Time Warp on the Go

Gabriele D'Angelo

<gda@cs.unibo.it>

http://www.cs.unibo.it/gdangelo/

joint work with:

Stefano Ferretti** and **Moreno Marzolla

Desenzano, Italy

DIstributed **SI**mulation & **O**nline gaming (**DISIO**), 2012

Presentation **outline**

- A little **background** on simulation
- **P**arallel **A**nd **D**istributed **S**imulation (**PADS**)
- Synchronization: the **Time Warp** mechanism
- The **Go** programming language
- The **Go-Warp** simulator
- Performance evaluation: the **PHOLD** benchmark
- **Conclusions**

Starting from scratch: **simulation**

- **"A computer simulation is a computation that models the behavior of some real or imagined system over time"** **(R.M. Fujimoto)**
- **Motivations:**
  - performance evaluation
  - study of new solutions
  - creation of virtual worlds such as online games and digital virtual environments
  - …

Discrete **E**vent **S**imulation (**DES**)

- The **state** of the simulated system is represented through a **set of variables**
- The key concept is the "**event**"
- An event is a **change in the system state** and it **occurs at an instant in time**
- The evolution is given by a **chronological sequence of events**
- All is done through the **creation**, **delivery** and **computation** of events

DES on a **single CPU**: **sequential simulation**

- All such tasks are accomplished by a **single execution unit** (that is a **CPU** and some RAM)

- **PROS**: it is a **very simple approach**

- **CONS**: there are a few **significant limitations**
  - the **time** required to complete the simulation run
  - if the model is complex the **RAM could be not enough**

- **This approach does not scale!**

Going **Parallel**: **PDES**

Parallel **D**iscrete **E**vent **S**imulation (**PDES**)

- **Multiple interconnected** execution units (**CPU**s or **hosts**)
- Each unit manages **a part of the simulation model**
- Each execution unit has to manage a **local event list**
- **Locally generated events** may have to be **delivered to remote execution units**
- All of this needs to be carefully **synchronized**
- "**Concurrent events**" can be **executed in parallel**, this can lead to a significant **speedup of the execution**

Parallel **A**nd **D**istributed **S**imulation (**PADS**)

- **"Any simulation in which more than one processor is employed"** **(K.S. Perumalla)**
- This is a very simple and general definition, there are many different "flavors" of PADS
- A lot of **good reasons for going PADS**:
  - **scalability**
  - **performance** (obtaining the results faster)
  - to model **larger** and **more complex** scenarios
  - **interoperability**, to integrate commercial off-the-shelf simulators
  - **composability** of different simulation models
  - to integrate simulators that are **geographically distributed**
  - **Intellectual Property** (IP) protection
  - …

Parallel **A**nd **D**istributed **S**imulation (**PADS**)

- There is **no global state**: this is the key aspect of **PADS**
- A **PADS** is the **interconnection** of a set of **model components**, usually called **Logical Processes** (**LP**s)
- Each **LP** is responsible to manage the evolution of only **a part of the simulation**
- Each **LP** has to interact with other **LP**s for **synchronization** and **data distribution**
- In practice, each **LP** is usually executed by a **processor** (or a **core** in modern multi-core architectures)

Synchronization: on the correct order of events

- Some kind of **network** interconnects the **LP**s running the simulation

- Each **LP** is executed by a different **CPU** (or **core**), **possibly at a different speed**

- The **network** can **introduce delays**

- The results of a **PADS** are **correct** only if its outcome is **identical** to the one obtained from the corresponding **sequential** simulation

- **Synchronization mechanisms** are used to **coordinate** the **LP**s: **different approaches are possible**

In-depth: **synchronization**, **causal ordering**

- All generated events have to be **timestamped** and delivered following a **message-passing** approach
- Two events are in **causal order** if one of them **can have some consequences on the other**
- The execution of events in **non causal order** leads to **causality errors**
- In a sequential simulation it is easy avoid causality errors given that there is a single ordered pending event list
- But in a **PADS** this is **much harder**!
- In this case the goal is to:
  - *execute events **in parallel**, as much as possible*
  - *do not introduce **causality errors***

In-depth: **synchronization**, **approaches**

- The most studied aspect in PADS because of its importance

- Many different approaches and variants have been proposed, with some simplification **three main methods**:

  - **time-stepped**: *the simulated time is divided in* ***fixed-size timesteps***

  - **conservative**: ***causality errors are prevented,*** *the simulator is built to avoid them*

  - **optimistic**: *the* ***causality constraint can be violated*** *and errors introduced. In case of causality violations the simulator will fix them*

In-depth: **synchronization**, **optimistic**

- The **LP**s are **free to violate** the **causality constraint**
- They can **process events in receiving order** (vs. timestamp order)
- There is **no *a priori* attempt** to **avoid causality violations**
- In case of violation this will be **detected** and appropriate mechanisms will be used to **go back to a prior state**
- The main mechanism is the **roll back of internal state variables** of the **LP** in which happened the violation
- If the **error propagated** to other **LP**s, then also the **roll back has to be propagated** to all the affected **LP**s

In-depth: **synchronization**, **Time-warp**

- The Jefferson's **Time Warp** mechanisms implements optimistic synchronization
- Each **LP** processes all events that it has received up to now
- An event is “**late**” if it has a timestamp that is smaller than the current clock value of the **LP** (that is the timestamp of the last processed event)
- The violation of **local causality** is fixed with the **roll-back** of all the **internal state variables** of the simulated model
- The violation has likely propagated to other **LP**s
- The goal of “**anti-messages**” is to **annihilate** the corresponding unprocessed events in **LP**s pending event list or to cause a **cascade of roll-backs** up to a **globally correct state**

What is **next**? What is **wrong**?

- **Multi** and **many cores processors**

- **General purpose CPUs**: Intel 10-core Xeon processors, UltraSPARC T3 (16 cores), AMD FX-series (up to 8 cores)

- **Embedded market**: Tile-GX (100 cores) and many others

- **In the (near) future:** Intel **Many Integrated Core (MIC)** architecture

  #cores -> 32... 64...

- **As many LPs as cores?**

What is **next**? What is **wrong**?

- **Multi a**
- **Gener** rs,

  UltraS es)
- **Embe** thers
- **In the**

  Integ

  #cores ->
- **As many LPs as cores?**

Increasing the number of parts

makes the model partitioning

harder and harder

A solution is to work on each single LP

(parallelizing it)

but with current programming

languages this is not easy at all

The **Go programming language**

- **General purpose programming language** announced by Google in 2009, Open Source project

- Very **easy** and **clean syntax**, with **garbage collection**

- The language core provides support for **concurrent execution** and **inter-process communication**

- **Main new features:**
  - **goroutines**
  - **channels**

Go: goroutines

- Function **executing in parallel** with other goroutines, **in the same address space**

- **Lightweight** implementation, goroutines can communicate using shared memory

- Multiplexed into **multiple OS threads**

- If a goroutine is blocked waiting for I/O the others can continue to run

- It is possible to pack multiple-goroutines **in the same OS thread**, to further reduce overhead

- **Very easy to implement**: prefix a function or method call with the "go" keyword

Go: channels (chan)

- Used for the **communication between goroutines**
- A chan is a **data type** that can be used for both **communication** and **synchronization**
- The capacity of the chan is given by its buffer size
- Zero capacity channels are **synchronous** and are used for **synchronizing goroutines**
- In all other cases the channels are **asynchronous** and used for the **transmission of typed messages**

Go-Warp: design and implementation

- **Simulator** based on the **Time Warp** synchronization algorithm
- Each **LP** is implemented using a **single goroutine**
- **LP-to-LP** communication uses **asynchronous chans**
- Some **shared variables** ease the implementation of specific tasks (e.g. Samadi's GVT calculation, fossil collection)
- **In the next version**: **parallel execution** of some **LP internal mechanisms**

Performance evaluation: **PHOLD benchmark**

- It is a **simulation model**, the ***de facto* standard** for the **performance evaluation** of Time Warp implementations
- A **set of entities**, partitioned among the **LPs**
- Each **LP** contains the same number of **entities**
- Each **entity produces** and **consumes events**
- When an **event** is processed, a new one is created and delivered to a (randomly chosen) **entity**
- Fixed total number of events, “almost steady state” model

Performance evaluation: **PHOLD parameters**

- Number of **simulated entities** (#entities)
- **Event density**: amount of time elapsed from the receiving of an event and the generation of a new one (density)
- **Workload**: amount of synthetic work executed by the LP when an event is processed (FPops)
- **Standard values** in the following performance evaluation: *simulation length = 1000 time-units, #entities = 1500, density = 0.5 time-units, FPops = 10000*

Execution environment and methodology

- **Intel**(R) **Core**(TM) **i7**-2600 CPU 3.40GHz with **4 cores** and **Hyper-Threading** (HT) technology

Execution environment and methodology

- **Intel**(R) **Core**(TM) **i7**-2600 CPU 3.40GHz with **4 cores** and **Hyper-Threading** (HT) technology

HT works duplicating some parts of the processor except the main execution units

For the OS, each physical processor core is seen as two "virtual" processors

Execution environment and methodology

- **Intel**(R) **Core**(TM) **i7**-2600 CPU 3.40GHz with **4 cores** and **Hyper-Threading** (HT) technology

8 virtual cores on a desktop PC

HT works duplicating some parts of the processor except the main execution units

For the OS, each physical processor core is seen as two "virtual" processors

Execution environment and methodology

- **Intel**(R) **Core**(TM) **i7**-2600 CPU 3.40GHz with **4 cores** and **Hyper-Threading** (HT) technology
- 8 GB RAM
- Ubuntu 11.10 (x86_64 GNU/Linux, 3.0.0-15-generic #26-Ubuntu SMP
- **Multiple runs, controlled environment, average results**

Performance evaluation: WCT

Average Wall Clock Time (milliseconds)

	Number of Cores							
#LPs	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	**1704**	1691	1701	1685	1700	1683	1703	1685
2		**1050**	1049	1051	1056	1047	1049	1050
3			**864**	854	858	856	865	853
4				**787**	799	787	807	**785**
5					**795**	**775**	**778**	790
6						817	823	822
7							817	842
8								908

real cores **hyper-threading**

Performance evaluation: WCT

Average Wall Clock Time (milliseconds)

	Number of Cores							
#LPs	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	**1704**	1691	1701	1685	1700	1683	1703	1685
2		**1050**	1049					[illegible]
3								
4								
5								
6								
7								
8								

Having #LPs > cores

is not a good idea:

- more context switches
- imbalances and extra rollbacks

real cores **hyper-threading**

Performance evaluation: WCT

Average Wall Clock Time (milliseconds)

	Number of Cores							
#LPs	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	**1704**	1691	1701	1685	1700	1683	1703	1685
2		**1050**	1049	1051	1056	1047	1049	1050
3			**864**	854	858	856	865	853
4				**787**	799	787	807	**785**
5					**795**	**775**	**778**	790
6						817	823	822
7							817	842
8								908

real cores **hyper-threading**

Performance evaluation: speedup

	Number of Cores							
#LPs	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	**1**	1	1	1	1	1	1	1
2		**1.61**	1.62	1.60	1.61	1.61	1.62	1.60
3			**1.97**	1.97	1.98	1.97	1.97	1.98
4				**2.14**	2.13	2.14	2.11	**2.15**
5					**2.14**	**2.17**	**2.19**	2.13
6						2.06	2.07	2.05
7							2.08	2.00
8								1.86

real cores

hyper-threading

Performance evaluation: speedup

	Number of C							
#LPs	*1*	*2*	*3*	*4*	*5*		*7*	*8*
1	**1**	1	1	1	1		1	1
2		1						1.60
3								.98
4								**.15**
5								.13
6								.05
7								.00
8								.86

Speedup: ratio of the execution times of the sequential algorithm (*LP = 1*) and the parallel version (with *n LPs*)

real cores

hyper-threading

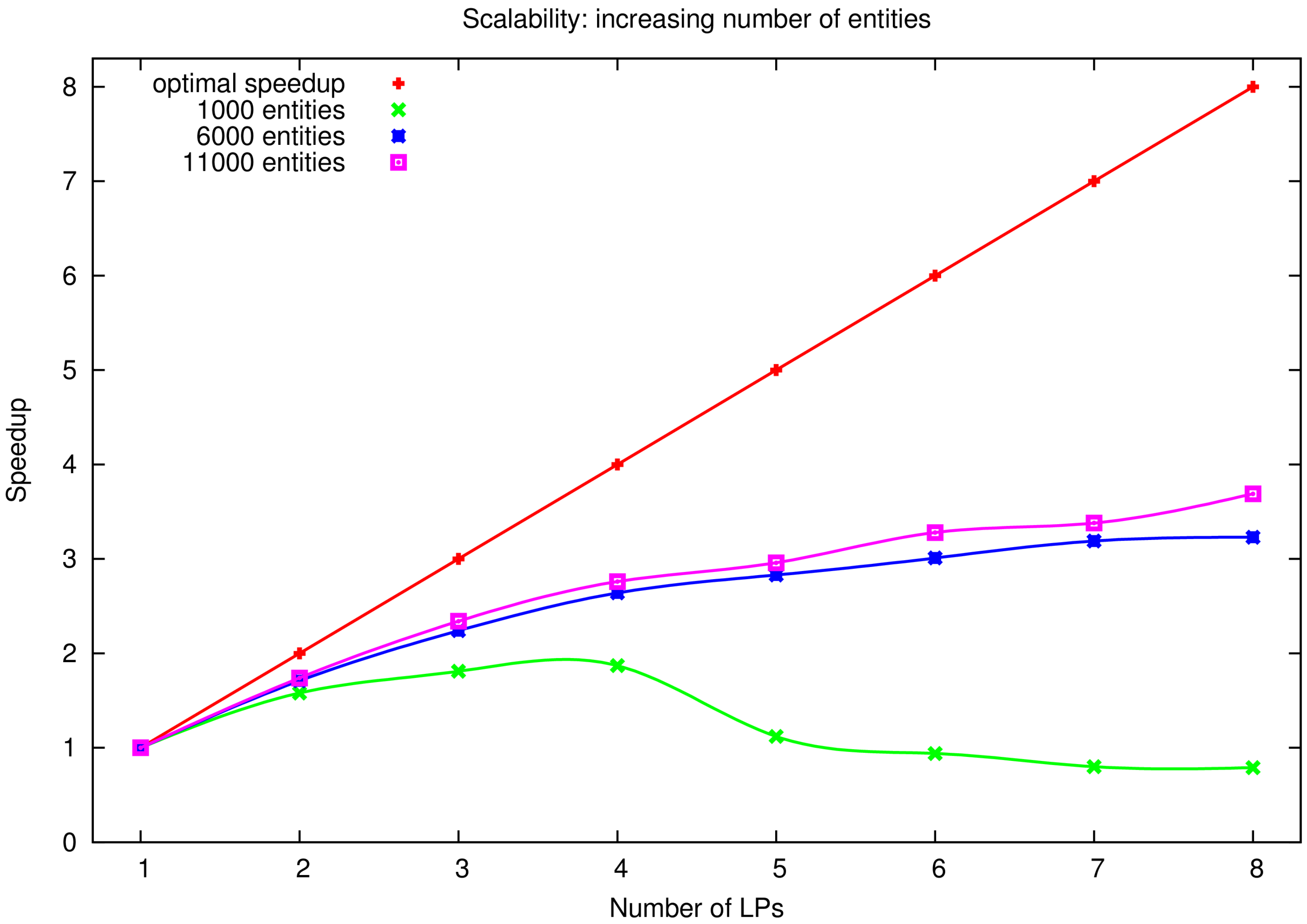
Performance evaluation: speedup

	Number of Cores							
#LPs	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	**1**	1	1	1	1	1	1	1
2		**1.61**	1.62	1.60	1.61	1.61	1.62	1.60
3			**1.97**	1.97	1.98	1.97	1.97	1.98
4				**2.14**	2.13	2.14	2.11	**2.15**
5					**2.14**	**2.17**	**2.19**	2.13
6						2.06	2.07	2.05
7							2.08	2.00
8								1.86

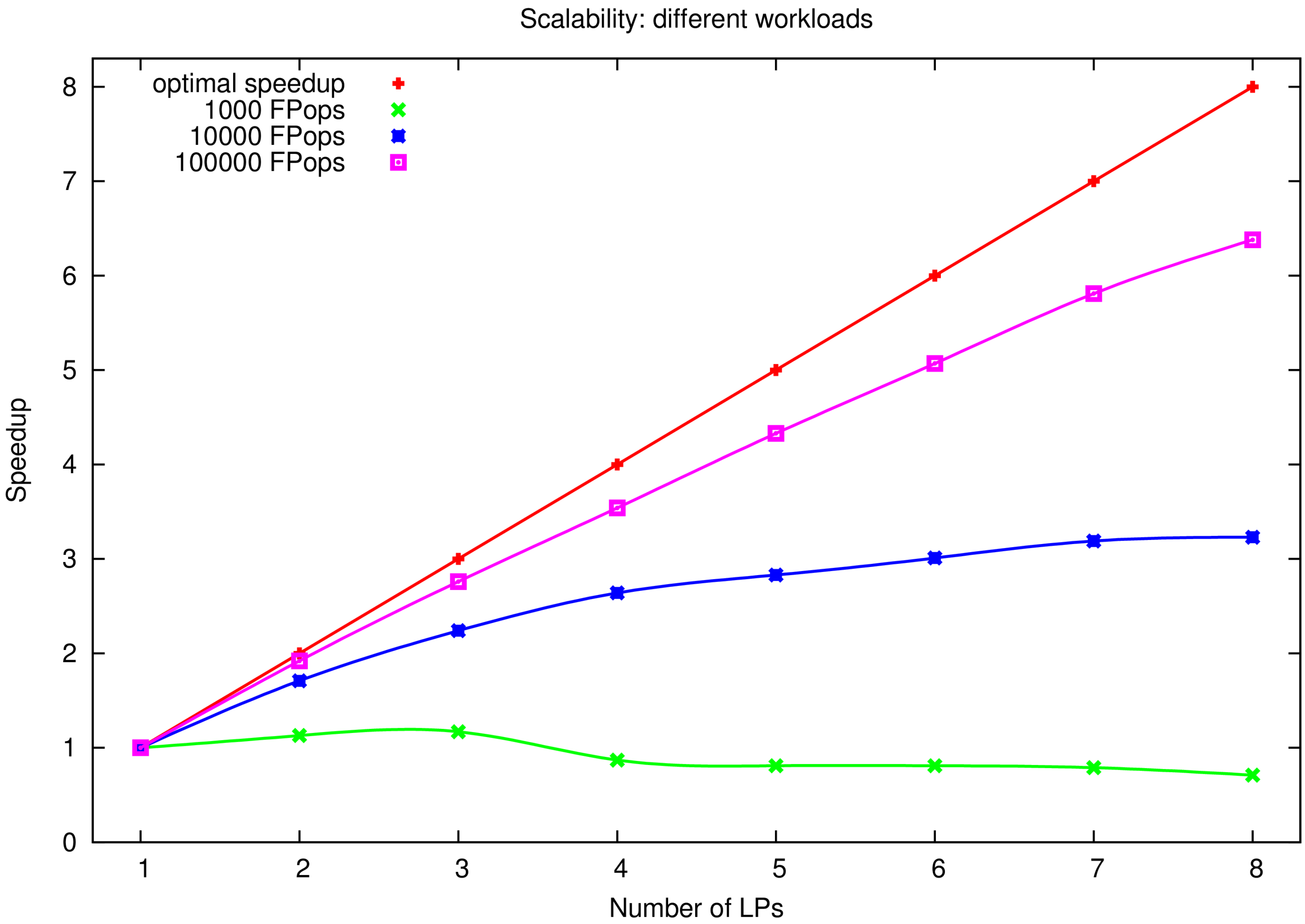
real cores

hyper-threading

Performance evaluation: #entities



Performance evaluation: workloads



Conclusions

- New approaches are needed to **deal with an increasing number of cores**
- The **LP** and the **simulation model** part that it implements **need to be parallelized**
- The **Go programming language** is an interesting choice
- The **Go-Warp simulator** needs to support some extra features but has shown encouraging performance results
- The next step is to work on **more realistic simulation models**

Further information

Gabriele D'Angelo, Stefano Ferretti, Moreno Marzolla

Time Warp on the Go

Proceedings of the 3rd Workshop on Distributed Simulation and Online gaming (DISIO). Desenzano, Italy, March 2012

An **extended version** of this paper will be soon available on the

open e-print archive

In the next months the **source code of Go-Warp** will be released at

http://pads.cs.unibo.it

Gabriele D'Angelo

- E-mail: <g.dangelo@unibo.it>
- http://www.cs.unibo.it/gdangelo/

Time Warp on the Go

Gabriele D'Angelo

<gda@cs.unibo.it>

http://www.cs.unibo.it/gdangelo/

joint work with:

Stefano Ferretti** and **Moreno Marzolla

Desenzano, Italy

DIstributed **SI**mulation & **O**nline gaming (**DISIO**), 2012