Revisiting the Issue of Performance Enhancement of Discrete Event Simulation Software [1]

Alex Bahouth, Steven Crites, Norman Matloff and Todd Williamson
Department of Computer Science
University of California at Davis
Davis, CA 95616 USA
matloff@cs.ucdavis.edu

[1] We wish to thank Victor Castillo and the Lawrence Livermore National Laboratory for supporting this research.

This presentation is produced using C. Campani's Beamer LaTeX class.
See `http://heather.cs.ucdavis.edu/~matloff/beamer.html` for a quick tutorial.
Disclaimer: Our slides here won't show off what Beamer can do. Sorry. :-)

Issues Addressed in This Paper

- Interpreted languages (Java, Python) now popular for DES

Issues Addressed in This Paper

- Interpreted languages (Java, Python) now popular for DES
- Interpreted languages are slow.

Issues Addressed in This Paper

- Interpreted languages (Java, Python) now popular for DES
- Interpreted languages are slow.
- DES literature mainly algorithm-centric.

Issues Addressed in This Paper

- Interpreted languages (Java, Python) now popular for DES
- Interpreted languages are slow.
- DES literature mainly algorithm-centric.
- What can be done specifically for interpreted languages?

Issues Addressed in This Paper

- Interpreted languages (Java, Python) now popular for DES
- Interpreted languages are slow.
- DES literature mainly algorithm-centric.
- What can be done specifically for interpreted languages?
- What can be done for systems considerations, e.g. VM?

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.

About SimPy:

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.

About SimPy:

- Written by Klaus Muller and Tony Vignaux.

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.
About SimPy:

- Written by Klaus Muller and Tony Vignaux.
- I have developed an online DES course based on SimPy, available at `heather.cs.ucdavis.edu/~matloff/simcourse.html`.

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.

About SimPy:

- Written by Klaus Muller and Tony Vignaux.
- I have developed an online DES course based on SimPy, available at `heather.cs.ucdavis.edu/~matloff/simcourse.html`.
- SimPy uses Python:

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.

About SimPy:

- Written by Klaus Muller and Tony Vignaux.
- I have developed an online DES course based on SimPy, available at `heather.cs.ucdavis.edu/~matloff/simcourse.html`.
- SimPy uses Python:
  - Lots of high-level Python constructs make programming much easier.

Case Study: SimPy

Our investigation took the form of a case study: enhancing the peformance of the SimPy DES language.
About SimPy:

- Written by Klaus Muller and Tony Vignaux.
- I have developed an online DES course based on SimPy, available at `heather.cs.ucdavis.edu/~matloff/simcourse.html`.
- SimPy uses Python:
  - Lots of high-level Python constructs make programming much easier.
  - Python *generator* construct used by SimPy to set up coroutines, i.e. non-preemptive threads.

Sample SimPy Code

Sample SimPy Code

- Machine repair, several machines.

Sample SimPy Code

- Machine repair, several machines.
- Have class **MachineClass**, with member variables such as **UpTime**, etc.

Sample SimPy Code

- Machine repair, several machines.
- Have class **MachineClass**, with member variables such as **UpTime**, etc.
- Each class has a member function **Run()** which simulates one machine.

Sample Run() Function

```
def Run(self):
  while 1:
    self.StartUpTime = SimPy.Simulation.now()
    # hold for up time
    UpTime = G.Rnd.expovariate(MachineClass.UpRate)
    yield SimPy.Simulation.hold,self,UpTime
    # update up time total
    MachineClass.TotalUpTime +=
      SimPy.Simulation.now() - self.StartUpTime
    RepairTime = G.Rnd.expovariate(MachineClass.RepairRate)
    # hold for repair time
    yield SimPy.Simulation.hold,self,RepairTime
```

Sample Run() Function

```
def Run(self):
  while 1:
    self.StartUpTime = SimPy.Simulation.now()
    # hold for up time
    UpTime = G.Rnd.expovariate(MachineClass.UpRate)
    yield SimPy.Simulation.hold,self,UpTime
    # update up time total
    MachineClass.TotalUpTime +=
      SimPy.Simulation.now() - self.StartUpTime
    RepairTime = G.Rnd.expovariate(MachineClass.RepairRate)
    # hold for repair time
    yield SimPy.Simulation.hold,self,RepairTime
```

The yield actually does yield the processor.

Sample Run() Function

```
def Run(self):
  while 1:
    self.StartUpTime = SimPy.Simulation.now()
    # hold for up time
    UpTime = G.Rnd.expovariate(MachineClass.UpRate)
    yield SimPy.Simulation.hold,self,UpTime
    # update up time total
    MachineClass.TotalUpTime +=
      SimPy.Simulation.now() - self.StartUpTime
    RepairTime = G.Rnd.expovariate(MachineClass.RepairRate)
    # hold for repair time
    yield SimPy.Simulation.hold,self,RepairTime
```

The yield actually does yield the processor. But yield is a coroutine release—next time this function runs, it resumes after the yield.

SimPy Data Structures

- Assume for simplicity no tied event times.

SimPy Data Structures

- Assume for simplicity no tied event times.
- The Python list **timestamps** stores all event times, in ascending order. e.g. to determine the earliest scheduled event.

SimPy Data Structures

- Assume for simplicity no tied event times.
- The Python list **timestamps** stores all event times, in ascending order. e.g. to determine the earliest scheduled event.

  A Python list is not an array! One may insert and delete elements, with the corresponding overhead of shifting data.

SimPy Data Structures

- Assume for simplicity no tied event times.
- The Python list **timestamps** stores all event times, in ascending order. e.g. to determine the earliest scheduled event.
  A Python list is not an array! One may insert and delete elements, with the corresponding overhead of shifting data.
- The actual events are in a Python *dictionary* (associative array) named **events**.
  Python dictionaries are implemented as hash tables, reasonably fast.

SimPy Queue Operations

SimPy Queue Operations

When a new event is created at time t, then these operations occur:

(i) add t to list **timestamps**

(ii) add event to dictionary **events**

SimPy Queue Operations

When a new event is created at time t, then these operations occur:

(i) add t to list **timestamps**

(ii) add event to dictionary **events**

Step (i) makes use of Python's **bisect()** function, which performs bisection sort.

SimPy Queue Operations

When a new event is created at time t, then these operations occur:

(i) add t to list **timestamps**

(ii) add event to dictionary **events**

Step (i) makes use of Python's **bisect()** function, which performs bisection sort.
That would appear to be O(log n) time, for an n-item event list.
Due to SimPy's use of Python's list structure, it is actually O(n), due to right-shifting of the data.

SimPy Dequeue Operations

SimPy Dequeue Operations

When the next event is executed, these operations occur:

(iii) remove head of list **timestamps**, time t

(iv) reactivate (invoke Python iterator for) **Run()** function for event of time t in dictionary **events**

SimPy Dequeue Operations

When the next event is executed, these operations occur:

(iii) remove head of list **timestamps**, time t

(iv) reactivate (invoke Python iterator for) **Run()** function for event of time t in dictionary **events**

Again, what would appear to be an O(1) event is actually O(n).

Summary of Sources of SimPy Slowness

Summary of Sources of SimPy Slowness

- Dictionary (smaller problem).

Summary of Sources of SimPy Slowness

- Dictionary (smaller problem).
- O(n) insert operation instead of O(log n) (big problem).

Summary of Sources of SimPy Slowness

- Dictionary (smaller problem).
- O(n) insert operation instead of O(log n) (big problem).
- O(n) dequeue operation instead of O(1) (big problem).

Summary of Sources of SimPy Slowness

- Dictionary (smaller problem).
- O(n) insert operation instead of O(log n) (big problem).
- O(n) dequeue operation instead of O(1) (big problem).
- Possible VM issues.

Our Solutions

Our Solutions

- Remove dictionary entirely.

Our Solutions

- Remove dictionary entirely.
- Rewrite core event-list operations in C for speed.

Our Solutions

- Remove dictionary entirely.
- Rewrite core event-list operations in C for speed.
- SWIG forms the “glue.”

Our Solutions

- Remove dictionary entirely.
- Rewrite core event-list operations in C for speed.
- SWIG forms the “glue.”
- Rethink event-list algorithms.

Removal of Events Dictionary

Removal of Events Dictionary

- Incorporate into the **timestamps** list, so list elements are now of the form `(time, event)` instead of `(time)`.

Removal of Events Dictionary

- Incorporate into the **timestamps** list, so list elements are now of the form `(time, event)` instead of `(time)`.
- The **bisect()** operation still works!

Removal of Events Dictionary

- Incorporate into the **timestamps** list, so list elements are now of the form `(time, event)` instead of `(time)`.
- The **bisect()** operation still works!
- Needed to overload Python's $<$ operator.

Rewriting Event List Ops in C for Speed

Rewriting Event List Ops in C for Speed

- "Best of both worlds"—core runs in C, but apps programmer still writes in high-level Python.

Rewriting Event List Ops in C for Speed

- "Best of both worlds"—core runs in C, but apps programmer still writes in high-level Python.
- Used SWIG Python/C "glue" tool. (Available for Java etc. too.)

Rewriting Event List Ops in C for Speed

- "Best of both worlds"—core runs in C, but apps programmer still writes in high-level Python.
- Used SWIG Python/C "glue" tool. (Available for Java etc. too.)
- SWIG very easy to learn, use.

Rewriting Event List Ops in C for Speed

- "Best of both worlds"—core runs in C, but apps programmer still writes in high-level Python.
- Used SWIG Python/C "glue" tool. (Available for Java etc. too.)
- SWIG very easy to learn, use.
- We did have to be careful regarding reference counts.

Rethinking Event List Algorithms

Rethinking Event List Algorithms

- Lots of work in the past.

Rethinking Event List Algorithms

- Lots of work in the past.
- However, most algorithm-centric.

Rethinking Event List Algorithms

- Lots of work in the past.
- However, most algorithm-centric.
- Typically "simulations of simulation," not timing of actual programs.

Rethinking Event List Algorithms

- Lots of work in the past.
- However, most algorithm-centric.
- Typically “simulations of simulation,” not timing of actual programs.
- No consideration of systems issues, e.g. VM.

Empirical Evaluation

Tested many different modifications of SimPy

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)
- SimPy with dictionary removed, but still all-Python implementation (SimPyND)

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)
- SimPy with dictionary removed, but still all-Python implementation (SimPyND)
- SimPy with original event structures retained (though no dictionary) but operations implemented in C (PQArr)

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)
- SimPy with dictionary removed, but still all-Python implementation (SimPyND)
- SimPy with original event structures retained (though no dictionary) but operations implemented in C (PQArr)
- SimPy modified to use C-language calendar queue (CQ)

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)
- SimPy with dictionary removed, but still all-Python implementation (SimPyND)
- SimPy with original event structures retained (though no dictionary) but operations implemented in C (PQArr)
- SimPy modified to use C-language calendar queue (CQ)
- SimPy modified to use C-language splay tree (Splay)

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)
- SimPy with dictionary removed, but still all-Python implementation (SimPyND)
- SimPy with original event structures retained (though no dictionary) but operations implemented in C (PQArr)
- SimPy modified to use C-language calendar queue (CQ)
- SimPy modified to use C-language splay tree (Splay)
- Many others were tried but found to be noncompetitive.

Empirical Evaluation

Tested many different modifications of SimPy

- original SimPy (SimPy)
- SimPy with dictionary removed, but still all-Python implementation (SimPyND)
- SimPy with original event structures retained (though no dictionary) but operations implemented in C (PQArr)
- SimPy modified to use C-language calendar queue (CQ)
- SimPy modified to use C-language splay tree (Splay)
- Many others were tried but found to be noncompetitive.

Testbeds:

- Call center application. Indexed by arrival rates.
- Hold Model. Indexed by coeff. of var. of service times.

Results

Summary, from fastest to slowest:

Results

Summary, from fastest to slowest:

$CQ \approx$

Results

Summary, from fastest to slowest:
$CQ \approx PQArr >$

Results

Summary, from fastest to slowest:
$CQ \approx PQArr > SplayTree >$

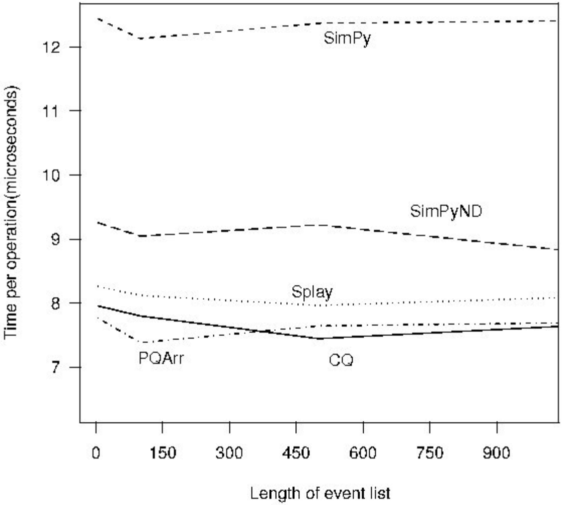
Results

Summary, from fastest to slowest:
$CQ \approx PQArr > SplayTree > SimPyND >$

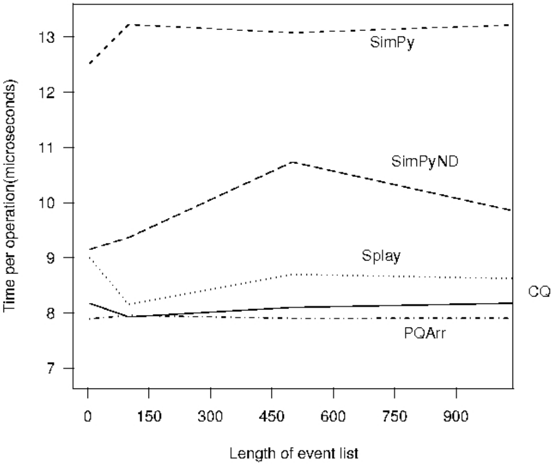
Results

Summary, from fastest to slowest:
$CQ \approx PQArr > SplayTree > SimPyND > SimPy$

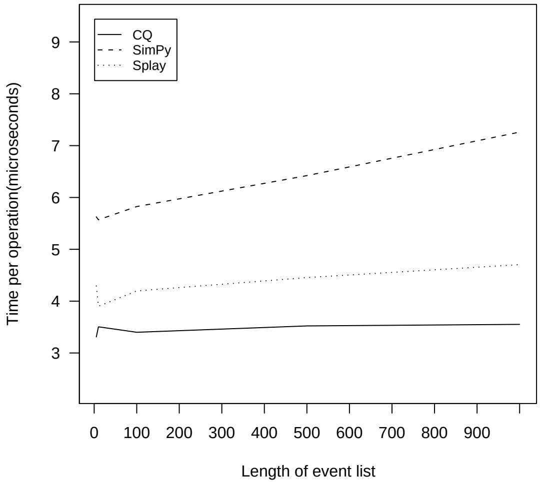
Call Center Times Per Op, Lower Traffic



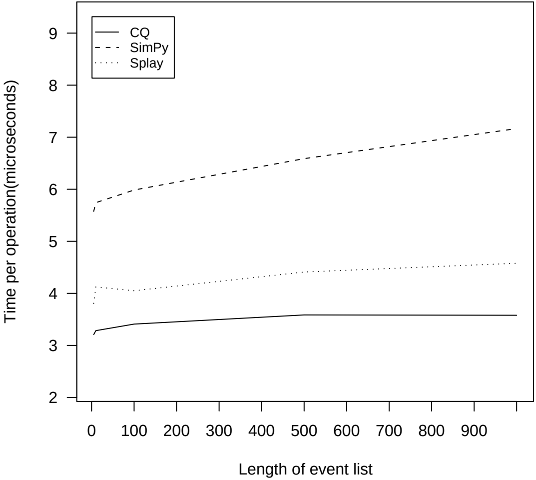
Call Center Times Per Op, Higher Traffic



Hold Model Times Per Op, Smaller COV



Hold Model Times Per Op, Larger COV



Scalability Issues

Scalability Issues

Even though CQ and PQArr were about equal in performance, PQArr appears not to scale well to larger event sets:

Scalability Issues

Even though CQ and PQArr were about equal in performance, PQArr appears not to scale well to larger event sets:

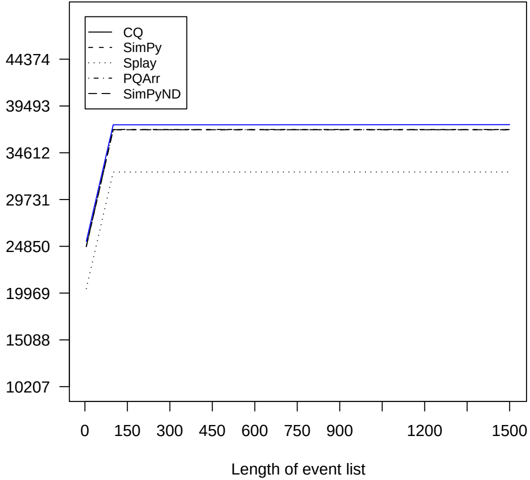
struct	user time	sys. time	event op. time
PQArr	79.47	4.50	57.87

Scalability Issues

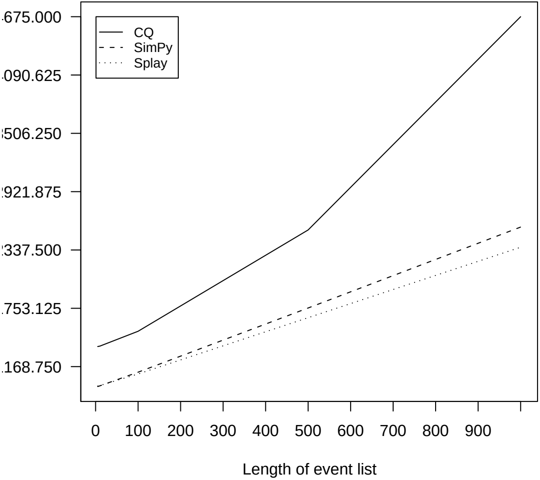
Even though CQ and PQArr were about equal in performance, PQArr appears not to scale well to larger event sets:

struct	user time	sys. time	event op. time
PQArr	79.47	4.50	57.87
CQ	33.24	3.95	12.69

Number of Page Faults, Call Center (lower traffic)



Number of Page Faults, Hold Model (medium COV)



Discussion of VM Issues

Discussion of VM Issues

- CQ paging performance poor in our experiments, run on 32-bit PCs running Linux kernel 2.6.20.

Discussion of VM Issues

- CQ paging performance poor in our experiments, run on 32-bit PCs running Linux kernel 2.6.20.
- Preliminary experiments on a 64-bit PC, same kernel, suggest greater variability.

Discussion of VM Issues

- CQ paging performance poor in our experiments, run on 32-bit PCs running Linux kernel 2.6.20.
- Preliminary experiments on a 64-bit PC, same kernel, suggest greater variability.
- $\therefore$ CQ may do poorly on some systems.

Conclusions and Discussion

Conclusions and Discussion

- Hybrid interpreted/C approach "best of both worlds"—transparent to apps programmer but with better performance

Conclusions and Discussion

- Hybrid interpreted/C approach "best of both worlds"—transparent to apps programmer but with better performance
- Attention to non-algorithmic issues, e.g. paging, may be worthwhile.

Conclusions and Discussion

- Hybrid interpreted/C approach “best of both worlds”—transparent to apps programmer but with better performance
- Attention to non-algorithmic issues, e.g. paging, may be worthwhile.
- What about JIT?

Conclusions and Discussion

- Hybrid interpreted/C approach "best of both worlds"—transparent to apps programmer but with better performance
- Attention to non-algorithmic issues, e.g. paging, may be worthwhile.
- What about JIT? Tried Pyscho but with disappointing results.