On the Dependability of Transient Neuron Simulations

Alexandros Mavrogiannis

July 9, 2014

Table of contents

Introduction

Dependability

The dependability of a system is the sum of the following attributes [2]:

- **Availability**, readiness for correct service
- **Reliability**, continuity of correct service
- **Safety**, absence of catastrophic consequences
- **Integrity**, absence of improper system alterations
- **Maintainability**, ability to undergo modifications

Dependability can be achieved through the following means:

- **Fault prevention**
- **Fault tolerance**
- **Fault removal**
- **Fault forecasting**

Checkpoint/Restart

Checkpoint/Restart (C/R) is a technique used to enable fault tolerance in a system.

- *Checkpoints* containing the state of the system are stored frequently in memory or in the filesystem.
- The *Restart* operation restores the system state from previous checkpoints.
- C/R can be applied at the system level or at the application level. [18]
- Coordination is required to guarantee consistency in distributed systems [20].
- Several automated C/R tools are available [1, 6, 3]

C/R Modelling and Optimization

C/R can be modelled as a sequences of cycles described by the following parameters:

τ	Checkpoint Interval
ℓ	Checkpoint Latency
R	Restart Time
T_r	Rollback Time

Optimization goal: select a checkpoint placement that minimizes the total time overhead [22, 5, 10, 15, 4, 11]

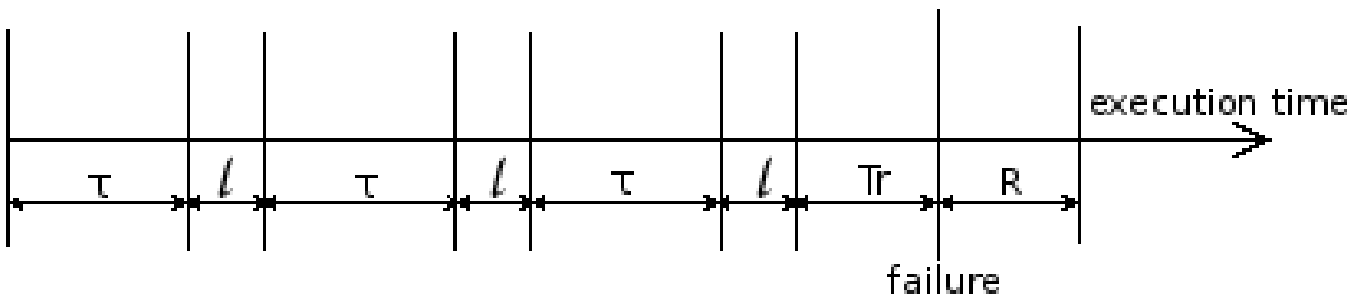


Figure: Time Modelling of the Checkpointing Cycle

Error Profiles

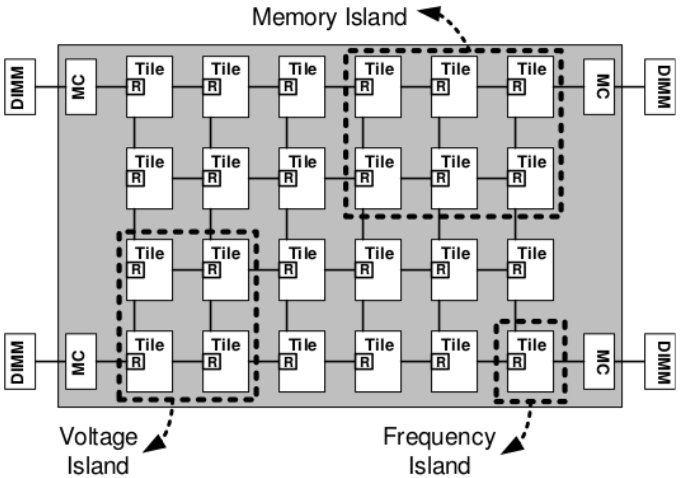
Two classes of errors will be considered [14]:

- DUE: Detected Unrecoverable Error
  - The application process is terminated
- SDC: Silent Data Corruptions
  - The application process continues operation
  - The corruption may not manifest as a failure
  - Occurences can be reliably detected only through redundancy

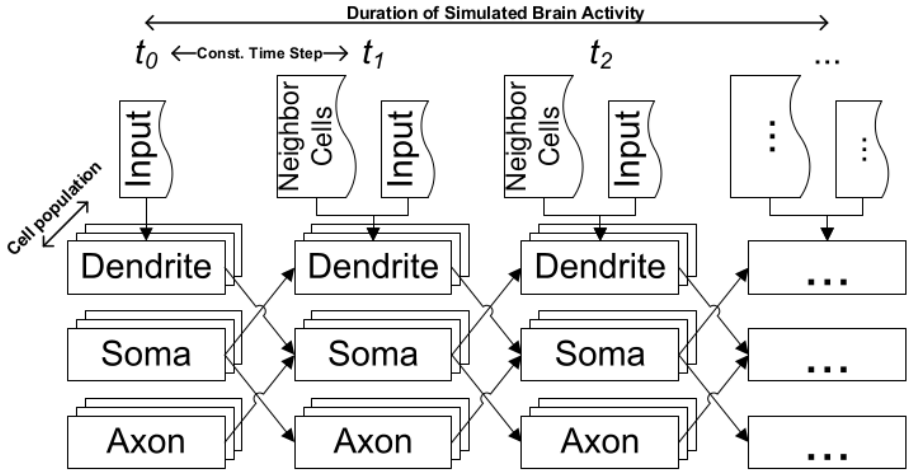
It is assumed that the failure rate of a system follows a stochastic process and it will be described by the Mean Time to Failure (MTTF) metric.

Experimental Setup

Target Platform and Application



(a) The frequency, voltage and memory islands of the SCC



(b) Illustration of the InfOli simulator runtime

The Single-Chip Cloud Computer (SCC) by Intel Labs. [8, 12]

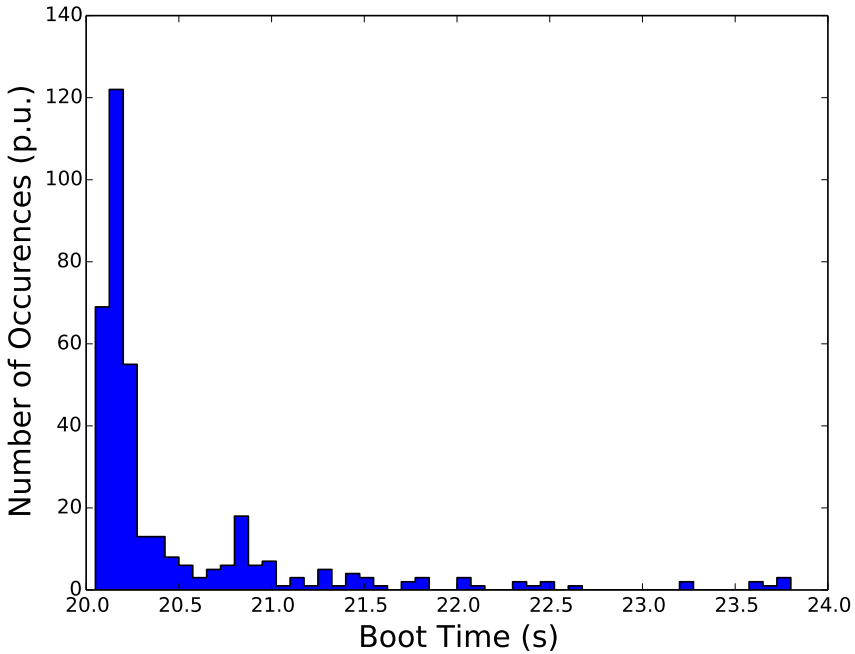
- Homogeneous chip of 48 cores.
- Management-Console Personal Computer (MCPC).
- `shared` directory between the MCPC and the SCC cores.
- Voltage and frequency islands.

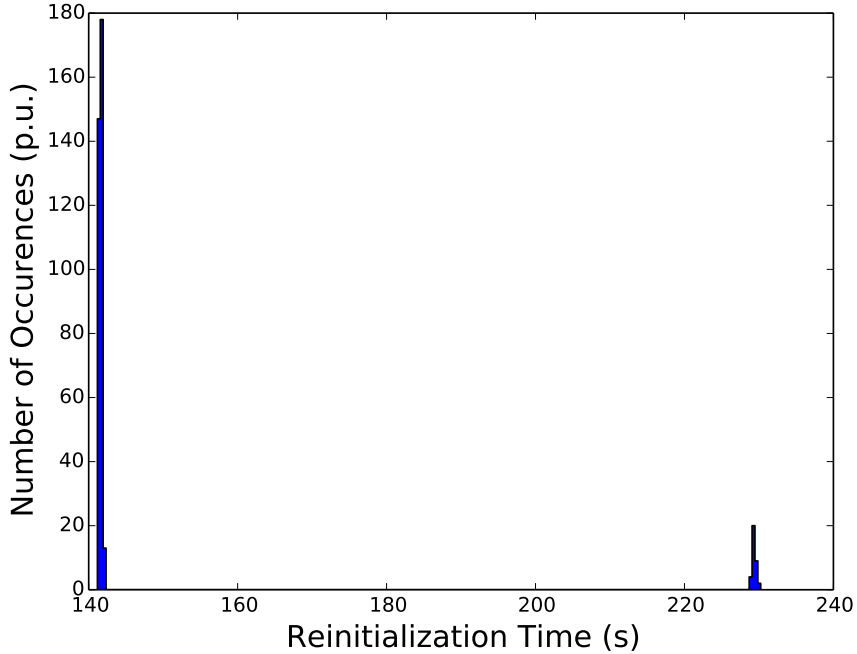
The InfOli simulator [16]

- Biologically plausible computational model of the Inferior Olivary nucleus.
- Based on the Hodgkin-Huxley model [7].
- Transient simulator with a step of $\Delta t = 50\mu s$.
- Implemented to achieve data parallelism.

Available Countermeasures

The `SCCKit` framework permits two actions that can lead to failure resolution:





- `sccBoot` redistributes the linux image and restarts cores.
- `sccBmc` reinitializes voltage and frequency of the SCC board

Checkpoint/Restart Implementation

A custom application-level C/R scheme was introduced.

- Periodic checkpoints of the state of each cell compartment are stored on the `shared` directory.
- Synchronization through a blocking barrier operation and consistency through double buffering of the checkpoint files.
- During the restart operation, each core reads the checkpoint files, restores the cells in memory and jumps to the appropriate simulation step.
- The implementation allows restarting on a different number of cores through reconstruction of the output files.

Interval and Latency

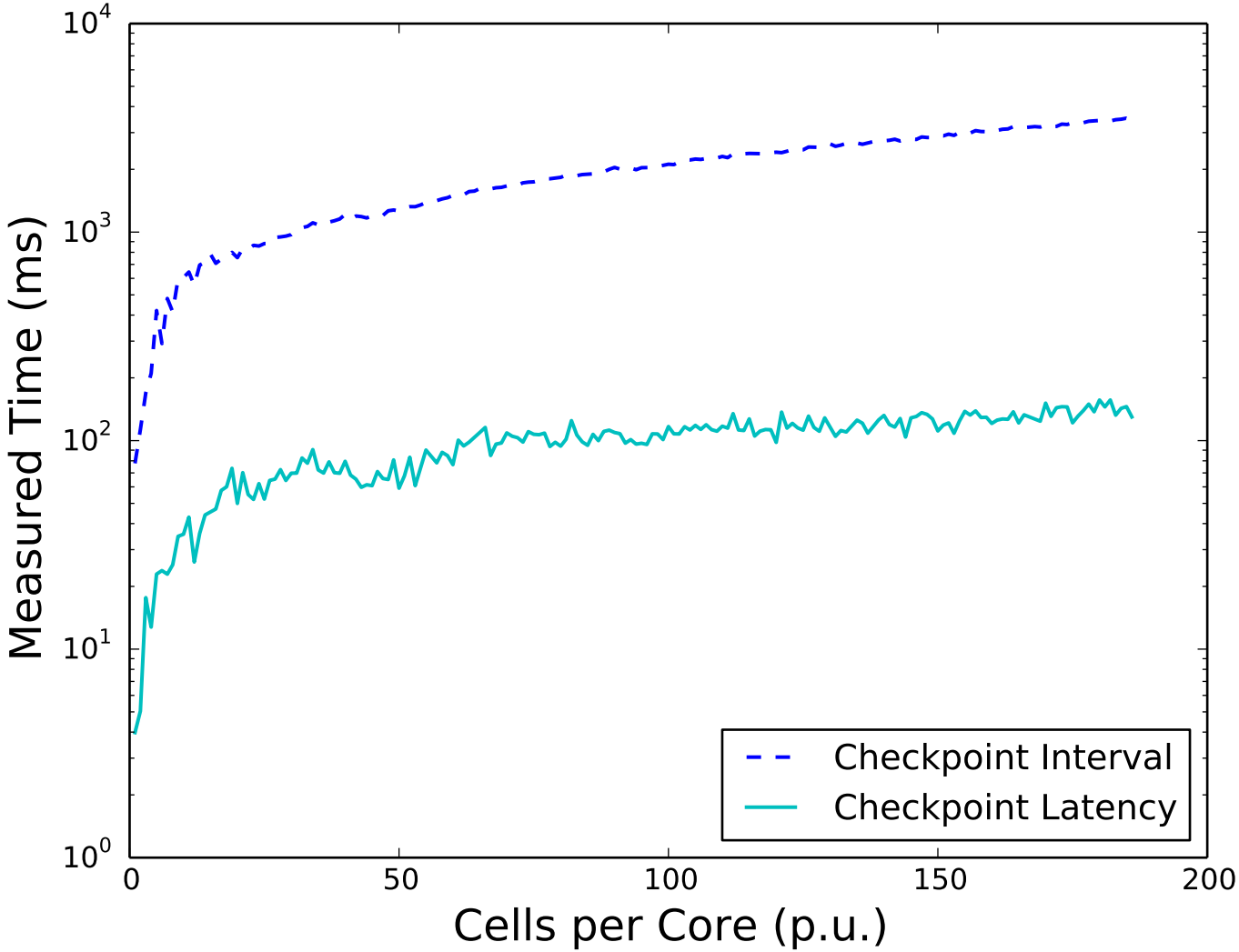


Figure: Checkpoint Interval and Latency of the developed C/R implementation for Intervals of 500 Simulation Steps

InfOli Restart

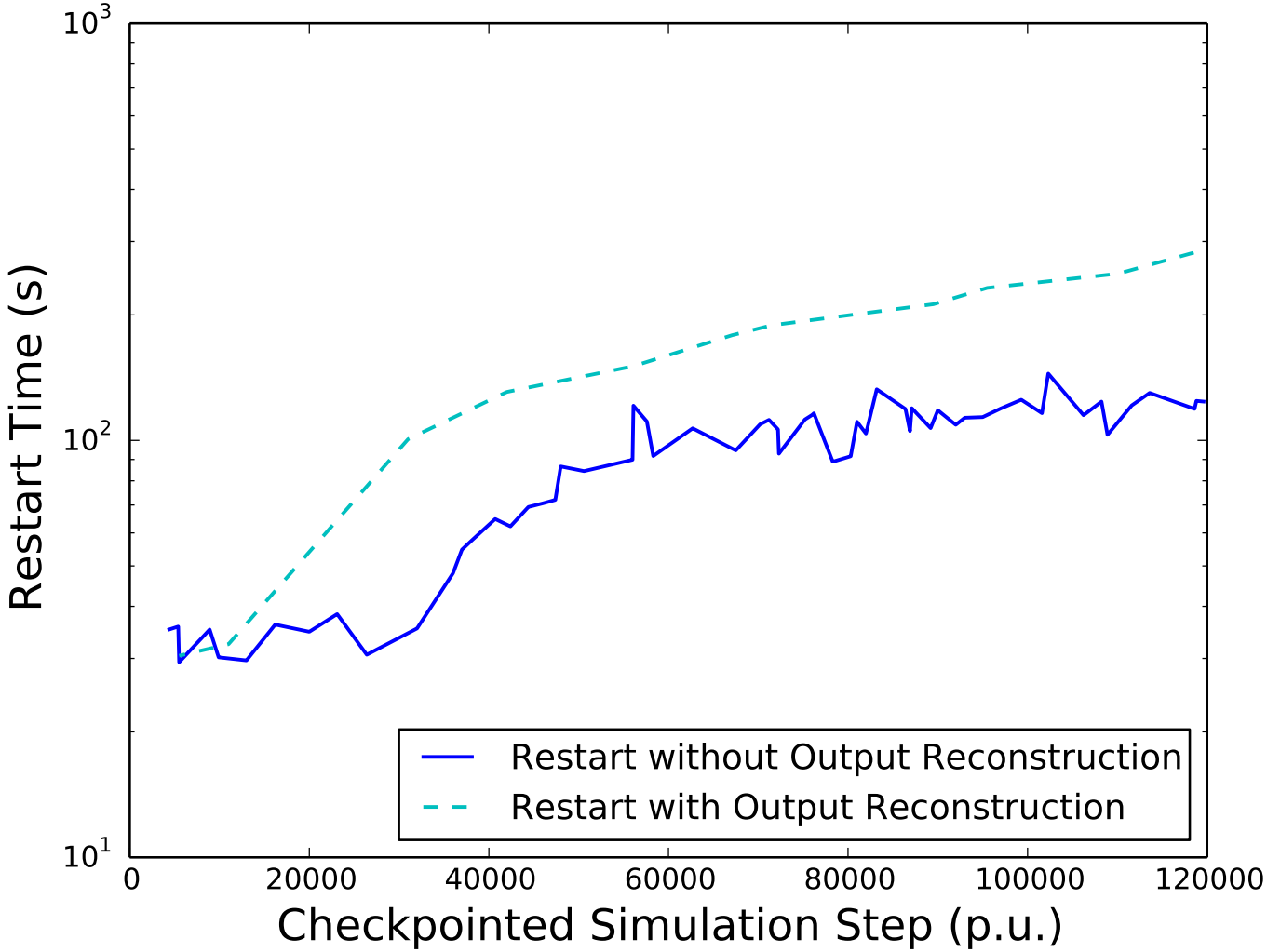


Figure: InfOli Restart Duration for 2400 Neuron Cells

Depman Tool

Depman Tool

The Depman tool (**Dep**endability **Man**ager) is a process on the MCPC that provides fault tolerance, fault removal and fault forecasting to the experimental setup.

- The runtime environment is monitored for DUE and SDC occurences.
- Countermeasures are performed in order, according to the detected error.
- The Time to Failure is measured and used to optimize the managed C/R implementation.
- The application is restarted in a thermal-aware fashion when permanent core failures occur.

Flow Diagram

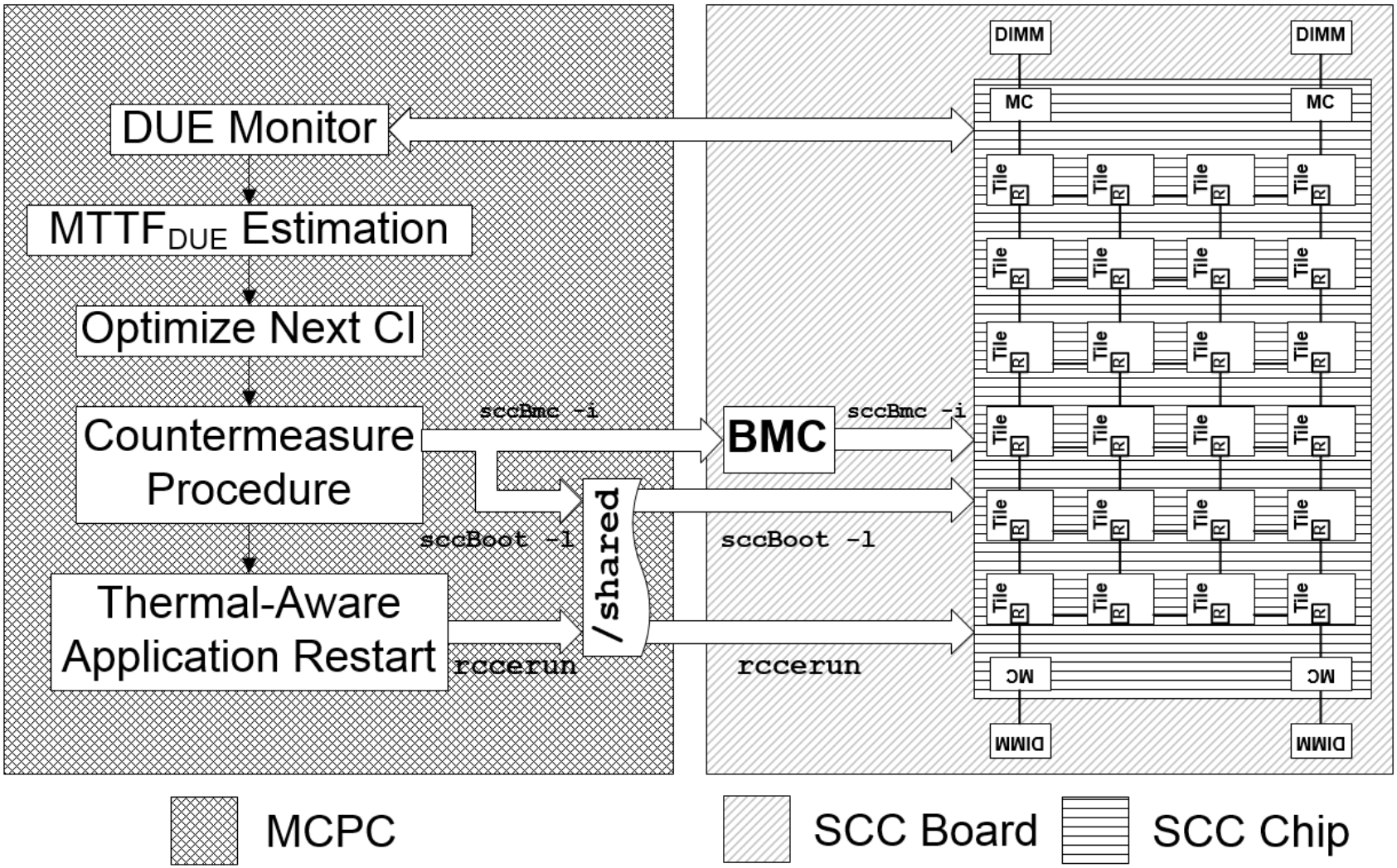


Figure: Flow Diagram of Depman's Closed Loop Operation for DUE Occurences

Block Diagram

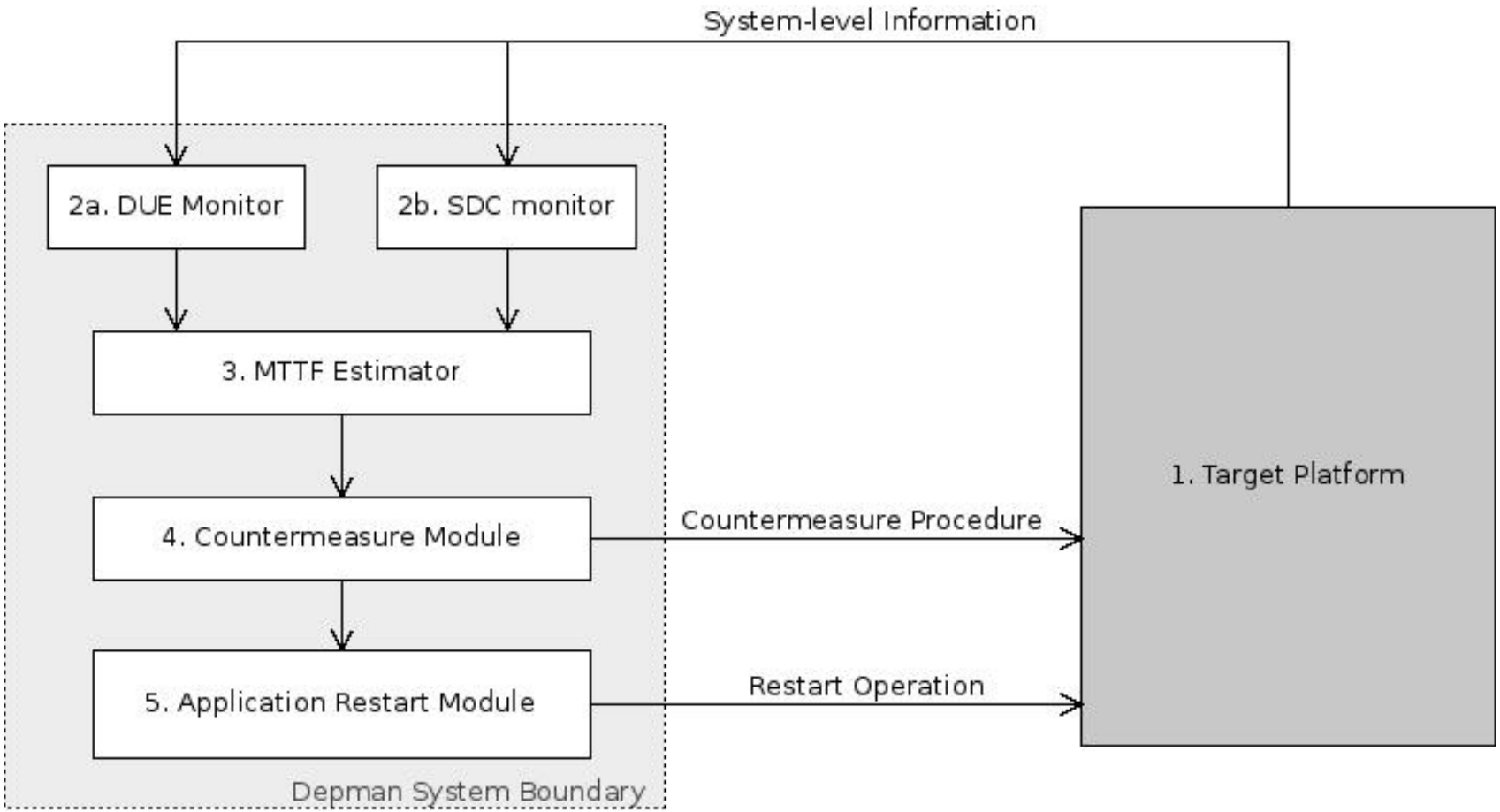
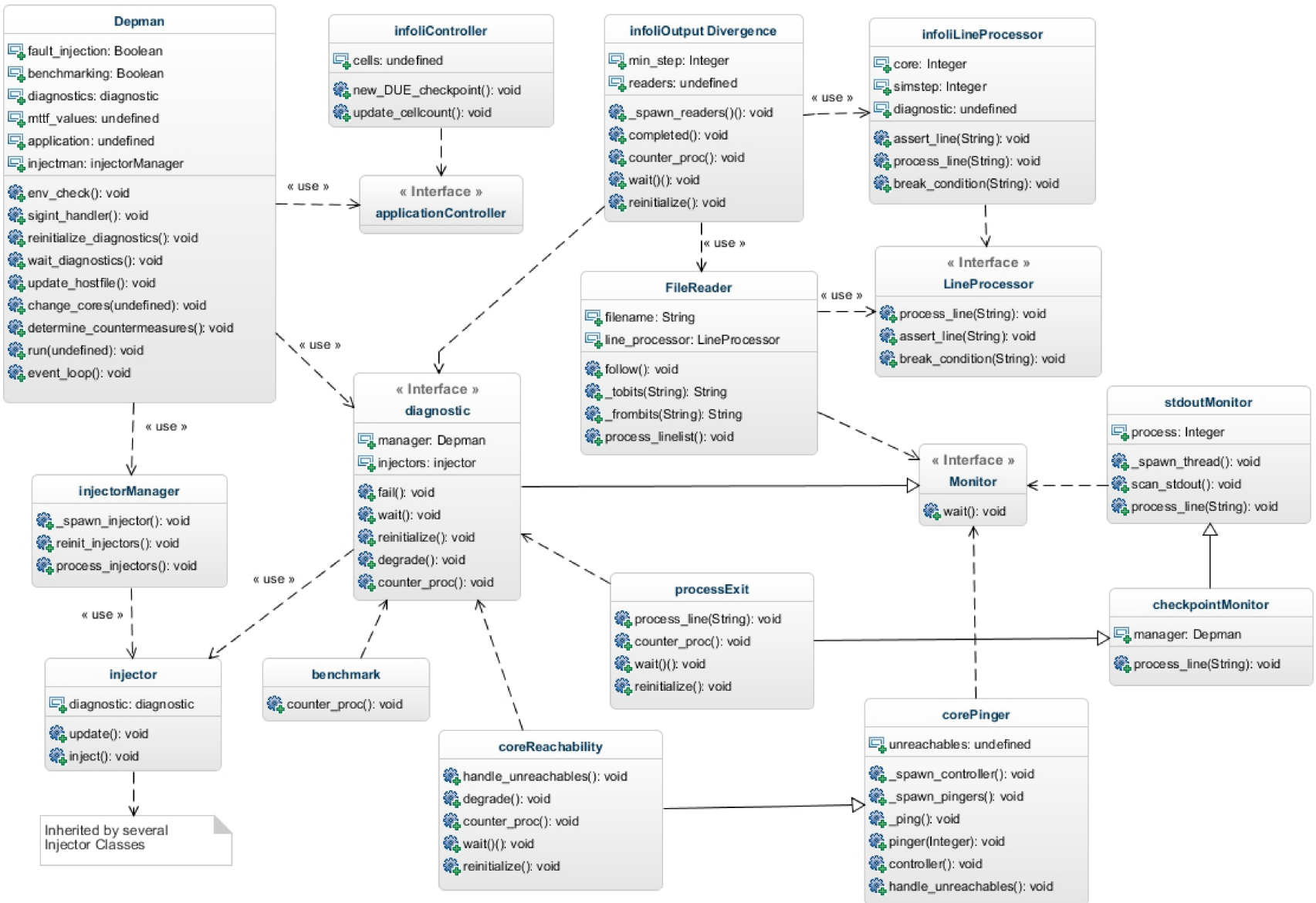


Figure: Block Diagram of the Depman tool

Class Diagram



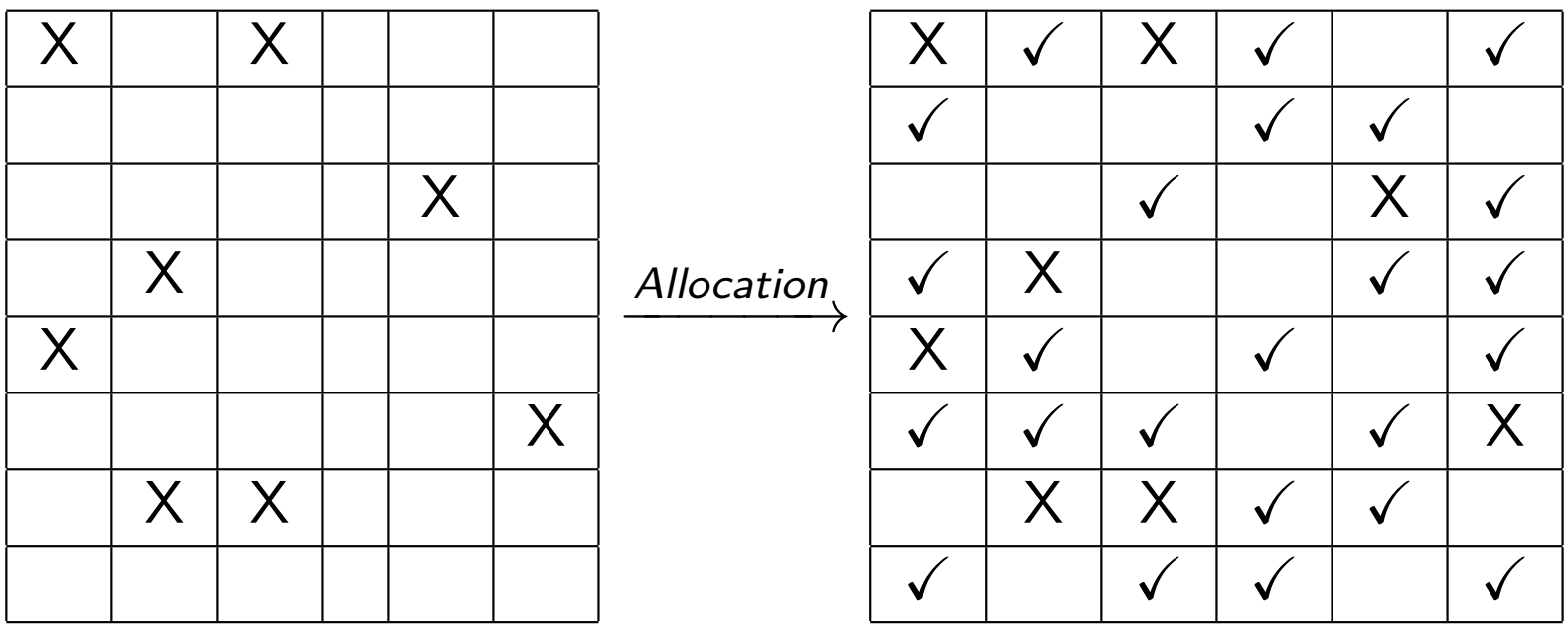
Diagnostics & Countermeasure Procedures

Diagnostic	Countermeasure Procedure
Process Exited	Restart Reboot, Restart Reinitialize, Reboot, Restart
Core Unreachable	Reboot, Restart Reinitialize, Reboot, Restart
Corrupted Infoli Ouput	Restart Reboot, Restart Reinitialize, Reboot, Restart

- Countermeasures are ordered by decreasing MTTR values in each procedure
- Countermeasure Procedures are attempted in an increasing MTTR order

Task Reallocation

- In the event of unreachable cores, if the countermeasure procedure is not successful then we can ignore the defunct cores.
- For the InfOli simulator, the total number of neuron cells needs to be a multiple of the number of operational cores.

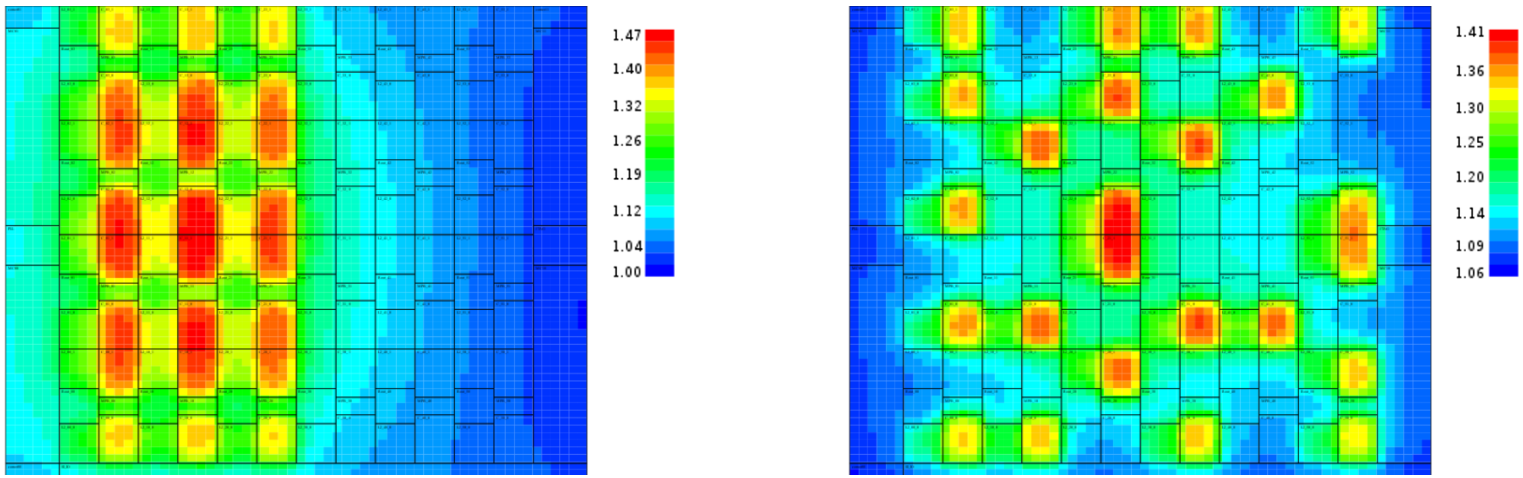


Task Reallocation Algorithm

```
1  def SCC_task_allocation(tasks, allowed_cores):
2      Grid = new Matrix[8][6]
3      for core not in allowed_cores:
4          Grid[core] = -1
5
6      x,y = random_available_edge_cell(Grid)
7
8      while tasks > 0:
9          Grid[x][y] = -2
10         tasks--
11         for i,j in (8,6):
12             if Grid[i][j] > 0:
13                 r = manhattan_distance((x,y), (i,j))
14                 if r < Grid[i][j]:
15                     Grid[i][j] = r
16                 else if (r - 1 < Grid[i][j] < r):
17                     Grid[i][j] -= 0.01
18
19         max_list = get_all_max_values(Grid)
20         min_list = get_all_minimum_distances_to_edge(max_list)
21         x, y = select_random(min_list)
22
23     return core_names_from_coordinates(Grid.find(-2))
```

Figure: Pseudocode of the Thermal-Aware Task Allocation Algorithm

Hotspot Results



(a) Naive Task Allocation (b) Thermal-Aware Task Allocation

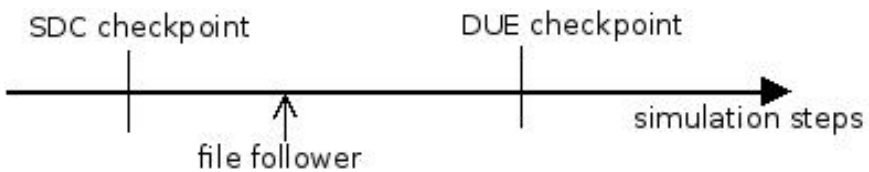
Figure: Temperature Modeling of 24 Running Cores on the SCC [19, 17]

It can be observed that in the thermal-aware model:

- The maximum temperature was reduced
- The temperature gradients are less steep

SDC & DUE checkpoints

- Problem: An SDC may be detected after an unsafe checkpoint is taken
- Solution: Store multiple checkpoints, one of which must be located before the SDC detector
  - On SDC detection, the closest checkpoint to the SDC detector is used and the others are discarded.
  - On DUE detection, the last available checkpoint is used and the SDC detector resumes operation from its previous location.



Checkpoint Interval Optimization

Injection Campaign

- Tests were performed by injecting failures through a Monte Carlo simulation at predetermined $\mathrm{MTTF_s}$ vectors.
- The probability of error occurence at an interval Δt is:

$$P_{\mathrm{s}} = 1 - e^{-\Delta t/\mathrm{MTTF_s}} \tag{1}$$

Checkpoint Interval Optimization

- The use of C/R introduces a time overhead called the *Waste Time* (W)
- The total waste time is the sum of the waste times of each cycle (W_i)
- In periodic C/R implementations, the waste time depends on the checkpoint interval τ.
- Goal: minimize the waste time by selecting an appropriate checkpoint interval τ_{opt}
- This selection requires knowledge of the failure rate of the system

Periodic checkpointing is only optimal for failure rates that follow Poisson processes [9].

Optimal Checkpoint Interval

Assuming that k_i checkpoints will be taken on the i-th cycle:

$$W_i = k_i \times \ell + T_r \tag{2}$$

$$T_r = \frac{\tau + \ell}{2} \tag{3}$$

$$k_i = \frac{\mathrm{MTTF}_i}{\tau + \ell} \tag{4}$$

$$W_i = \frac{\mathrm{MTTF}_i \times \ell}{\tau + \ell} + \frac{\tau + \ell}{2} \tag{5}$$

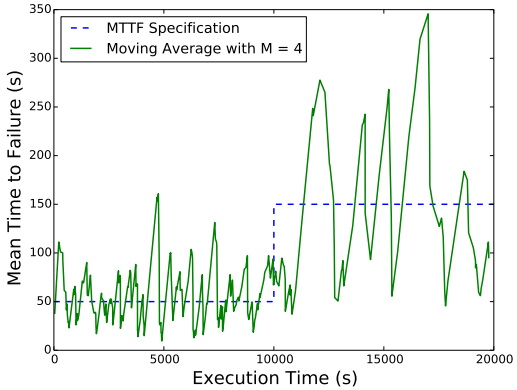
$$\tau_{opt} = \sqrt{2 \times \mathrm{MTTF}_i \times \ell} - \ell \tag{6}$$

Moving Average Estimation

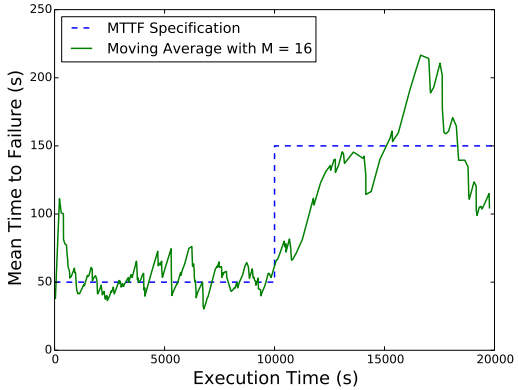
$MTTF_i$ is calculated by using a moving average filter. The filter uses the failure occurence times of the last M checkpointing cycles to provide an estimate for $MTTF_i$, $\mathrm{MTTF}_{\mathrm{DUE}}$

$$\mathrm{MTTF}_{\mathrm{DUE}} = \sum_{i=0}^{M-1} \frac{TTF_i}{M} \tag{7}$$

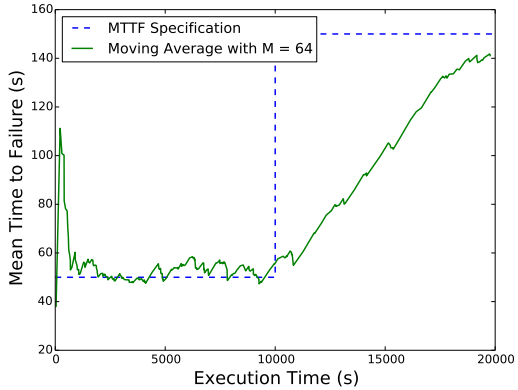
Step Response of the Moving Average Filter



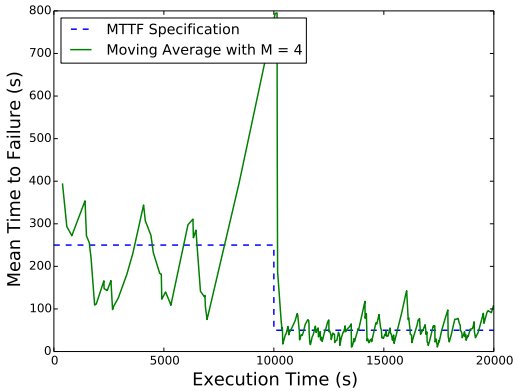
(a) $M = 4$



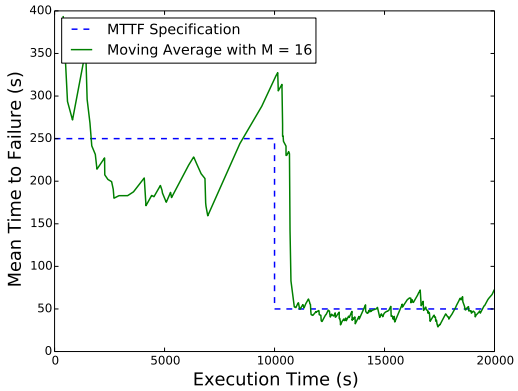
(b) $M = 16$



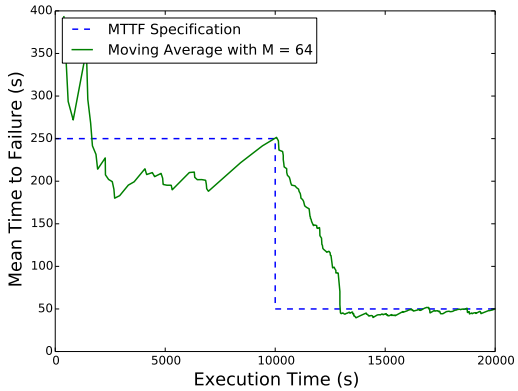
(c) $M = 64$



(d) $M = 4$



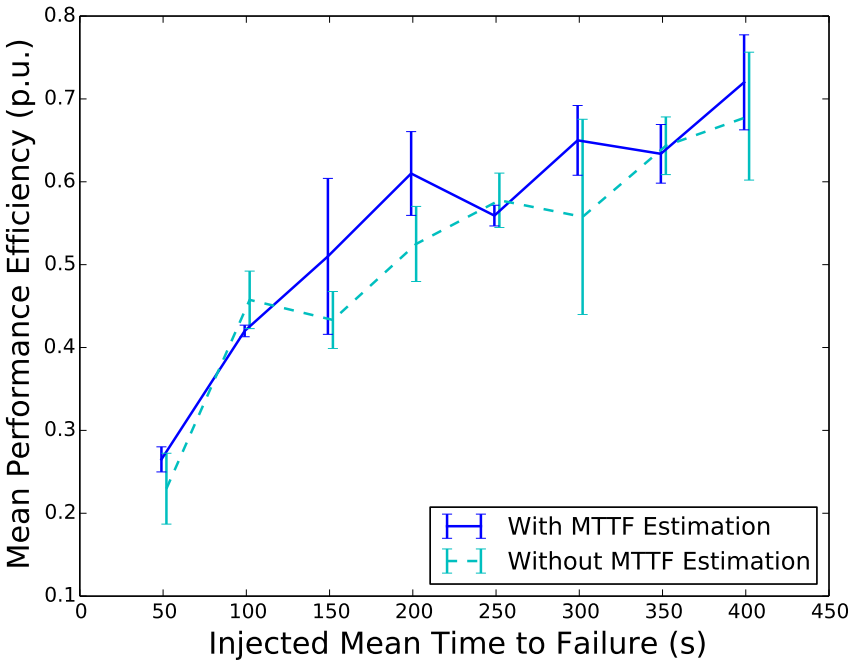
(e) $M = 16$



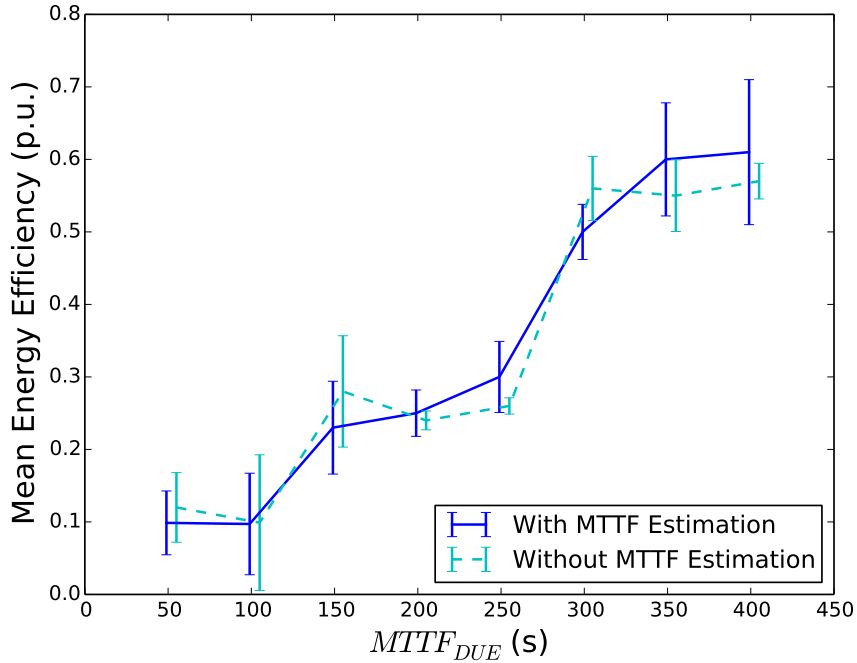
(f) $M = 64$

Figure: Rising and Falling Step Response of the Moving Average Filter for Various Filter Lengths M

Efficiency in Constant MTTF Scenarios



(a)

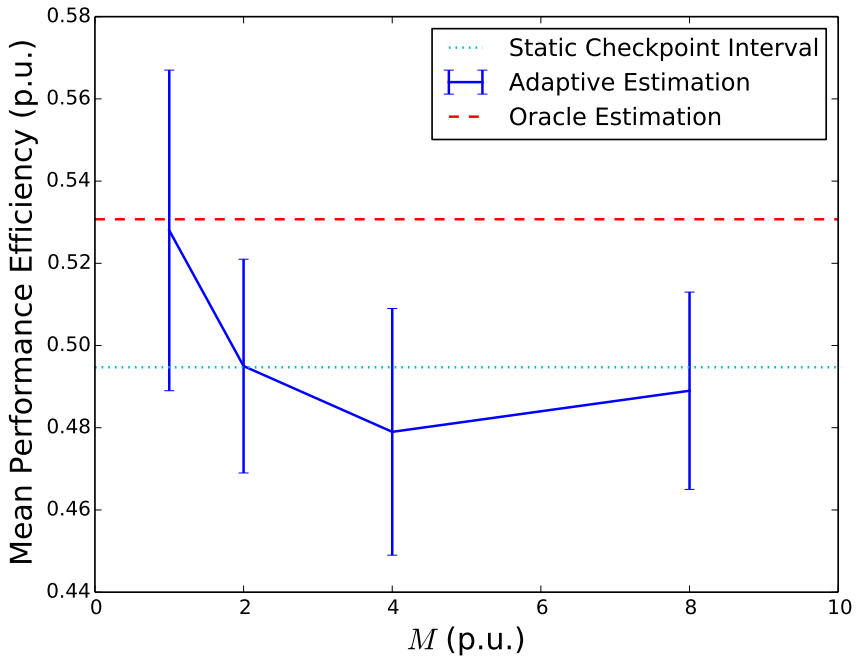


(b)

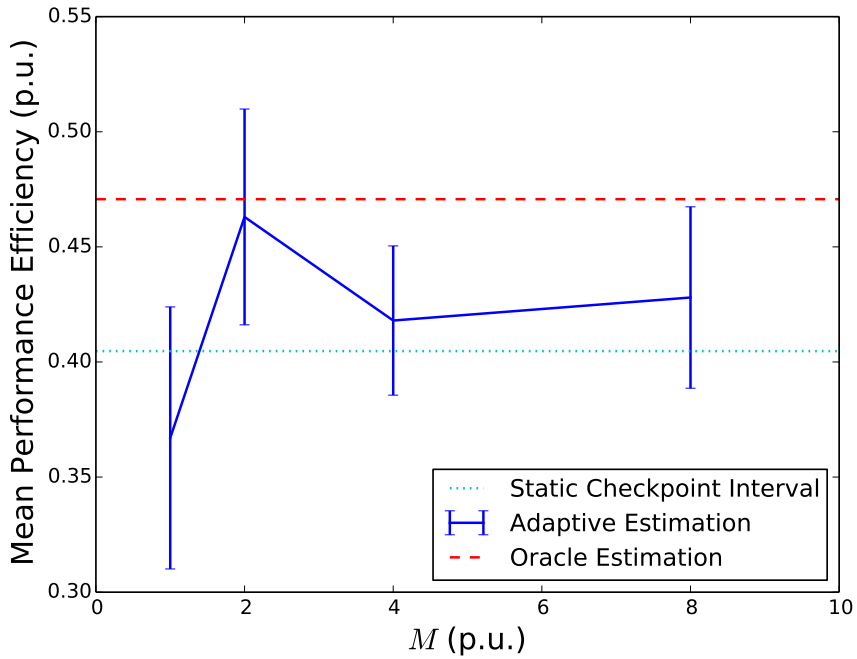
Figure: Performance and Energy Efficiency for a Constant Injection Rate

The error bars represent a 95% confidence interval [13].

Performance Efficiency in Time-Varying MTTF Scenarios



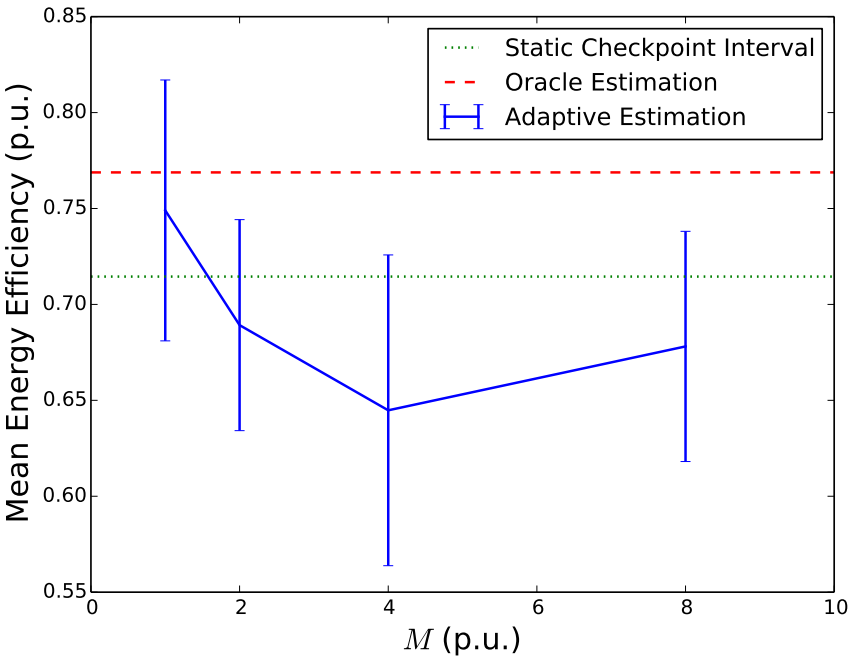
(a) Rising MTTF edge



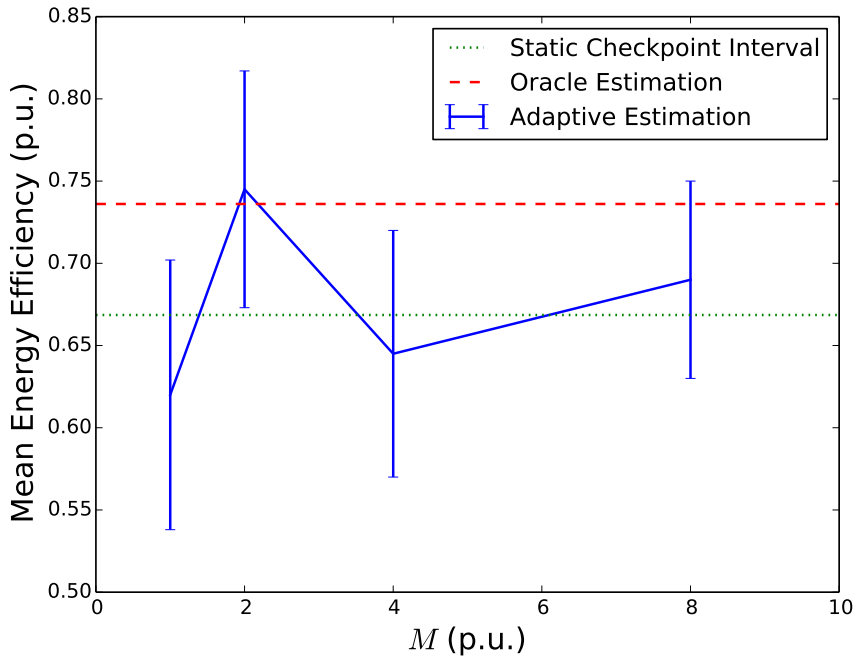
(b) Falling MTTF edge

Figure: Performance Efficiency of `Depman` for Two Cases of Time-Varying MTTF

Energy Efficiency in Time-Varying MTTF Scenarios



(a) Rising MTTF edge



(b) Falling MTTF edge

Figure: Energy Efficiency of `Depman` for Two Cases of Time-Varying MTTF

Conclusions and Future Work

Conclusions

- The `Depman` tool operated according to its specifications
- The on-the-fly estimation does not introduce measurable efficiency loss in constant failure rate scenarios
- On-the-fly CI optimization shows efficiency improvements, as demonstrated by the oracle estimator (upper bound)
- The optimization scheme shows efficiency benefits in falling MTTF steps where the estimator converges faster to the new MTTF value
- The resulting work was submitted for peer review to the HiPEAC 2015 conference

Future Work

- Extension of the Depman tool for operation on other platforms and other applications.
- Restatement of the τ optimization problem as a state-space control problem, utilizing state observers for the non-observable MTTF variable [21].
- Utilization of fault tolerance techniques, such as C/R, for the machine hosting the Depman tool.
- Expansion of the Depman tool for concurrent dependability management of multiple platforms.
- Application of the discussed closed-loop optimization techniques for redundancy optimization.

References

References I

[1] Ansel, J., Arya, K., and Cooperman, G.
Dmtcp: Transparent checkpointing for cluster computations and the desktop.
In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* (2009), IEEE, pp. 1–12.

[2] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C.
Basic concepts and taxonomy of dependable and secure computing.
Dependable and Secure Computing, IEEE Transactions on 1, 1 (2004), 11–33.

[3] Bronevetsky, G., Marques, D., Pingali, K., and Stodghill, P.
C 3: A system for automating application-level checkpointing of mpi programs.
In *Languages and Compilers for Parallel Computing*. Springer, 2004, pp. 357–373.

[4] Daly, J. T.
A higher order estimate of the optimum checkpoint interval for restart dumps.
Future Generation Computer Systems 22, 3 (2006), 303–312.

[5] Geist, R., Reynolds, R., and Westall, J.
Selection of a checkpoint interval in a critical-task environment.
Reliability, IEEE Transactions on 37, 4 (1988), 395–400.

[6] Hargrove, P. H., and Duell, J. C.
Berkeley lab checkpoint/restart (blcr) for linux clusters.
In *Journal of Physics: Conference Series* (2006), vol. 46, IOP Publishing, p. 494.

[7] Hodgkin, A. L., and Huxley, A. F.
A quantitative description of membrane current and its application to conduction and excitation in nerve.
The Journal of physiology 117, 4 (1952), 500.

[8] Howard, J. et al.
A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling.
IEEE JSSC 46, 1 (2011), 173–183.

References II

[9] Ling, Y., Mi, J., and Lin, X.
A variational calculus approach to optimal checkpoint placement.
Computers, IEEE Transactions on 50, 7 (2001), 699–708.

[10] Liu, Y., Nassar, R., Leangsuksun, C., Naksinehaboon, N., Paun, M., and Scott, S. L.
An optimal checkpoint/restart model for a large scale high performance computing system.
In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (2008), IEEE, pp. 1–9.

[11] Lu, G., Zheng, Z., and Chien, A. A.
When is multi-version checkpointing needed?
In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale* (2013), ACM, pp. 49–56.

[12] Mattson, T. G., Riepen, M., Lehnig, T., Brett, P., Haas, W., Kennedy, P., Howard, J., Vangal, S., Borkar, N., Ruhl, G., et al.
The 48-core scc processor: the programmer's view.
In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (2010), IEEE Computer Society, pp. 1–11.

[13] McConaghy, T., Breen, K., Dyck, J., and Gupta, A.
Variation-Aware Design of Custom Integrated Circuits: A Hands-On Field Guide.
Springer, 2013.

[14] Mukherjee, S.
Architecture design for soft errors.
Morgan Kaufmann, 2011.

[15] Oliner, A. J., Rudolph, L., and Sahoo, R. K.
Cooperative checkpointing: a robust approach to large-scale systems reliability.
In *Proceedings of the 20th annual international conference on Supercomputing* (2006), ACM, pp. 14–23.

References III

[16] Rodopoulos, D., Chatzikonstantis, G., Padelopoulos, A., Soudris, D., Zeeuw, C. I. D., and Strydis, C.
Optimal mapping of inferior olive neuron simulations on the single-chip cloud computer.
In *Embeded Computer Systems (SAMOS) 2014 International Conference on* (2014).

[17] Sadri, M., Bartolini, A., and Benini, L.
Single-chip cloud computer thermal model.
In *Thermal Investigations of ICs and Systems (THERMINIC), 2011 17th International Workshop on* (Sept 2011), pp. 1–6.

[18] Silva, L. M., and Silva, J. G.
System-level versus user-defined checkpointing.
In *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on* (1998), IEEE, pp. 68–74.

[19] Skadron, K., Stan, M. R., Sankaranarayanan, K., Huang, W., Velusamy, S., and Tarjan, D.
Temperature-aware microarchitecture: Modeling and implementation.
ACM Trans. Archit. Code Optim. 1, 1 (Mar. 2004), 94–125.

[20] Walters, J. P., and Chaudhary, V.
Application-level checkpointing techniques for parallel programs.
In *Distributed Computing and Internet Technology*. Springer, 2006, pp. 221–234.

[21] Wang, W., and Gao, Z.
A comparison study of advanced state observer design techniques.
In *American Control Conference, 2003. Proceedings of the 2003* (2003), vol. 6, IEEE, pp. 4754–4759.

[22] Young, J. W.
A first order approximation to the optimum checkpoint interval.
Communications of the ACM 17, 9 (1974), 530–531.