        int c = a
        a = b
        b = b + c
}
```

Reverse

Reverse

for *i* from *n* to 2:
    Invoke $f^{-1}()$

```
f-1()
{
        int c = a
        a = -a + b
        b = c
}
```

i		2	3	4	5	6
a	0	1	1	2	3	5
b	1	1	2	3	5	8
c		0	1	1	2	3

$f^{-1}(f(a,b)) = (a,b)$

$f^{-1}(f^{-1}(f(f(a,b)))) = (a,b)$ ...

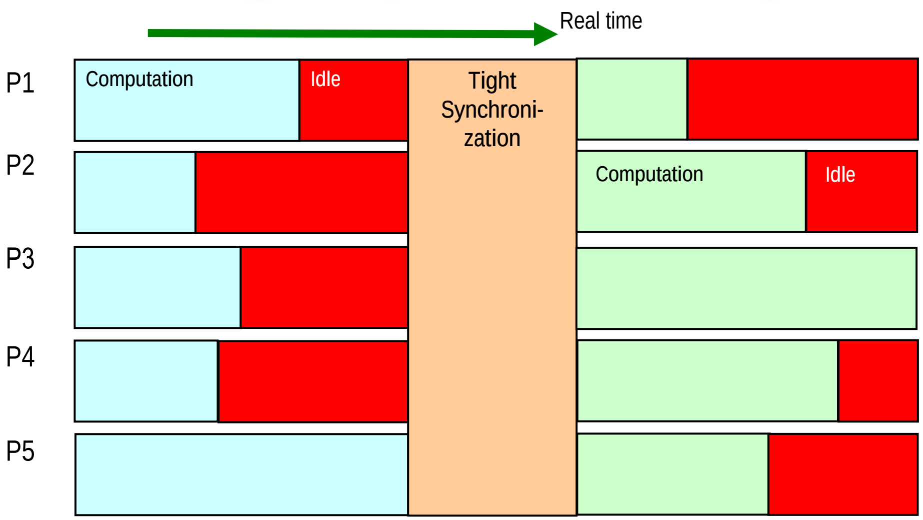
In general, can reverse linear codes, by using single static assignment (SSA), inversion and reduction. *Examples*: Swap, Circular Shift

Automation (cont.) – Libraries

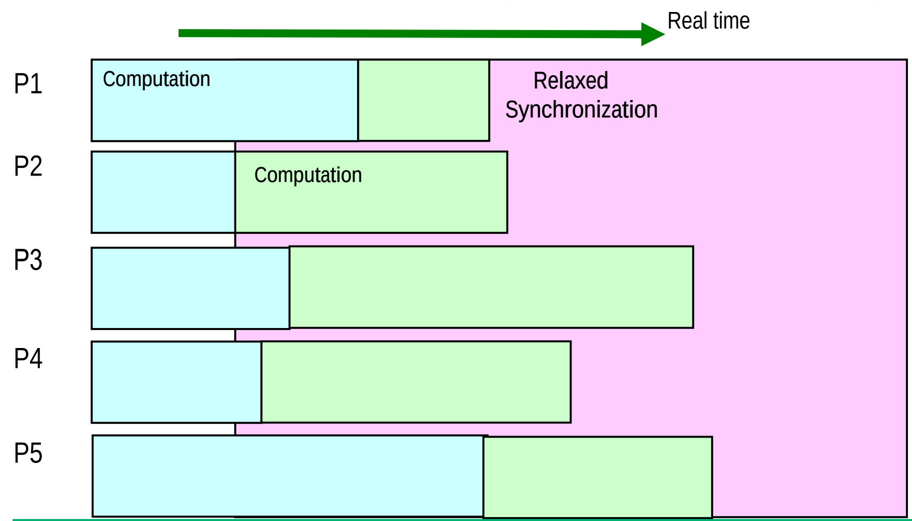
- **Reversible versions of commonly-used libraries**
- **Example 1: Random number generation**
  - Reversible random number generator RRNG (to be released soon) in C, Java, and FORTRAN
  - Large period, multiple independent streams
- **Example 2: Reversible data structures**
  - We are developing container data types with forward and reverse modes
- **Example 3: Reversible linear algebra**
  - We are developing a first cut at Rever-BLAS (Reversible Basic Linear Algebra Services)

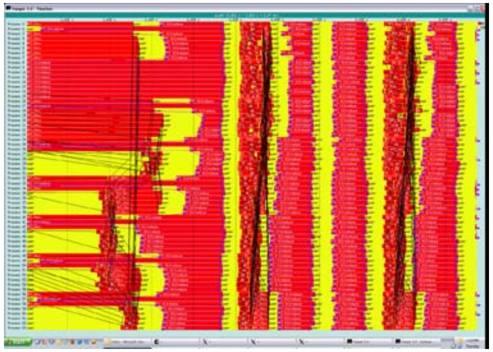
Reversible Runtime: Relaxation of Synchronization

Tight (traditional)



Relaxed (reversible)





E.g., Vampir Trace of
Parallel Ocean Program (POP)

Local and Global Causality

- Causal errors detected and corrected at runtime
- Intra-processor reversal with anti-computation
- Inter-processor reversal with anti-messages
- Runtime performs intermediate message buffering and flow control

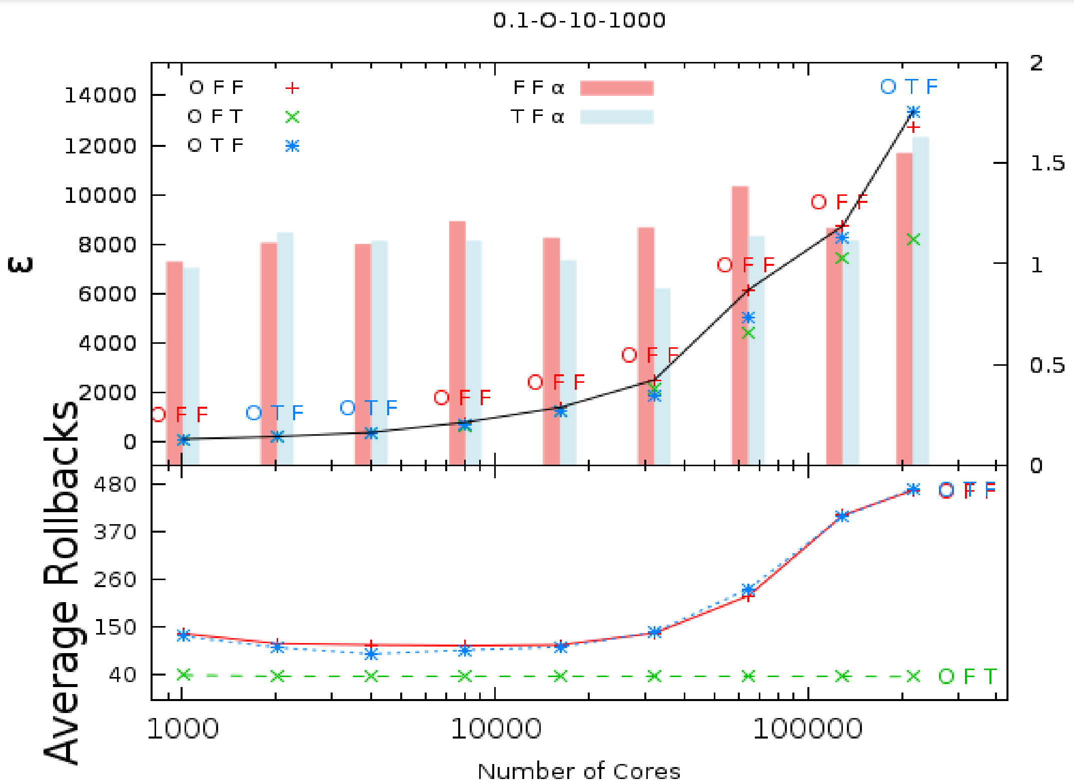
Reversible Runtime (cont.) – Feasibility

- **Relatively fine-grained GUPS-like model**
  - discrete event model, hand-coded reverse code, model size increased with no. of cores (weak scaling), randomized processor neighborhood of ±100
- **Reversible asynchronous execution achieved with high efficiency**

Executed on up to 216K cores of Jaguar (Cray XT5)

ε = millions of events/second

α = factor of improvement over synchronous execution



Reversible Runtime (cont.)

- **Reversible execution of a reaction-diffusion process model**
- **High efficiency achieved at scale, with low-level Portals communication layer; over 2× faster than synchronous mode**

Reaction:

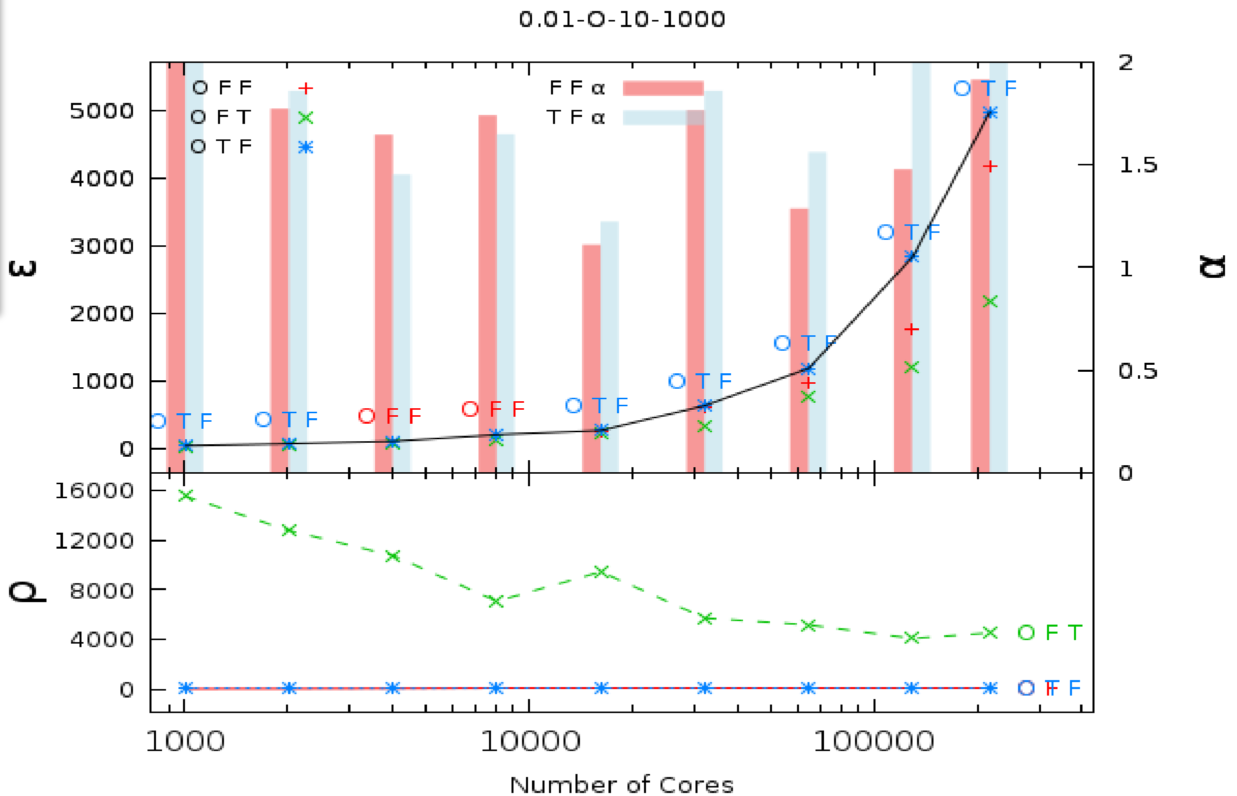
$$p_i = 1 - e^{t \sum_{r \in R} N_r \ln(1 - r s_i \rho)}$$

Diffusion:

Exponentially distributed dwell times, travel times

Executed on up to 216K cores of Jaguar (Cray XT5)

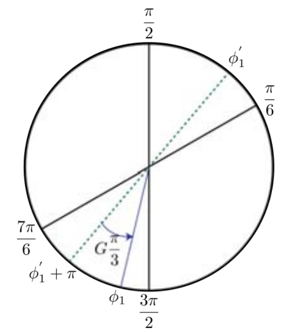
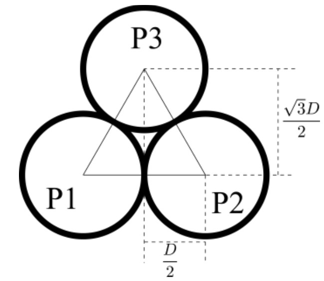
ε = millions of events/second
ρ = average number of rollbacks per core
α = factor of improvement over synchronous mode

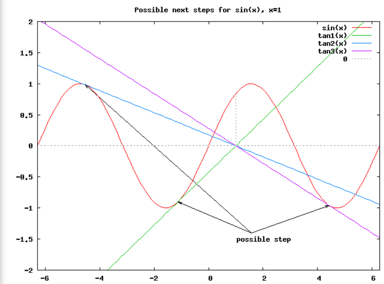


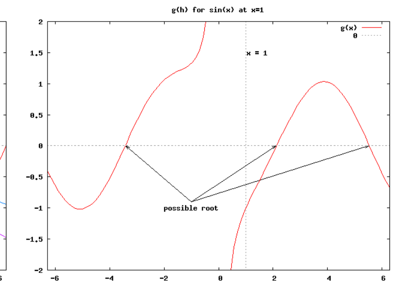
Oak Ridge National Laboratory
Managed by UT-Battelle for the Department of Energy

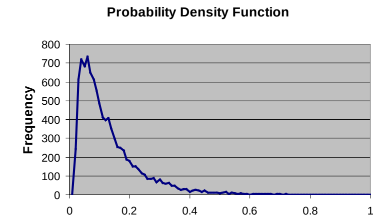
Theory

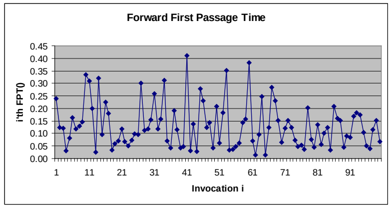
- **New, reversible physical system models**
  - E.g., reversible elastic collisions, phase space coverage
- **Numerical reversal challenges**
  - E.g., reversible transcendental functions
- **Reversible complex probability distributions**
  - E.g., reversible rejection sampling
- **Reversible root-finding**
  - E.g., reversibility of Newton method
- **Other reversible numerical methods**
- **Reversible Turing machines, entropy**

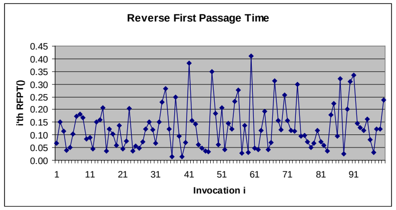







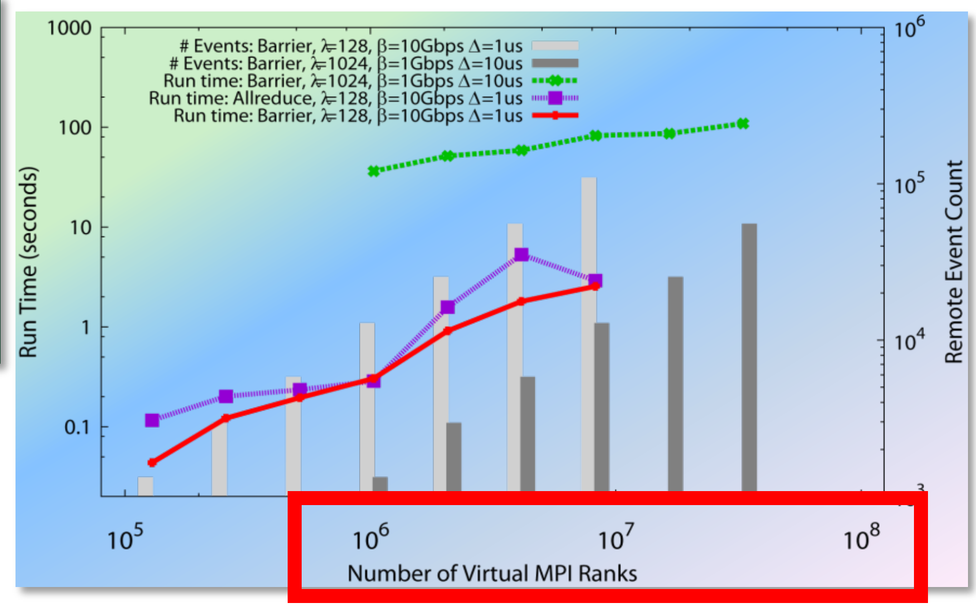




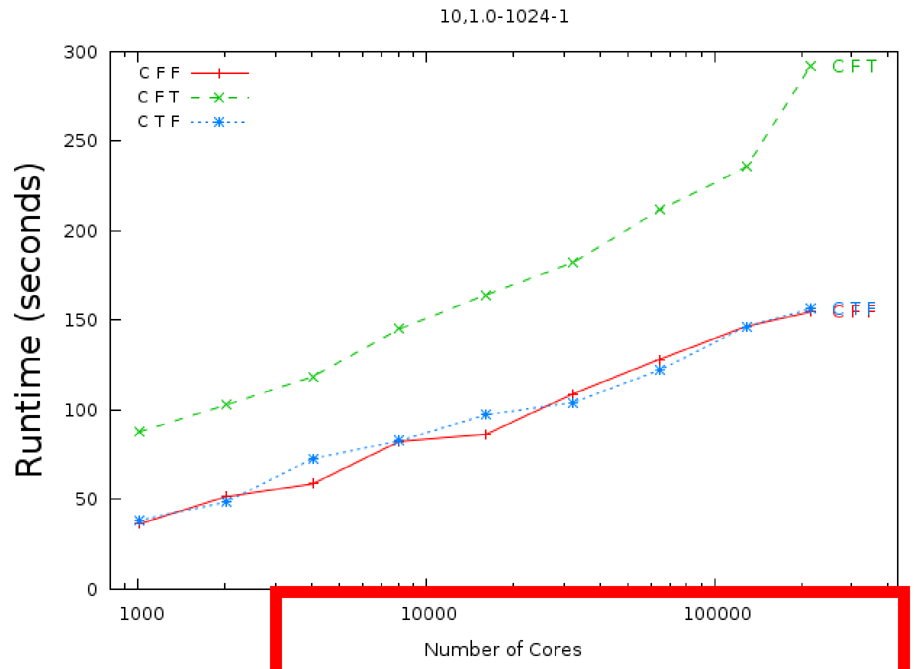


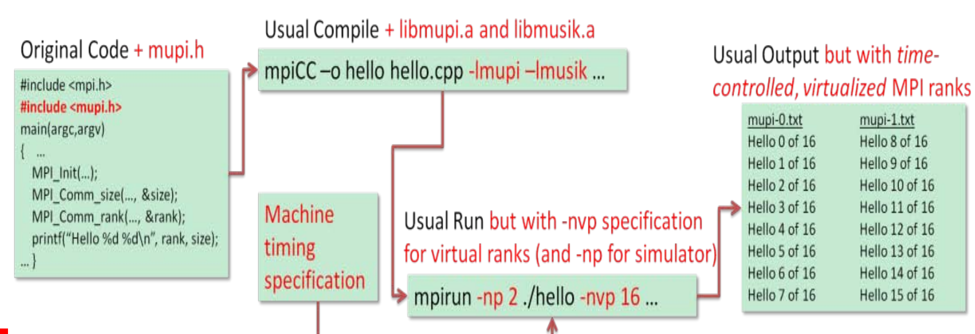
Experimentation – μπ Virtual MPI Testbed 216K Real Cores, 221M Virtual Ranks

- **Facilitates co-design, helps us in experimentation with new runtime frameworks in/over MPI**
- **Several MPI applications tested over μπ**
  - **Tests**: mpiping, allreduce, matmul, pical, …
  - **Benchmarks**: FFTE, GUPS, Gaussian Elimination, …
  - **Applications**: Sweep3D, NWChem, …
  - **DOE Mini-Apps**: pHPCG, CGPOP, …



virtual-barrier, LPX=1024, on Jaguar (Cray XT5)





Experimentation (cont.) – μπ System for Exploring new Million-Rank MPI Runtimes and Applications

What *Challenges*

- **Usable**: Simulate real MPI codes
- **Fast**: Purely discrete event execution
- **Ultra-scale**: 10^6-10^7 ranks
- **Highest fidelity**: Function, timing, feedback

Why *Motivation*

- Exascale co-design
- Prepare software for scaling
- Debug, test, fine-tune frameworks, codes
- Experiment with user-chosen hardware

μπ

Software-based virtual experimentation system

"Seed Corn" of HPC*

How *Techniques*

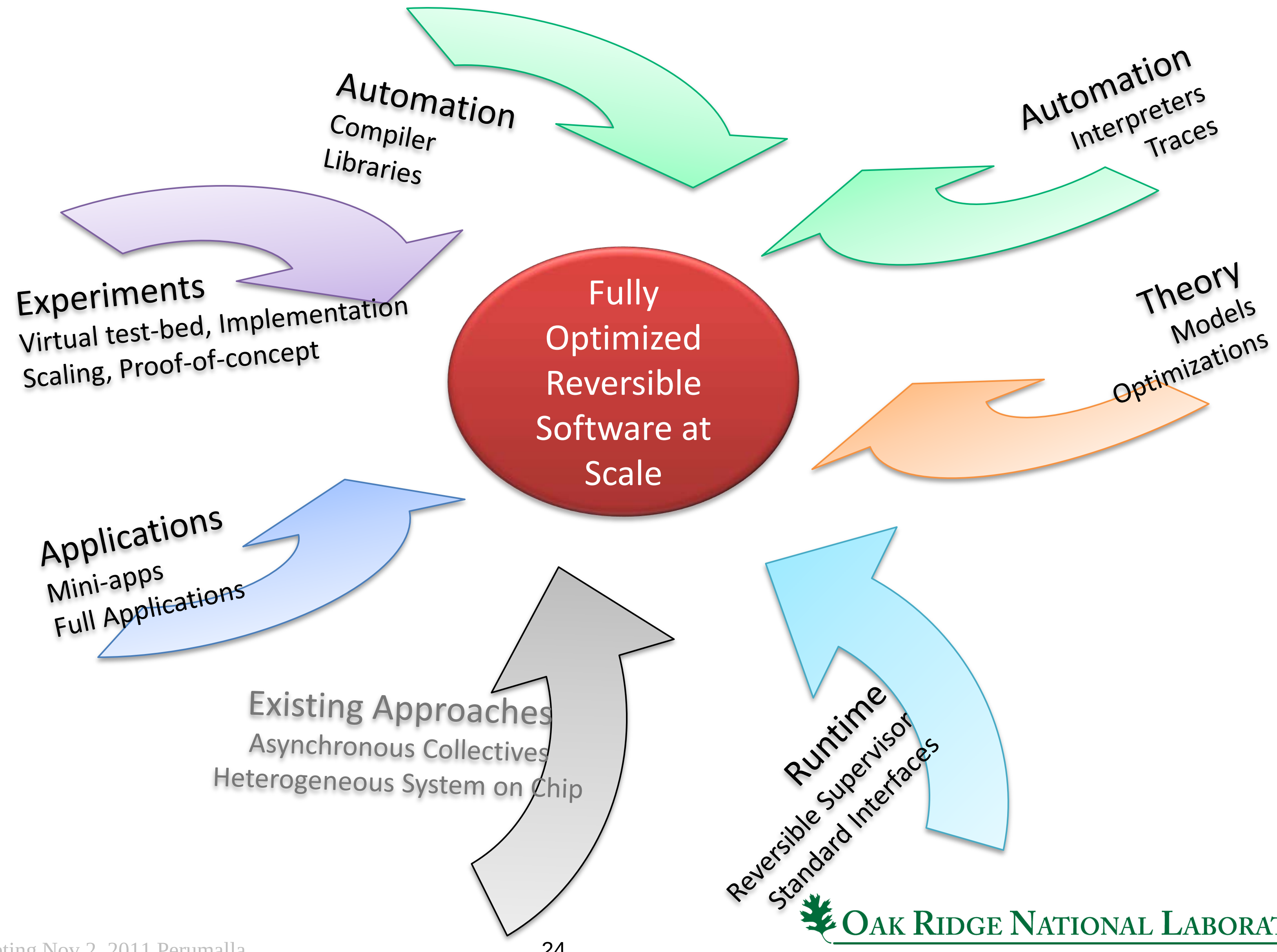
- Multiplex virtual on real ranks
- Optimize virtual MPI layer speed
- Achieve correct timing / ordering
- Expand supported MPI primitives

Etc *Highlights*

- Tested with **several million simulated ranks**
- **1,024 virtual ranks per real rank**
- Tested with multiple MPI benchmarks, apps
- **To be used with DOE mini-apps**

*HPC=High Performance Computing

Vision: Integrated Reversible Software



Outreach – Near-term (Planned)

Software Releases

- Rever-RNG (C, FORTRAN, Java)
- Rever-C (Source-to-Source C Reversal Compiler)
- Rever-BLAS (BLAS extension)
- Rever-Apps (Mini-apps)

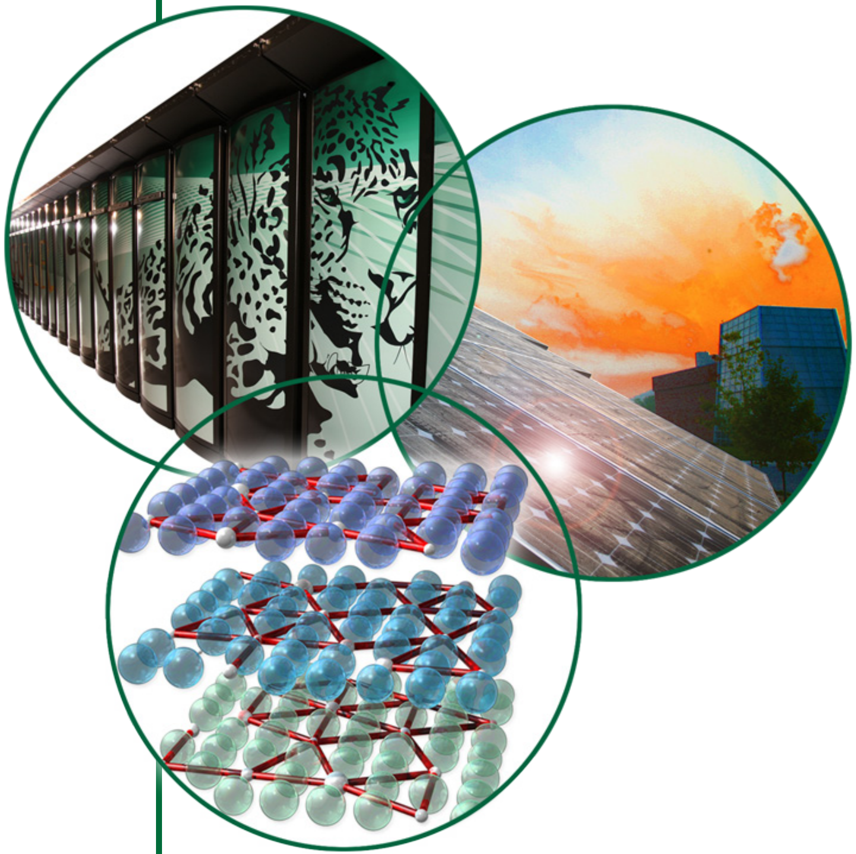
"Rever Challenge Series"

- Competition open to the community
- Easy, medium, hard set of reversal problems
- Reference solutions and implementations to be provided by us at the end of each competition

Summary

- Reversible Software Execution adds an important, orthogonal dimension to high performance computing
- Successful proof-of-concept can jumpstart a new class of research all the way from the top to the bottom layers of HPC
- Technology is in basic research stage
- Challenge is immense but potential payoff for future is high

- Also, positions for future developments
  - Adiabatic Computing, Quantum Computing



Publications

- K. S. Perumalla, A. J. Park and V. Tipparaju, “GVT Algorithms and Discrete Event Execution Dynamics on 129K+ Processor Cores,” International Conference on High Performance Computing, 2011
- K. S. Perumalla and A. J. Park, “Improving Multi-Million Virtual Rank MPI Execution in μπ,“ Intl. Conference on Modeling and Simulation of Computing and Telecommunication Systems, 2011
- S. K. Seal and K. S. Perumalla, “Reversible Parallel Discrete Event Formulation of a TLM-based Radio Signal Propagation Model”, ACM Transactions on Modeling and Computer Simulation, 2011
- K. S. Perumalla and S. K. Seal, “Discrete Event Modeling and Massively Parallel Execution of Epidemic Outbreak Phenomena,” Transactions of the Society for Modeling and Simulation Intl., 2011
- S. K. Seal, K. S. Perumalla and S. P. Hirshman, “Improved Parallelization of the SIESTA Magneto-hydrodynamic Equilibrium Code Using Cyclic Reduction,” DP11 American Physical Society, 2011.
- C. D. Carothers, and K. S. Perumalla, “On Deciding between Conservative and Optimistic Approaches on Massively Parallel Platforms,” Winter Simulation Conference, 2010
- K. S. Perumalla and C. D. Carothers, “Compiler-based Automation Approaches to Reverse Computation,” Workshop on Reverse Computation, 2010
- K. S. Perumalla, “μπ: A Scalable and Transparent System for Simulating MPI Programs,” ICST Intl. Conference on Simulation Tools and Techniques, 2010
- K. S. Perumalla, A. J. Park and V. Tipparaju, “Towards Software-level Virtual Experimentation of MPI-based Million-way Concurrency,” (in preparation)
- K. S. Perumalla and V. A. Protopopescu, “Reversible Simulation of Elastic Collisions,” (in preparation, for Mathematical Modeling in Applied Sciences)

Thank you

Contact

Kalyan S. Perumalla

Senior R&D Staff & Manager, ORNL

Adjunct Professor, Georgia Tech

perumallaks@ornl.gov

www.ornl.gov/~2ip

+1 (865) 241-1315

+1 (865) 576-0003 (fax)

Oak Ridge National Laboratory

PO Box 2008, MS-6085

Oak Ridge, TN 37831-6085