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

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 stateb
    - *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 stateb
    - *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* syncronization [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* syncronization [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!

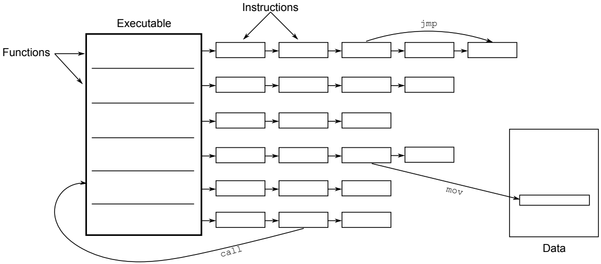
- Common ground
  - Static instrumentation methodologies/tool to reshuffle the code

Intrumenting Tool (1)

- Static rule-based instrumentation tool
- Lightweight modification of the actual executable
- Wide applicability

Intrumenting Tool (1)

- Static rule-based instrumentation tool
- Lightweight modification of the actual executable
- Wide applicability



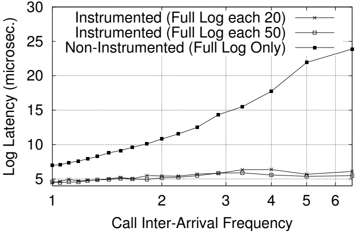
Instrumenting Tool (2)

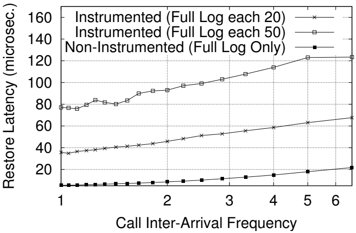
- Possible application scenarios:
  - Profiling
  - Performance Enhancements
  - Synchronization Transparency
  - Post-Mortem Debugging [48]

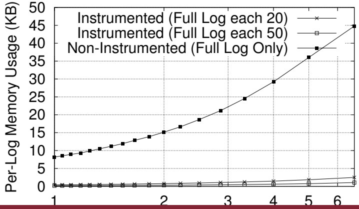
Private Data Management

- 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)

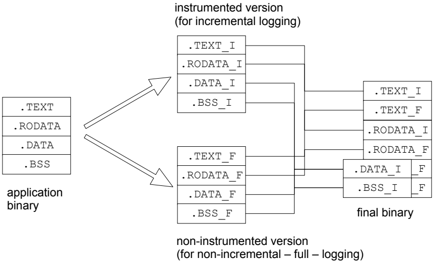




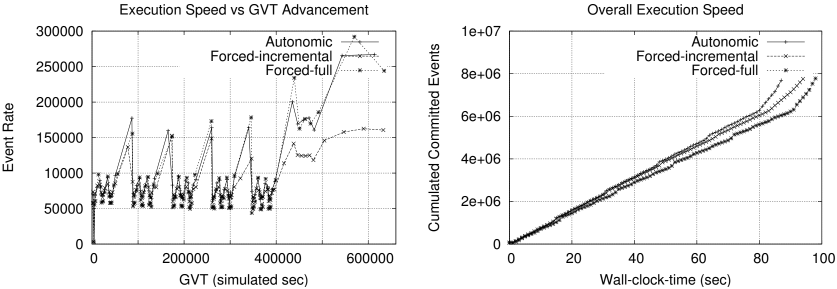


Autonomic Approach based on Dual Coding

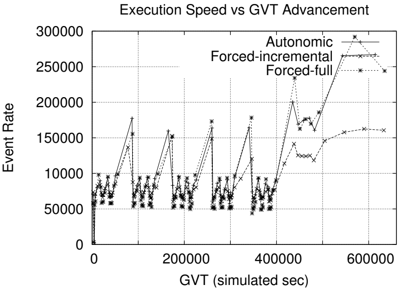
- First proposal in literature on this topic in this context
- Two versions of the same executable, differently instrumented coexist
- Switch amongst the modes involves reassigning function pointers
- Based on an analytical integral model, which accounts for stability regardless of perturbations and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

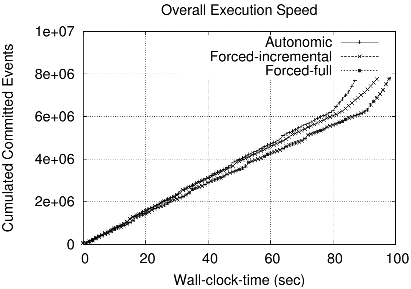


Autonomic Approach based on Dual Coding (2)



Autonomic Approach based on Dual Coding (2)





Superlinear speedup wrt the serial execution

Shared Data Management

$$\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset \qquad S = \bigcup_{i=1}^{numLP} S_i$$

Shared Data Management

$$\forall i, j \ i \neq j \ : \ S_i \cap S_j = \emptyset \qquad S = \bigcup_{i=1}^{numLP} S_i$$

- Disjoint States: Message Passing to represent interactions

Shared Data Management

~~$\forall i, j\ i \neq j\ :\ S_i \cap S_j = \emptyset$~~ $\qquad S = \bigcup_{i=1}^{numLP} S_i$

- 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
- Relaxing this constraint can result in a more flexible paradigm

Shared Data Management

~~$\forall i, j \; i \neq j \; : \; S_i \cap S_j = \emptyset$~~ $\qquad S = \bigcup_{i=1}^{numLP} S_i$

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

Goal:

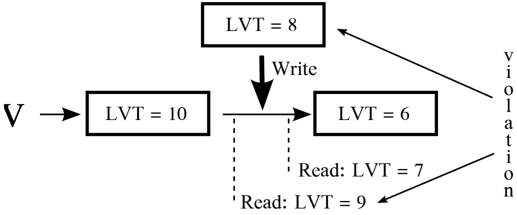
- Enable the application programmer to access both the object's private state and the global portion

Shared Data Management

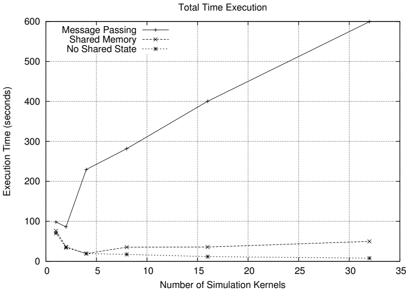
- Implement variables as multi-versioned lists
- Use non blocking algorithms for synchronization
- Remap shared data to shared memory

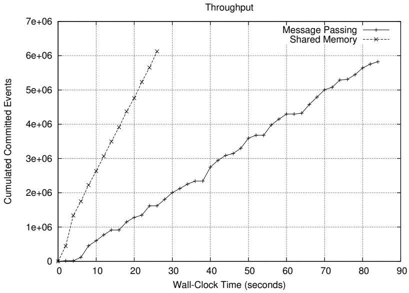
Shared Data Management

- 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)



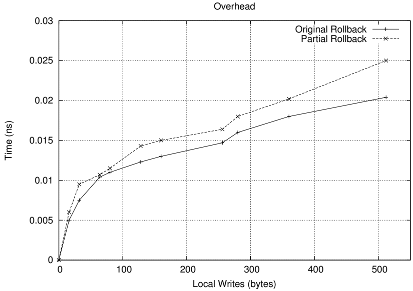


Partial Rollback in STMs

- 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

[1] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia.
A load sharing architecture for optimistic simulations on multi-core machines.
In Proceedings of the 19th International Conference on High Performance Computing, HiPC. IEEE Computer Society, December 2012.
To Appear.

[2] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia.
Assessing load sharing within optimistic simulation platforms (invited paper).
In Proceedings of the 2012 Winter Simulation Conference, WSC. Society for Computer Simulation, December 2012.
To Appear.

[3] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia.
Transparent and efficient shared-state management for optimistic simulations on multi-core machines.
In Proceedings 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS, pages 134–141. IEEE Computer Society, August 2012.

[4] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia.
Towards symmetric multi-threaded optimistic simulation kernels.
In Proceedings of the 26th International Workshop on Principles of Advanced and Distributed Simulation, PADS, pages 211–220. IEEE Computer Society, August 2012.

[5] Roberto Vitali, Alessandro Pellegrini, and Gionata Cerasuolo.
Cache-aware memory manager for optimistic simulations.
In Proceedings of the 5th International ICST Conference of Simulation Tools and Techniques, SIMUTools, March 2012.
Winner of the Best Paper Award.

List of Publications (2)

[6] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia.
The ROme OpTimistic Simulator: Core internals and programming model.
In Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, SIMUTools. ICST, 2011.

[7] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia.
An evolutionary algorithm to optimize log/restore operations within optimistic simulation platforms.
In Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, SIMUTools. SIGSIM, 2011.

[8] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia.
Autonomic log/restore for advanced optimistic simulation systems.
In Proceedings of the Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS, pages 319–327. IEEE Computer Society, 2010.

[9] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia.
Benchmarking memory management capabilities within root-sim.
In Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications. IEEE Computer Society, 2009.

[10] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia.
Di-DyMeLoR: Logging only dirty chunks for efficient management of dynamic memory based optimistic simulation objects.
In Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation, PADS, pages 45–53. IEEE Computer Society, 2009.
Candidate for (but not winner of) the Best Paper Award.

List of Pending/In Preparation Publications

[11] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia.
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.

[12] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia.
Autonomic state management for optimistic simulation platforms.
IEEE Transactions on Parallel and Distributed Systems.
In Preparation.

[13] Alessandro Pellegrini and Giuseppe Piro.
Multi-threaded simulation of 4G cellular systems within the LTE-Sim framework.
In Proceedings of the 8th IEEE International Workshop on the Performance Analysis and Enhancement of Wireless Networks, PAEWN. IEEE Computer Society, March 2013.
Under Review.

[14] Alice Porfirio, Alessandro Pellegrini, Pierangelo Di Sanzo, and Francesco Quaglia.
Efficient partial rollback in software transactional memories.
In Preparation for Submission to the 22nd Conference on Compiler Construction, 2013.

References

[15] Herb Sutter.
The free lunch is over: A fundamental turn toward concurrency in software.
Dr. Dobb's Journal, 30(3):202–210, 2005.

[16] Alex Kogan and Erez Petrank.
Wait-free queues with multiple enqueuers and dequeuers.
In Proceedings of the 16th ACM symposium on Principles and practice of parallel programming, PPoPP, pages 223–234. ACM, 2011.

[17] Maged M. Michael and Michael L. Scott.
Simple, fast, and practical non-blocking and blocking concurrent queue algorithms.
In Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, PODC, pages 267–275. ACM, 1996.

[18] Maurice P. Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer, III.
Software transactional memory for dynamic-sized data structures.
In Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing, pages 92–101. ACM, 2003.

[19] Timothy Harris.
A pragmatic implementation of non-blocking linked-lists.
In Jennifer Welch, editor, Distributed Computing, volume 2180, pages 300–314. Springer Berlin / Heidelberg, 2001.

[20] James H. Anderson and Yong-Jik Kim.
A new fast-path mechanism for mutual exclusion.
Distrib. Comput., 14(1):17–29, January 2001.

References (2)

[21] Leslie Lamport.
A fast mutual exclusion algorithm.
ACM Trans. Comput. Syst., 5(1):1–11, January 1987.

[22] Jae-heon Yang and James H. Anderson.
A fast, scalable mutual exclusion algorithm.
Distributed Computing, 9:9–1, 1994.

[23] Yossi Levanoni and Erez Petrank.
An on-the-fly reference-counting garbage collector for java.
ACM Trans. Program. Lang. Syst., 28(1):1–69, January 2006.

[24] Maurice Herlihy and Nir Shavit.
The Art of Multiprocessor Programming.
Morgan Kaufmann, March 2008.

[25] Umut A. Acar.
Self-adjusting computation: (an overview).
In Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation, PEPM, pages 1–6. ACM, 2009.

[26] Umut A. Acar, Guy E. Blelloch, Matthias Blume, and Kanat Tangwongsan.
An experimental analysis of self-adjusting computation.
SIGPLAN Not., 41(6):96–107, June 2006.

References (3)

[27] Umut A. Acar, Guy Blelloch, Ruy Ley-Wild, Kanat Tangwongsan, and Duru Turkoglu.
Traceable data types for self-adjusting computation.
SIGPLAN Not., 45(6):483–496, June 2010.

[28] Matthew A. Hammer, Umut A. Acar, and Yan Chen.
CEAL: a C-based language for self-adjusting computation.
In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI, 2009.

[29] Thorsten Grotker.
System Design with SystemC.
Kluwer Academic Publishers, 2002.

[30] P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier.
Signal–a data flow-oriented language for signal processing.
Acoustics, Speech and Signal Processing, IEEE Transactions on, 34(2):362–374, apr 1986.

[31] B.A. Myers, R.G. McDaniel, R.C. Miller, A.S. Ferrency, A. Faulring, B.D. Kyle, A. Mickish, A. Klimovitski, and P. Doane.
The amulet environment: new models for effective user interface software development.
Software Engineering, IEEE Transactions on, 23(6):347–365, June 1997.

[32] Camil Demetrescu, Irene Finocchi, and Andrea Ribichini.
Reactive imperative programming with dataflow constraints.
In Proceedings of the Object-Oriented Programming, Systems, Languages & Applications Conference, OOPSLA, pages 407–426. ACM, October 2011.

References (4)

[33] Richard M. Fujimoto.
Parallel discrete event simulation.
Communications of the ACM, 33(10):30–53, October 1990.

[34] Dale E. Martin, Timothy J. McBrayer, and Philip A. Wilsey.
WARPED: A time warp simulation kernel for analysis and application development.
In HICSS '96: Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96) Volume 1: Software Technology and Architecture, page 383. IEEE Computer Society, 1996.

[35] David R. Jefferson.
Virtual Time.
ACM Transactions on Programming Languages and System, 7(3):404–425, July 1985.

[36] Josef Fleischmann and Philip A. Wilsey.
Comparative analysis of periodic state saving techniques in time warp simulators.
In Proceedings of the 9th Workshop on Parallel and Distributed Simulation, pages 50–58. IEEE Computer Society, June 1995.

[37] Bruno R. Preiss, Wayne M. Loucks, and D. MacIntyre.
Effects of the checkpoint interval on time and space in Time Warp.
ACM Transactions on Modeling and Computer Simulation, 4(3):223–253, July 1994.

[38] Francesco Quaglia and Andrea Santoro.
Non-blocking checkpointing for optimistic parallel simulation: Description and an implementation.
IEEE Transactions on Parallel and Distributed Systems, 14(6):593–610, june 2003.

References (5)

[39] Robert Rönngren and Rassul Ayani.
Adaptive checkpointing in Time Warp.
In Proceedings of the 8th Workshop on Parallel and Distributed Simulation, pages 110–117. Society for Computer Simulation, July 1994.

[40] Robert Rönngren, M. Liljenstam, Rassul Ayani, and J. Montagnat.
Transparent incremental state saving in Time Warp parallel discrete event simulation.
In Proceedings of the 10th Workshop on Parallel and Distributed Simulation, pages 70–77. IEEE Computer Society, May 1996.

[41] Jeffrey S. Steinman.
Incremental state saving in SPEEDES using C plus plus.
In Proceedings of the Winter Simulation Conference, pages 687–696. Society for Computer Simulation, december 1993.

[42] Darrin West and Kiran Panesar.
Automatic incremental state saving.
In Proceedings of the 10th Workshop on Parallel and Distributed Simulation, pages 78–85. IEEE Computer Society, May 1996.

[43] Steve Franks, Fabian Gomes, Brian Unger, and John Cleary.
State saving for interactive optimistic simulation.
In Proceedings of the 11th workshop on Parallel and Distributed Simulation, pages 72–79. IEEE Computer Society, 1997.

References (6)

[44] H.M. Soliman and A.S. Elmaghraby.
An analytical model for hybrid checkpointing in Time Warp distributed simulation.
IEEE Transactions on Parallel and Distributed Systems, 9(10):947–951, october 1998.

[45] Shenin Hassan, Dhiya Al-Jumeily, and Abir Jaafar Hussain.
Autonomic computing paradigm to support system's development.
In Proceedings of the 2nd International Conference on Developments in eSystems Engineering, DESE, pages 273–278. IEEE Computer Society, December 2009.

[46] Paul Horn.
Autonomic computing: IBM's perspective on the state of information technology.
15:1–39, 2001.

[47] J.O. Kephart and D.M. Chess.
The vision of autonomic computing.
Computer, 36(1):41 – 50, January 2003.

[48] David Pacheco.
Postmortem debugging in dynamic environments.
Communications of the ACM, 54(12):44–51, December 2011.

Thanks for your attention

Questions?

http://www.dis.uniroma1.it/∼pellegrini

pellegrini@dis.uniroma1.it