Towards Unlocking Concurrency to the Masses
Second Year PhD Report

Alessandro Pellegrini
High Performance and Dependable Computing Systems Group
DIAG – Sapienza, University of Rome

October 3, 2012
Research Context (1)

• Moore’s law no longer involves an enhancement in computing performances [15]
Research Context (1)

- Moore’s law no longer involves an enhancement in computing performances [15]

- The answer:
  - Parallel Architectures
    - GPUs
    - Multi/many-core architectures

How to bring the power of parallelism to the masses?
Research Context (1)

- Moore’s law no longer involves an enhancement in computing performances [15]

- The answer:
  - Parallel Architectures
    - GPUs
    - Multi/many-core architectures
  - Parallel Programming Paradigm
Research Context (1)

- Moore’s law no longer involves an enhancement in computing performances [15]

- The answer:
  - Parallel Architectures
    - GPUs
    - Multi/many-core architectures
  - Parallel Programming Paradigm

- This solution is still a niche one
Research Context (1)

- Moore’s law no longer involves an enhancement in computing performances [15]

- The answer:
  - Parallel Architectures
    - GPUs
    - Multi/many-core architectures
  - Parallel Programming Paradigm

- This solution is still a niche one

How to bring the power of parallelism to the masses?
Research Context (2)

- Addressing every aspect of parallel/concurrent programming is non-trivial
Research Context (2)

- Addressing every aspect of parallel/concurrent programming is non-trivial

- So far, I have concentrated on special cases:
  - Event-Driven Programming Paradigm
    - The advancement of the execution is determined by the flow of timestamped events which produce changes in the state
  - Discrete Event Simulation Environments
Research Context (2)

- Addressing every aspect of parallel/concurrent programming is non-trivial

- So far, I have concentrated on special cases:
  - **Event-Driven Programming Paradigm**
    - The advancement of the execution is determined by the flow of timestamped events which produce changes in the state
  - **Discrete Event Simulation Environments**
  - **Software Transactional Memories**:
    - Allow a correct sequential object to be mapped into a correct concurrent object
    - Based on the notion of transactions
Goals (1)

- **Performance**
  - Explore new synchronization patterns and protocols
  - Specifically rely on non-blocking algorithms

- **Transparency**
  - Allow the programmer to easily produce a program which is then run as efficiently as possible
  - Need to rely on the most restricted set of new APIs
  - Rely on classical/standard programming models
Goals (1)

- **Performance**
  - Explore new synchronization patterns and protocols
  - Specifically rely on non-blocking algorithms

- **Transparency**
  - Allow the programmer to easily produce a program which is then run as efficiently as possible
  - Need to rely on the most restricted set of new APIs
  - Rely on classical/standard programming models

- **Tools**
  - Practical tools to help the unexperienced

- **Methodologies**
  - General approaches to efficiently support parallel execution
Related Work (1)

- Non-blocking Algorithms
  - Several data structures have been proposed [16, 17, 18, 19]
  - A particular focus is on queues and deques
Related Work (1)

• Non-blocking Algorithms
  ○ Several data structures have been proposed [16, 17, 18, 19]
  ○ A particular focus is on queues and deques

  ○ Concrete applications
    • Mutual exclusion problem [20, 21, 22]
    • Write barriers in garbage collectors [23]
    • Composite locks [24]
Related Work (1)

- Non-blocking Algorithms
  - Several data structures have been proposed [16, 17, 18, 19]
  - A particular focus is on queues and deques
  - Concrete applications
    - Mutual exclusion problem [20, 21, 22]
    - Write barriers in garbage collectors [23]
    - Composite locks [24]

- They have been proven to be an effective and viable approach
Related Work (2)

- Virtual Time Synchronization [33]
  - A set of rules specifying correctness for concurrent execution of Event-Based simulation models
  - Some implementations rely on global data structures, or special-purpose threads (e.g., [34])
  - Either conservative synchronization, or optimistic synchronization [35] protocols/runtime environments have been proposed
Related Work (2)

- **Virtual Time Synchronization** [33]
  - A set of rules specifying correctness for concurrent execution of Event-Based simulation models
  - Some implementations rely on global data structures, or special-purpose threads (e.g., [34])
  - Either *conservative* synchronization, or *optimistic* synchronization [35] protocols/runtime environments have been proposed
  - Efficient memory management in the optimistic case has been supported in several ways
    - Full State Saving [36, 37, 38, 39]
    - Incremental State Saving [40, 41, 42]
    - Mixture of the two [43, 44]
Where are we now?!

• I have moved on two main tracks:
  ○ Event-Driven Programming (Optimistic Simulation flavour):
    • Supports for transparents management of private and shared simulation state
    • Performance enhancements transparently introduced, relying on the autonomic computing paradigm [45, 46, 47]
  ○ Transactional Memories
    • Performance optimization by reducing the wasted work, still transparently!
Where are we now?!

- I have moved on two main tracks:
  - Event-Driven Programming (Optimistic Simulation flavour):
    - Supports for transparents management of private and shared simulation state
    - Performance enhancements transparently introduced, relying on the autonomic computing paradigm [45, 46, 47]
  - Transactional Memories
    - Performance optimization by reducing the wasted work, still transparently!

- Common ground
  - Static instrumentation methodologies/tool to reshuffle the code
Instrumenting Tool (1)

- Static rule-based instrumentation tool
- Lightweight modification of the actual executable
- Wide applicability
Instrumenting Tool (1)

- Static rule-based instrumentation tool
- Lightweight modification of the actual executable
- Wide applicability
Instrumenting Tool (2)

- Possible application scenarios:
  - Profiling
  - Performance Enhancements
  - Synchronization Transparency
  - Post-Mortem Debugging [48]
Private Data Management

- Based on static instrumentation + dynamic reconstruction of memory update targets
- Efficient
  - Recycling of cached disassembly information injected in the executable
  - Memory-update detection’s cost is $O(1)$
- Standard malloc services are wrapped, for transparency
- Fast reconstruction of the state using bit-wise masking of unimportant memory areas
- Wise usage of memory resources
- Several layers involved: compilation, linking and runtime execution
- Evaluated on a complex wireless network simulation model
Private Data Management (2)
Autonomic Approach based on Dual Coding

- First proposal in literature on this topic in this context
- Two versions of the same executable, differently instrumented coexist
- Switch amongst the modes involves reassigning function pointers
- Based on an analytical integral model, which accounts for stability regardless of perturbations and fluctuations
Autonomic Approach based on Dual Coding (2)
Autonomic Approach based on Dual Coding (2)

Execution Speed vs GVT Advancement

Overall Execution Speed

Superlinear speedup wrt the serial execution
Shared Data Management

\[ \forall i, j \ i \neq j \ : \ S_i \cap S_j = \emptyset \]

\[ S = \bigcup_{i=1}^{numLP} S_i \]
Shared Data Management

\[ \forall i, j \ i \neq j \ : \ S_i \cap S_j = \emptyset \quad S = \bigcup_{i=1}^{numLP} S_i \]

- Disjoint States: Message Passing to represent interactions
Shared Data Management

\[
\forall i, j \ i \neq j : S_i \cap S_j = \emptyset
\]

\[
S = \bigcup_{i=1}^{numLP} S_i
\]

- Disjoint States: Message Passing to represent interactions
Shared Data Management

\[ \forall i, j \neq j : S_i \cap S_j = \emptyset \]

\[ S = \bigcup_{i=1}^{numLP} S_i \]

- Disjoint States: Message Passing to represent interactions
- Relaxing this constraint can result in a more flexible paradigm
Shared Data Management

\[ \forall i, j \neq j : S_i \cap S_j = \emptyset \]

\[ S = \bigcup_{i=1}^{numLP} S_i \]

- Disjoint States: Message Passing to represent interactions
- Relaxing this constraint can result in a more flexible paradigm

**Goal:**

- Enable the application programmer to access both the object’s private state and the global portion
Shared Data Management

- Implement variables as multi-versioned lists
- Use non blocking algorithms for synchronization
- Remap shared data to shared memory
Shared Data Management

- Implement variables as multi-versioned lists
- Use non-blocking algorithms for synchronization
- Remap shared data to shared memory
- Efficient new rollback scheme: waste as minimum as possible
Shared Data Management (2)
Partial Rollback in STMs

- Allow an aborting transaction to save as much work as possible
- Rely on snapshot extension
- Changed relation between transactions and their snapshots
  - What a transaction sees might dynamically change
- Real implementation within TinySTM
Partial Rollback in STMs

- Allow an aborting transaction to save as much work as possible
- Rely on snapshot extension
- Changed relation between transactions and their snapshots
  - What a transaction sees might dynamically change
- Real implementation within TinySTM
Future Work

• Interaction Transparency
  ○ What if a parallel program wants to interact with external worlds, which are not necessarily parallel?
  ○ Input/Output problem when relying on speculative approaches
Future Work

• Interaction Transparency
  ○ What if a parallel program wants to interact with external worlds, which are not necessarily parallel?
  ○ Input/Output problem when relying on speculative approaches

• Deployment Transparency
  ○ Study how to transparently select the best amount of concurrent resources to avoid thrashing
  ○ Relevant as well in the Cloud Computing field: possible waste of money as well
Future Work

• Interaction Transparency
  ○ What if a parallel program wants to interact with external worlds, which are not necessarily parallel?
  ○ Input/Output problem when relying on speculative approaches

• Deployment Transparency
  ○ Study how to transparently select the best amount of concurrent resources to avoid thrashing
  ○ Relevant as well in the Cloud Computing field: possible waste of money as well

• Programming Model Transparency
  ○ What if the programmer knows about parallelization?
  ○ How to mix induced parallelism with explicit parallelism?
  ○ This is where all my proposals integrate
List of Publications

A load sharing architecture for optimistic simulations on multi-core machines.  
To Appear.

Assessing load sharing within optimistic simulation platforms (invited paper).  
To Appear.

Transparent and efficient shared-state management for optimistic simulations on multi-core machines.  

Towards symmetric multi-threaded optimistic simulation kernels.  

Cache-aware memory manager for optimistic simulations.  
Winner of the Best Paper Award.
List of Publications (2)


List of Pending/In Preparation Publications

A symmetric multi-threaded architecture for load-sharing in multi-core optimistic simulations.  
ACM Performance Evaluation Review.  
Fast Track invitation as InfQ 2012 Selected Paper. In Preparation.

Autonomic state management for optimistic simulation platforms.  
IEEE Transactions on Parallel and Distributed Systems.  
In Preparation.

Multi-threaded simulation of 4G cellular systems within the LTE-Sim framework.  
Under Review.

Efficient partial rollback in software transactional memories.  
In Preparation for Submission to the 22nd Conference on Compiler Construction, 2013.
References

The free lunch is over: A fundamental turn toward concurrency in software.

[16] Alex Kogan and Erez Petrank.
Wait-free queues with multiple enqueuers and dequeuers.

Simple, fast, and practical non-blocking and blocking concurrent queue algorithms.

[18] Maurice P. Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer, III.
Software transactional memory for dynamic-sized data structures.

A pragmatic implementation of non-blocking linked-lists.

A new fast-path mechanism for mutual exclusion.
References (2)

A fast mutual exclusion algorithm.

A fast, scalable mutual exclusion algorithm.

[23] Yossi Levanoni and Erez Petrank.
An on-the-fly reference-counting garbage collector for java.

The Art of Multiprocessor Programming.
Morgan Kaufmann, March 2008.

Self-adjusting computation: (an overview).

An experimental analysis of self-adjusting computation.
References (3)


References (4)


References (5)


References (6)

An analytical model for hybrid checkpointing in Time Warp distributed simulation.  

[45] Shenin Hassan, Dhiya Al-Jumeily, and Abir Jaafar Hussain.  
Autonomic computing paradigm to support system’s development.  

[46] Paul Horn.  
Autonomic computing: IBM’s perspective on the state of information technology.  

The vision of autonomic computing.  

[48] David Pacheco.  
Postmortem debugging in dynamic environments.  
Thanks for your attention

Questions?

http://www.dis.uniroma1.it/~pellegrini
pellegrini@dis.uniroma1.it