Debugging Large Scale Parallel Applications

Filippo Gioachin

Parallel Programming Laboratory
Departement of Computer Science
University of Illinois at Urbana-Champaign

Outline

- Introduction
  - Motivations
- Debugging on Large Machines
  - Scalability
- Using Fewer Resources
  - Virtualized Debugging
  - Processor Extraction
- Summary

Motivations

- Debugging is a fundamental part of software development
- Parallel programs have all the sequential bugs:
  - Memory corruption
  - Incorrect results
  - ....

Motivations (2)

- Parallel programs have other bugs:
  - Data races / multicore (heavily studied in literature)
  - Communication mistakes
  - Synchronization mistakes / Message races
- To complicate things more:
  - Non-determinism
  - Problems may show up only at large scale

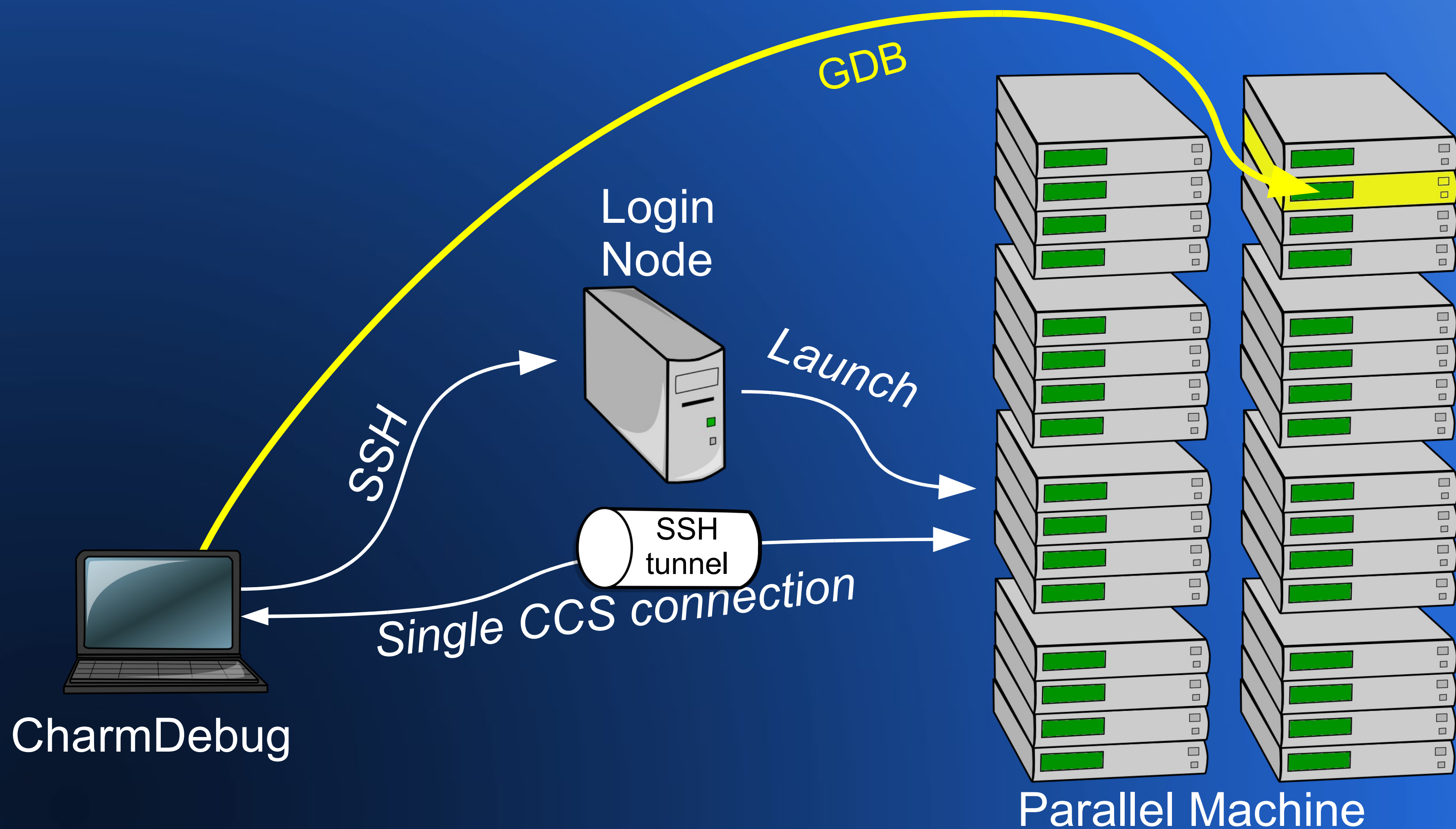
Problems at Large Scale

- Problems may not appear at small scale
  - Races between messages
    - Latencies in the underlying hardware
  - Incorrect messaging
  - Data decomposition

Problems at Large Scale (2)

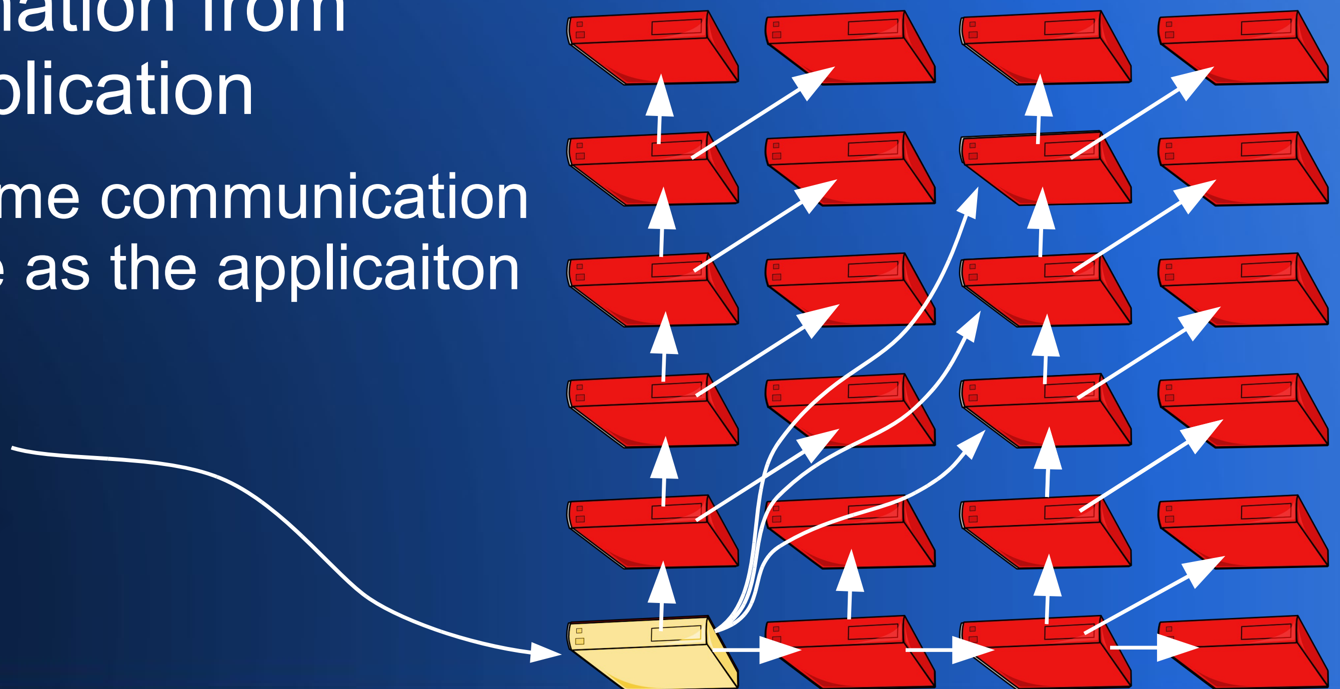
- Infeasible
  - Debugger needs to handle many processors
  - Human can be overwhelmed by information
  - Long waiting time in queue
  - Machine not available
- Expensive
  - Large machine allocation consume a lot of computational resources

CharmDebug Overview



Converse Client-Server Scalability

- CCS connects to the application as a whole
  - Forward requests for single processors
  - Gather information from the whole application
    - Uses the same communication infrastucture as the applicaiton



Debugging on Large Systems

- Attaching to running application
  - 48 processors cluster
    - **28 ms** with 48 point-to-point queries
    - **2 ms** with a single global query
- Example: Memory statistics collection
  - **12 to 20 ms** up to 4k processors
  - Counted on the client debugger

* F. Gioachin, C.W. Lee, L.V. Kalé: "Scalable Interaction with Parallel Applications", *In Proceedings of TeraGrid'09*

Autoinspection

- The programmer should not manually handle all the processors
  - Unsupervised execution
  - Notification to the user from interesting processors
    - Breakpoints
    - Abort / signals
    - Memory corruption
    - Assertion failure

Python Scripting

- Upload a script to perform checking on the correctness of data structures when needed

* F. Gioachin, L.V. Kalé: "Dynamic High-Level Scripting in Parallel Applications". *In Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009)*

Python script

HelloGroup (9)

```
def method(self):
    lehgth = charm.getValue(self, MyArray, len)
    arr = charm.getValue(self, MyArray, data)
    for i in range(0, length):
        value = charm.getArray(arr, double, i)
        if (value > 10 or value < -10):
            print "Error: value ", i, " = ", value
            return i
```

System Entries
User Entries
Main
Hello
Hello(CkMigrateM
Hello(void)
SayHi(int hiNo)
HelloGroup
HelloGroup(void)
SayHiGroup(int n)
HelloNode
HelloChare
SecondArray

Send Cancel

Suspend execution if a value is returned

Access program's data (circumvent lack of reflection)

Select on which entry points the script should run

Can you debug on a big machine?

- Feasibility
  - How long do you have to wait before your job starts?
    - Are you available when you job starts?
  - Is the machine even available?
- Cost
  - How many allocation units are you using to do your debugging?

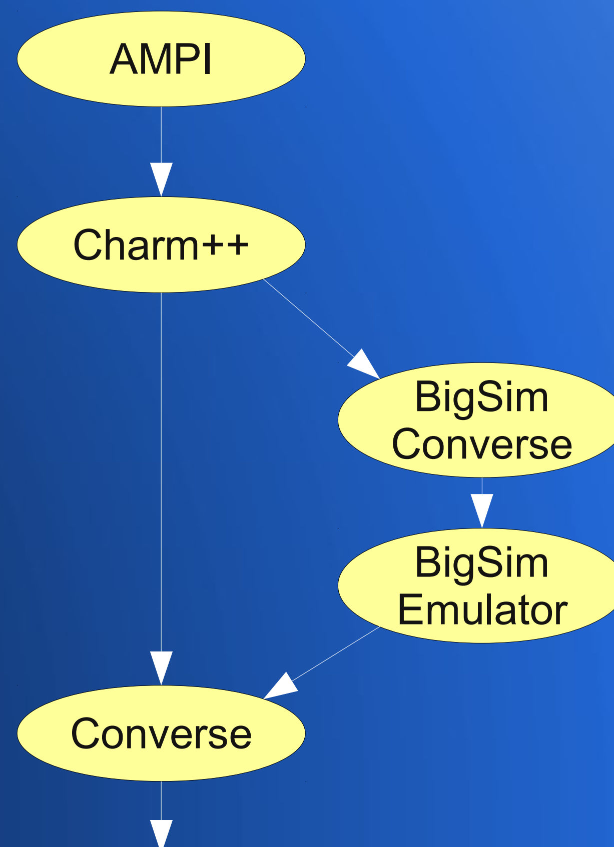
Virtualized Emulation

- Use emulation techniques to provide virtual processors to display to the user
  - Different scenario from performance analysis
    - Cannot assume correctness of program
  - Debugger needs to communicate with application
  - Single address space

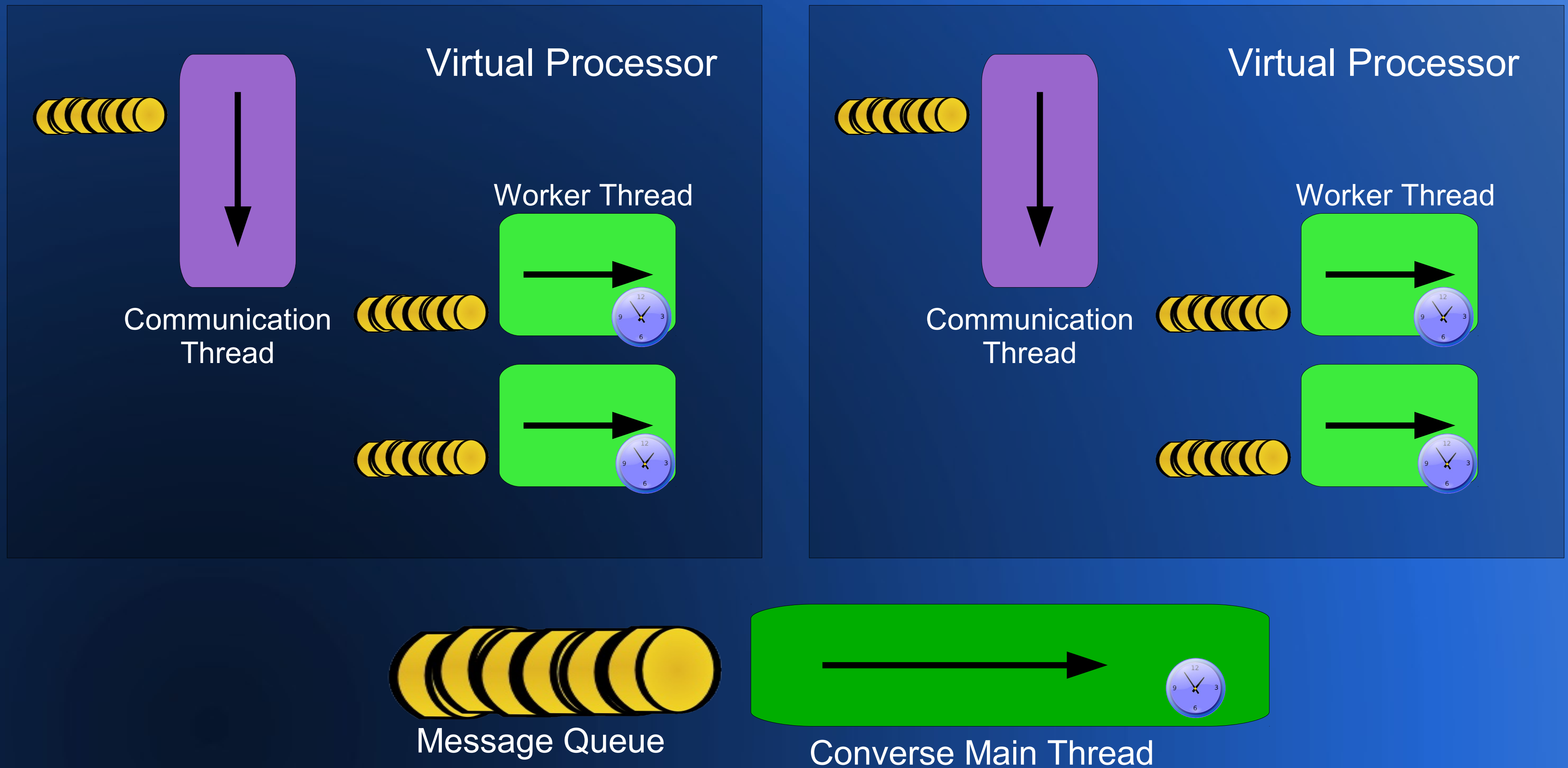
* F. Gioachin, G. Zheng, L.V. Kalé: "Debugging Large Scale Applications in a Virtualized Environment". *PPL Technical Report, April 2010*

Virtualized Charm++

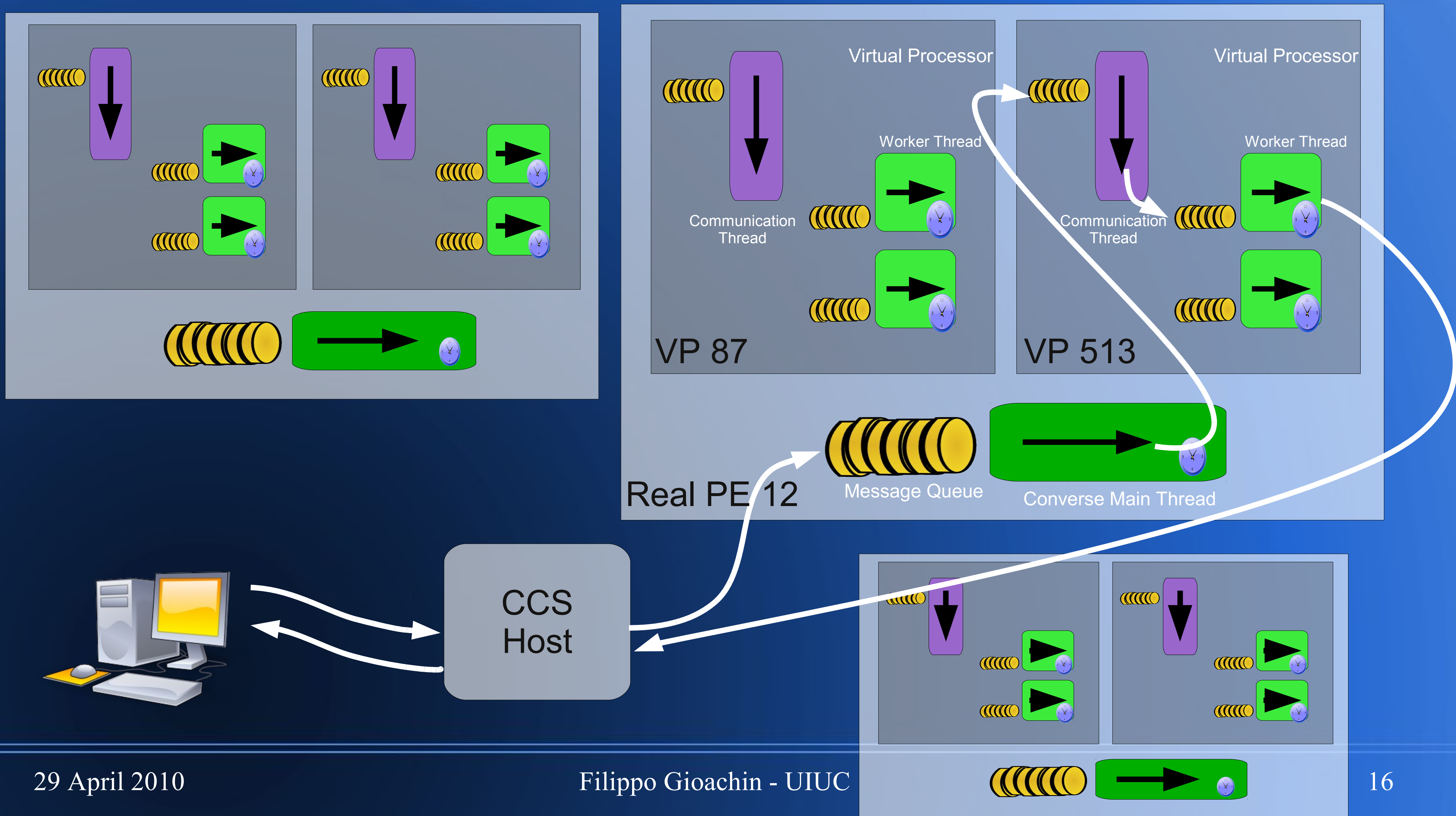
- Converse on top of BigSim
  - Processors become virtual processors
  - Two Converse layers
    - Virtualized
    - Original



BigSim Emulator



Converse Client-Server under Emulated Environment



Usage: Starting

Program Parameters

Executable: ../ChaNGa [Change]
Working dir: 1ome/gioachin/cosmology/CVS_latest/ChaNGa1/changa/testcosmo [Change]
Command Line Parameters: cube300.param
Number of Processors: 32
☑ Virtualized debugging: Number of Virtal Processors: 2048
Port Number:
SSH port number: 0
Host name: localhost
Username:
☐ Use ssh tunneling [OK] [CANCEL]

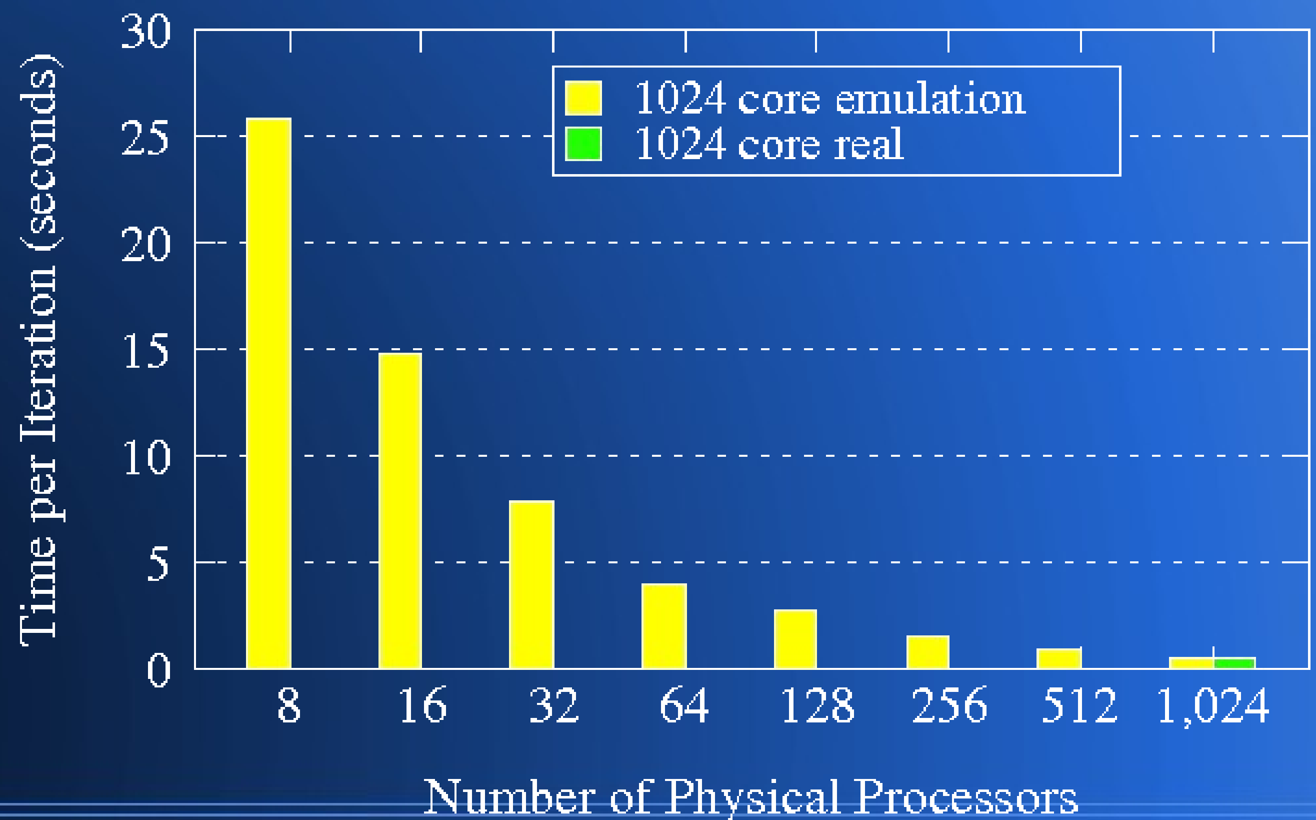
Program Parameters

Executable: ../ChaNGa [Change]
Working dir: 1ome/gioachin/cosmology/CVS_latest/ChaNGa1/changa/testcosmo [Change]
Command Line Parameters: cube300.param
Number of Processors: 0
☐ Virtualized debugging: Number of Virtal Processors:
Port Number: 1234
SSH port number: 0
Host name: localhost
Username:
☐ Use ssh tunneling [OK] [CANCEL]

Usage: Debugging

Performance: Jacobi
(on NCSA's BluePrint)

- User thinks for one minute about what to do:
  - 8 processors
    - 86 sec.
    - ~0.2 SU
  - 1024 procs
    - 60.5 sec.
    - ~17 SU

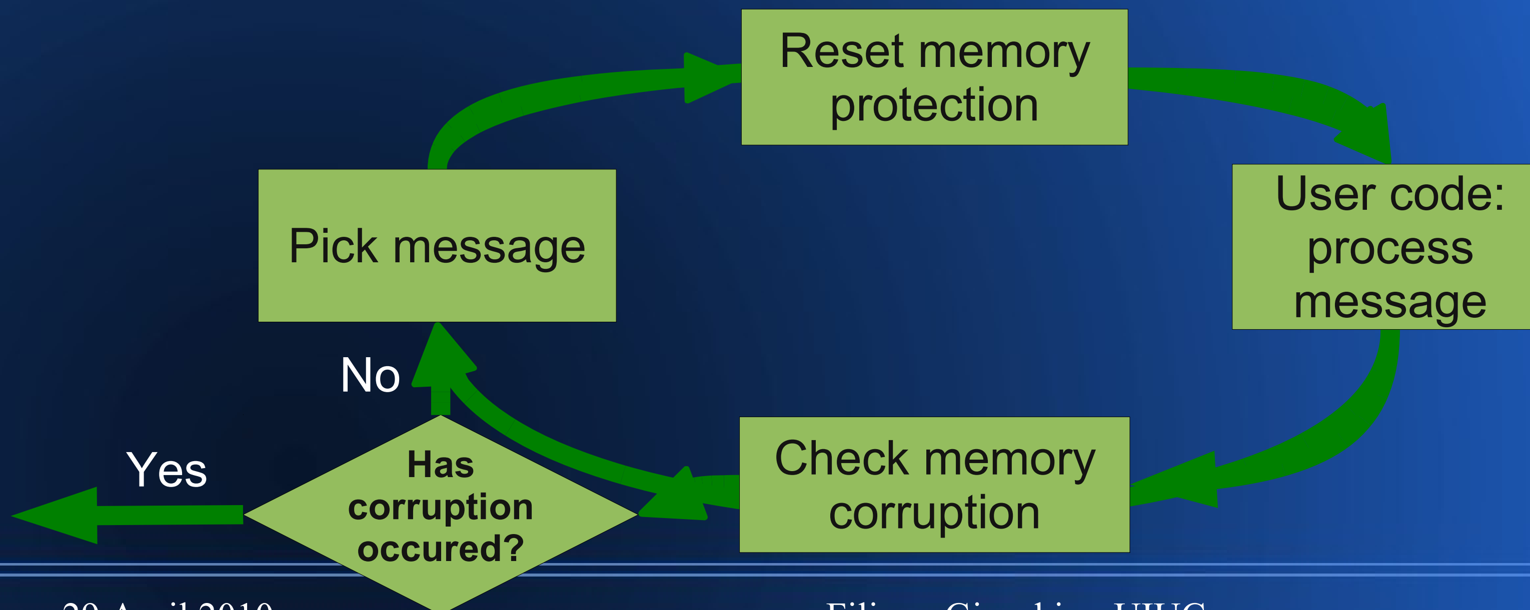


Restrictions

- Small memory footprint
  - Many processors needs to fit into a single physical processor
- Session should be constraint by human speed
  - Allocation idle most of the time waiting for user input
  - Bad for computation intensive applications

Separation of Virtual Entities

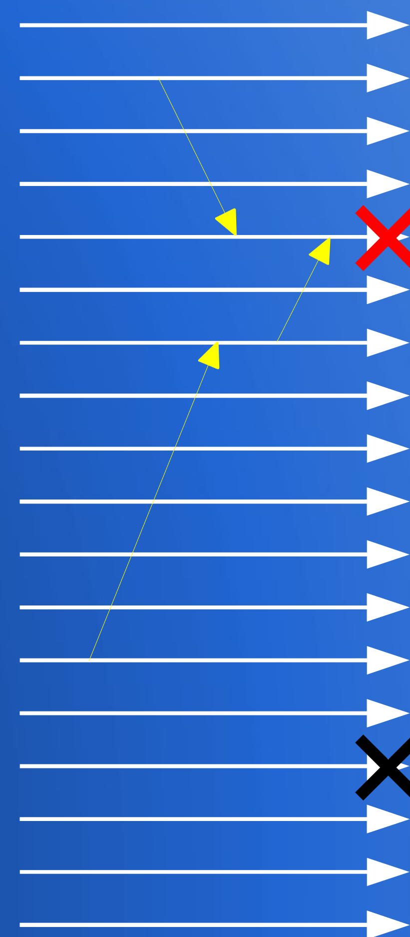
- Single address space shared by different entities
  - One entity can write in memory of another entity
    - Protect memory such that spurious writes can be detected
    - Exploit the scheduler in message driven systems



* F. Gioachin, L.V. Kalé: "Memory Tagging in Charm++" *in Proceedings of the 6th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD '08)*

Do we need all the processors?

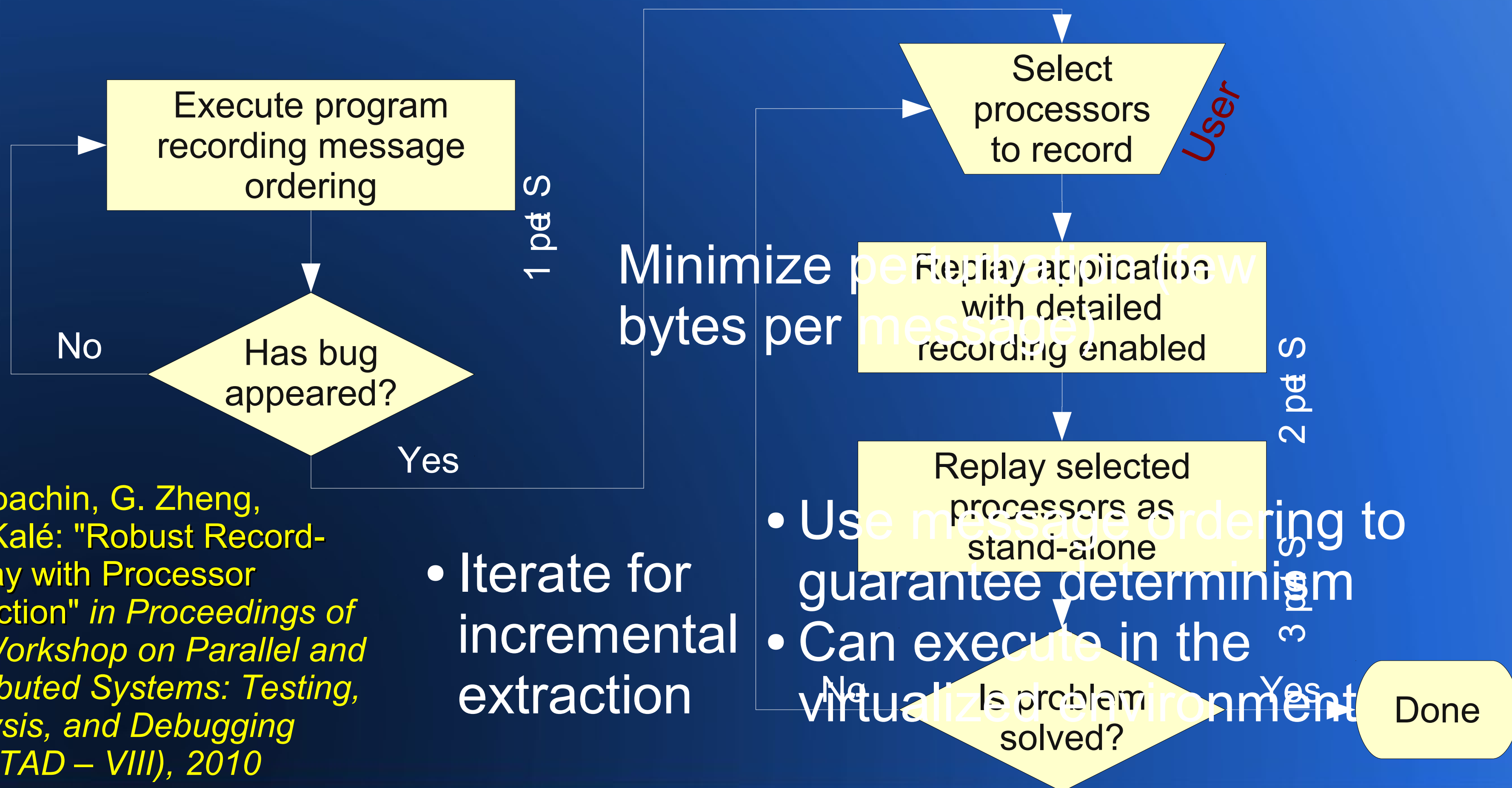
- The problem manifests itself on a single processor
  - If more than one, they are equivalent
- The cause can span multiple processors (causally related)
  - The subset is generally much smaller than the whole system
- Select the interesting processors and ignore the others



Fighting non-determinism

- Record all data processed by each processor
  - Huge volume of data stored
  - High interference with application
    - Likely the bug will not appear...
  - Need to run a non-optimized code
- Record only message ordering
  - Based on piecewise deterministic assumption
  - Must re-execute using the same machine

Three-step Procedure for Processor Extraction



Minimize perturbation (few bytes per message)

- Iterate for incremental extraction

- Use message ordering to guarantee determinism
- Can execute in the virtualized environment

* F. Gioachin, G. Zheng, L.V. Kalé: "Robust Record-Replay with Processor Extraction" *in Proceedings of the Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD – VIII), 2010*

What if the piecewise deterministic assumption is not met?

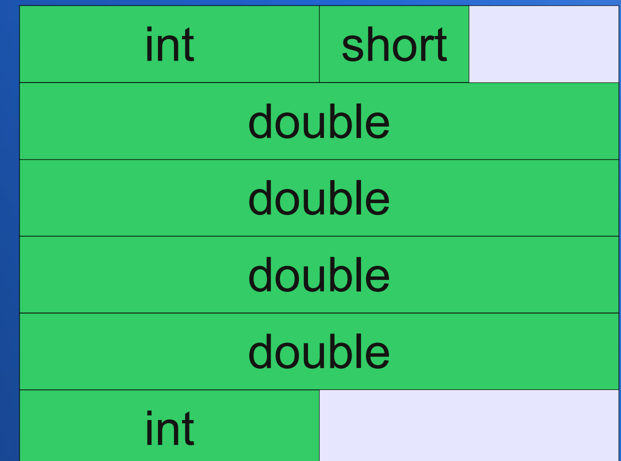
- Make sure to detect it, and notify the user

If all messages are identical, then we can assume the non-determinism was captured

- Methods to detect failure:
  - Message size and destination
  - Checksum of the whole message (XOR, CRC32)

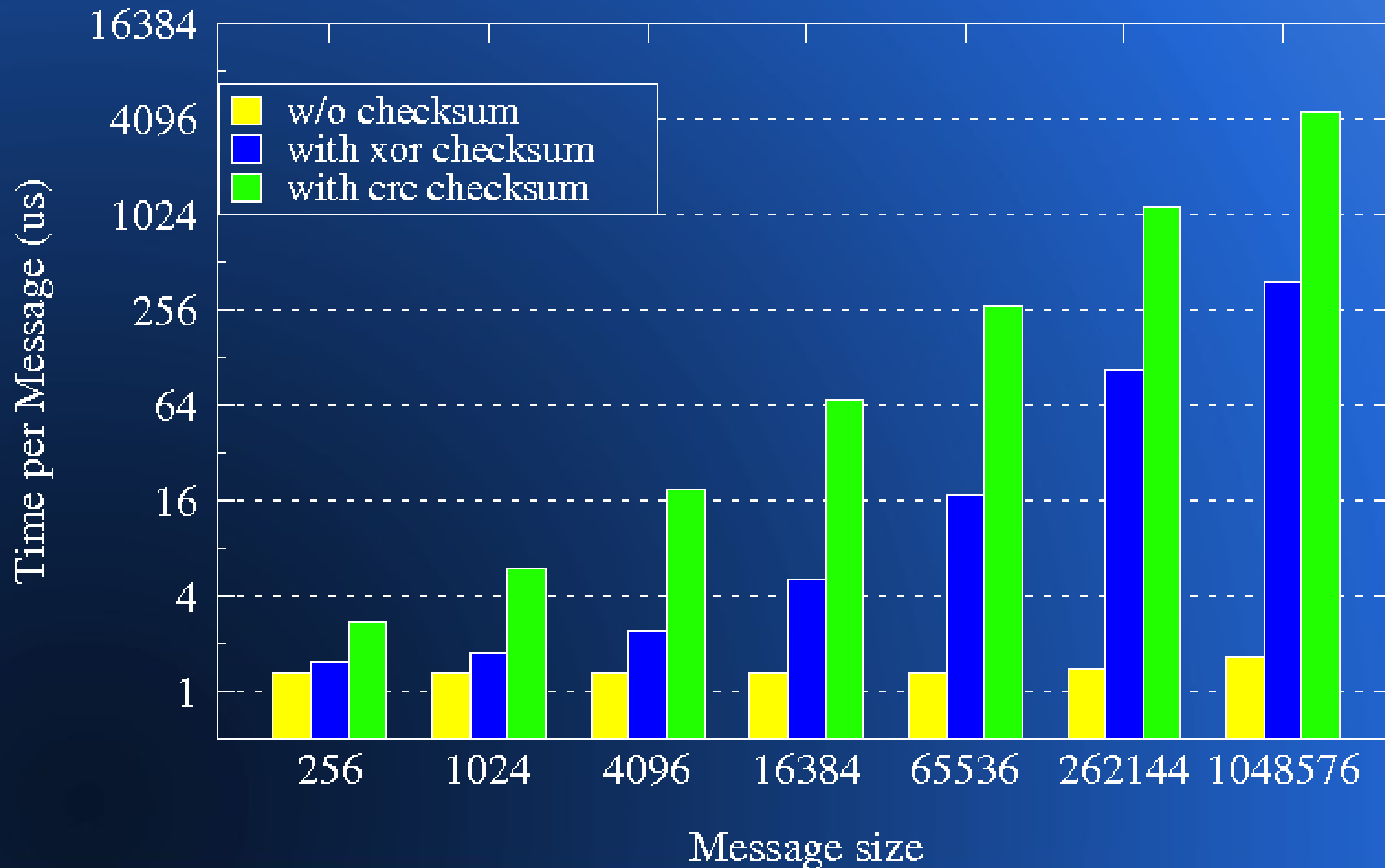
Computing Checksums

- Checksum considers memory as raw data, ignores what it contains
  - Pointers
  - Garbage
    - Uninitialized fields
    - Compiler padding

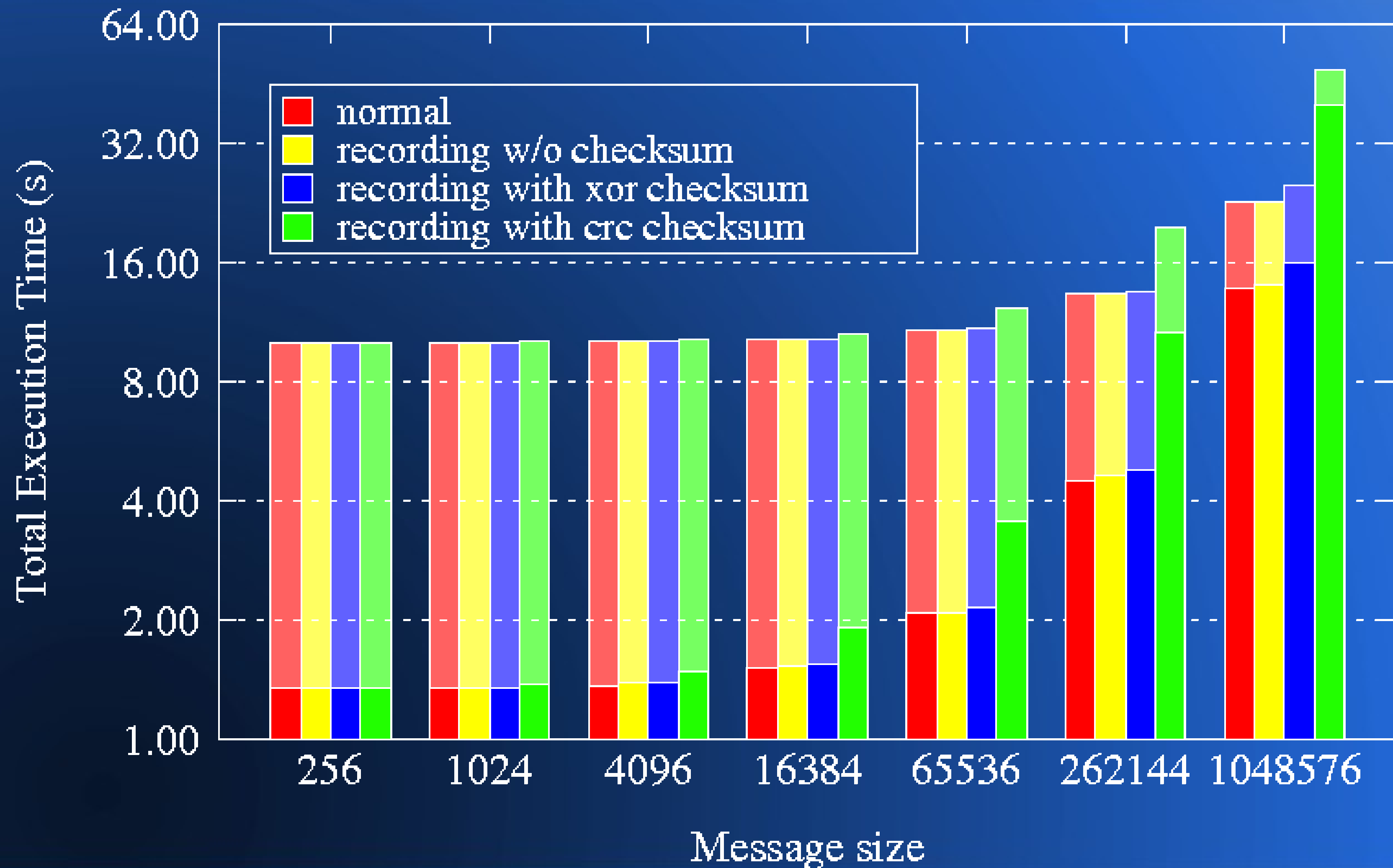


- Use Charm++ memory allocator
  - Intercept calls to *malloc* and pre-fill memory

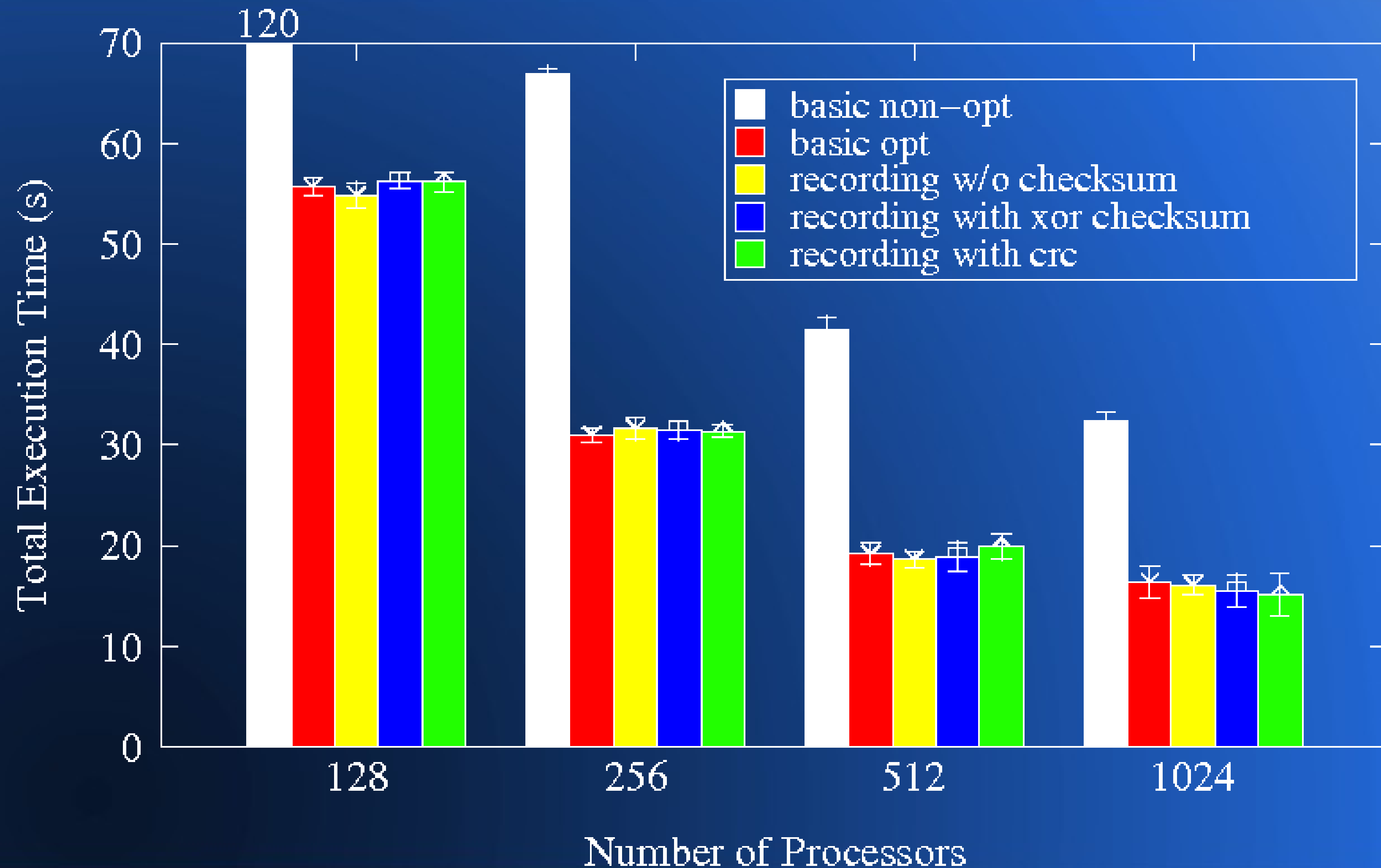
Message Order Recording Performance (on NCSA's Abe)



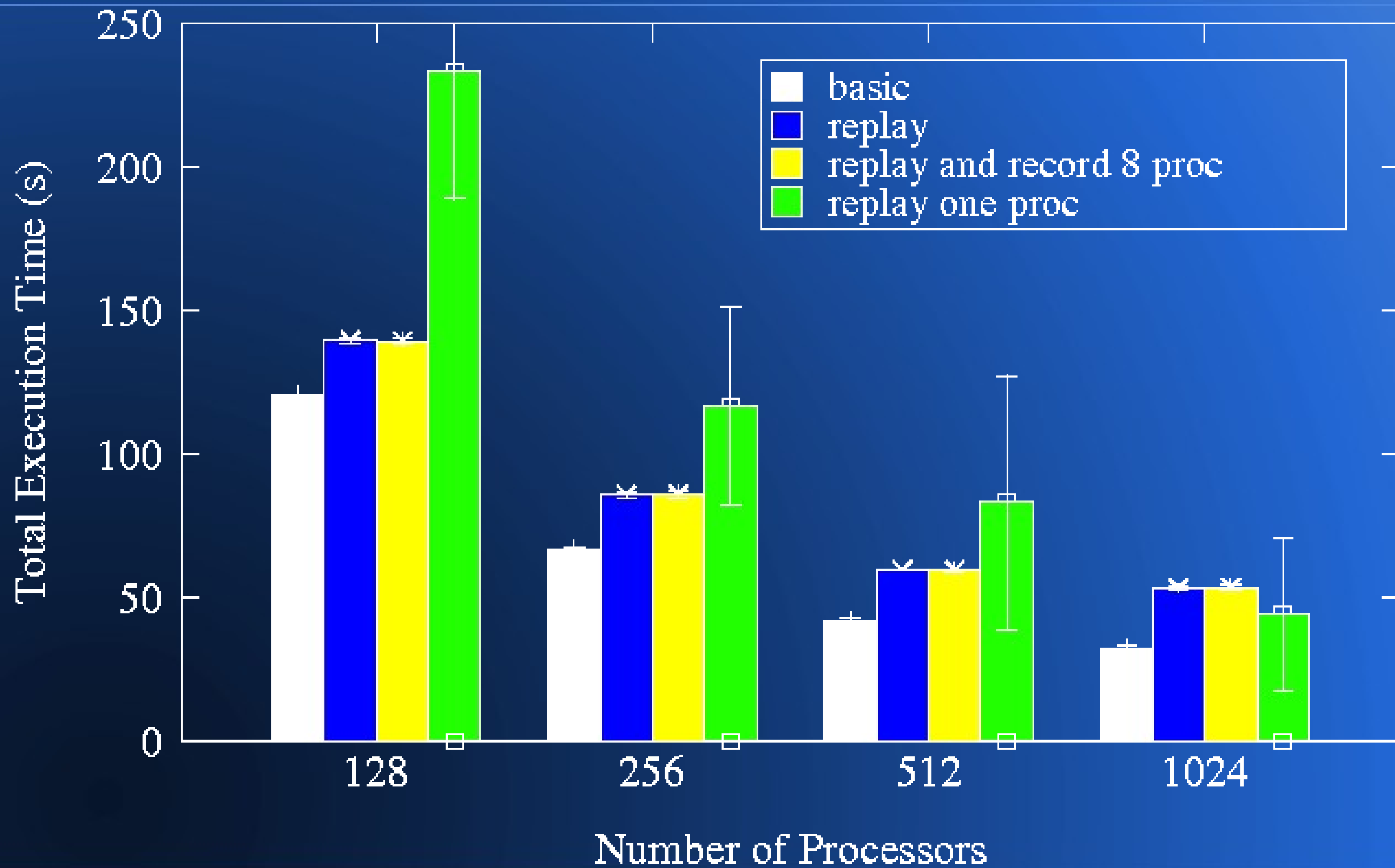
kNeighbor



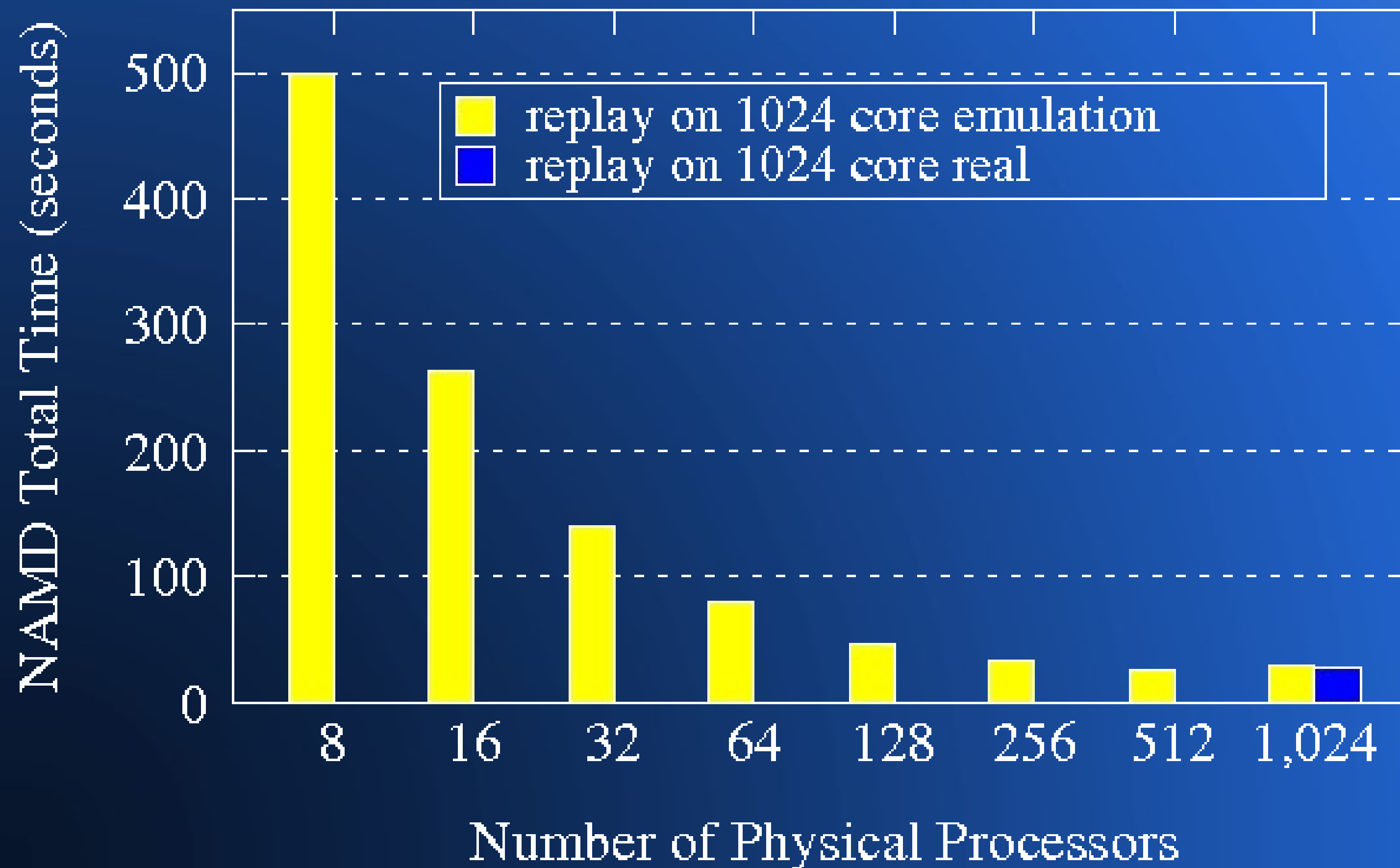
ChaNGa
(dwf1.2048 on NCSA's BluePrint)



Replaying the Application



Replaying under BigSim Emulation: NAMD



Amount of Data Saved

ChaNGa dwf1.2048, numbers in MB

Number of Processors		128	256	512	1024
Record	Per-processor	0.87	0.67	0.54	0.44
	Total	112	173	279	453
Record+checksum	Per-processor	1.49	1.14	0.92	0.75
	Total	190	292	473	765
Detailed record	Per-processor	111	79	59	47

Debugging Case Study

- Message race during particle exchange
  - Fixed with tedious print statements (while trying to avoid hiding the bug...)

```
../charmdebug +p16 ../ChaNGa cube300.param +record
        +recplay-crc

../charmdebug +p16 ../ChaNGa cube300.param +replay
        +recplay-crc +record-detail 7

gdb ../ChaNGa
>> run cube300.param +replay-detail 7/16
```

Summary

- Important for the debugging system to scale to large configurations
- Resources are expensive and should not be wasted
  - Virtualized Debugging to debug large scale applications on small clusters
  - Processor Extraction to capture non-determinism of parallel application
    - Must not interfere too much with the application timing

Future Extensions

- Shared memory compliance
- Race detector
  - Automated testing of message delivery to discover message races
- Replay in isolation of single virtual entities
  - Conditions of validity