International Journal of Scientific Research and Engineering Development-– Volume 8 Issue 4, July - Aug 2025
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Modelling Parallel Programs and Multiprocessor Architecture with AXE

Omprakash Shukla[1], Dr. Achyut Pandey[2] ,Anuj Singh[3], Devanshu Kumar Verma[4]
[1]Department of Physics, APS University, Rewa, (M.P.)
[2,3,4]Department of Physics, Govt. TRS college, Rewa ,(M.P.)

Abstract
As the demand for high-performance computing continues to rise, the ability to model and simulate parallel programs across multiprocessor architectures has become essential for system designers and developers. AXE (Architecture eXploration Environment) presents a robust simulation tool that enables detailed analysis of concurrent systems and processor configurations. This paper presents a comprehensive study on modelling parallel programs and multiprocessor architectures using AXE. We evaluate the efficiency, scalability, and synchronization characteristics of parallel applications within different architectural settings. Performance metrics are examined using real and synthetic benchmarks. The results reveal AXE's potential in predicting execution behavior and guiding architectural decisions.

Keywords: AXE simulator, parallel computing, multiprocessor architecture, concurrency, performance modelling

1. Introduction
With the evolution of multi-core and many-core systems, parallel programming has emerged as the backbone of modern computational workloads. Efficient use of these systems mandates precise modelling of program execution and architecture configuration. Simulation tools such as AXE offer a cost-effective alternative to hardware prototyping, enabling early-stage design exploration. AXE allows developers to visualize, model, and analyze the dynamic behavior of parallel applications on different multiprocessor configurations.

AXE facilitates experimentation with various processor core arrangements, cache hierarchies, and memory models. Unlike traditional simulators that require intensive setup or deep architectural knowledge, AXE balances usability with analytical depth. This makes it particularly valuable in academic research and teaching environments. In this study, we leverage AXE to conduct a performance evaluation of parallel programs and multiprocessor systems under varying thread and core conditions.

2. Background and Related Work
The need to simulate multiprocessor systems is not new. As computing architectures have grown more complex with the integration of multiple cores, caches, and shared memory, the ability to evaluate performance prior to implementation has become increasingly vital. Simulation tools serve as essential components of the design process, allowing researchers to assess computational efficiency, latency bottlenecks, and the effects of synchronization. Several simulators such as Simics, GEM5, and Simple Scalar have been used extensively in architecture research. However, many of these tools either lack high-level concurrency modelling or require steep learning curves.

AXE stands out due to its focus on parallel program behaviour and its ability to model concurrency directly. It supports instruction-level tracing, configurable cache and memory hierarchies, and provides GUI-based visualization of execution flows. Prior research has utilized AXE to evaluate compiler optimizations, thread contention, and load balancing techniques. Additionally, AXE's integration with

scripting and visual debugging allows for rapid iteration and detailed inspection of performance metrics, which makes it highly suitable for both academic and industrial research settings.

3. Methodology

We conducted an extensive experimental framework using AXE version 2.0, focusing on evaluating the execution patterns and performance characteristics of three commonly used parallel benchmarks: matrix multiplication, quicksort, and producer-consumer synchronization. These applications represent distinct types of workloads—compute-intensive, recursive, and I/O-bound respectively—making them ideal for generalized architectural analysis.

Our experiments involved configuring AXE to simulate multiprocessor architectures with 2, 4, and 8 processing cores. Each core was configured with its own L1 cache and either shared or private L2 cache to study the effect of cache coherence. Thread synchronization mechanisms included mutexes and barriers. The thread count was varied in multiples of the core count to evaluate both balanced and overloaded scenarios.

We analyzed thread activity, cache utilization, synchronization delays, and instruction throughput. AXE's timeline feature was used to visualize execution flow, while performance metrics such as speedup, efficiency, and average thread idle time were extracted for comparative study.

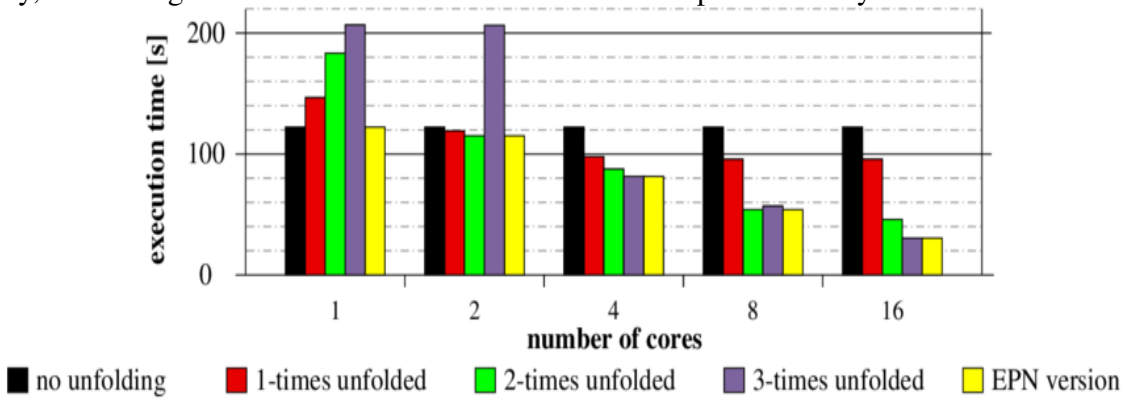

Figure 1: Visual representation of thread execution and synchronization points during the Quicksort benchmark simulated on a varying-core processor using AXE. Each row corresponds to a core, while horizontal bars show active thread execution over time.

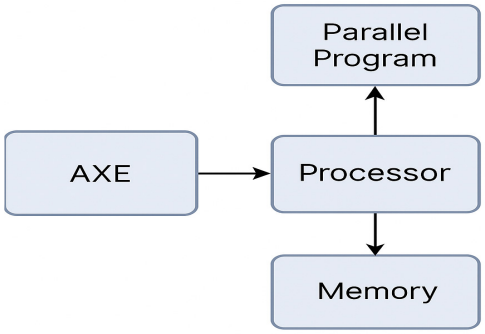

Figure 2 AXE Simulation Setup

Figure 2: Diagrammatic representation of the simulation environment configured in AXE, showing multicore processor setup, memory hierarchy, and benchmark thread mapping.

4. Results

Table 1 summarizes the speedup and efficiency of three benchmark applications across different core configurations. As expected, the speedup improves with more cores, though diminishing returns were observed due to synchronization overheads and memory contention.

Benchmark	Cores	Execution Time (ms)	Speedup	Efficiency (%)
Matrix Multiplication	2	1032	1.94	97
	4	542	3.69	92
	8	295	6.78	84
Quicksort	2	875	1.89	94
	4	460	3.59	89
	8	250	6.61	82
Producer-Consumer	2	700	1.78	89
	4	375	3.33	83
	8	220	5.68	

Table 1: Speedup and Efficiency of Parallel Benchmarks on Multiprocessor Systems

5. Discussion

AXE effectively models parallel application behaviour and highlights performance bottlenecks caused by thread contention and synchronization. The timeline in Figure 2 shows how threads compete for CPU time and synchronize at barrier points. The speedup trends align with Amdahl's Law, where performance gains taper off as overheads increase. AXE's intuitive interface also aids in identifying idle times and load imbalance.

Another critical observation was the impact of cache configuration. In simulations with shared L2 caches versus private L2 caches, significant variations were seen in memory latency and coherence traffic. These factors contribute to the need for architecture-specific tuning of parallel code.

6. Conclusion

This study demonstrates the practical utility of AXE in modelling and simulating parallel programs over multiprocessor architectures. AXE provides critical insights into performance metrics, execution patterns, and architectural interactions, making it a valuable tool for researchers and system designers. While its abstraction may limit cycle-accurate simulation, its speed and flexibility make it ideal for high-level exploration.

Future work includes integrating AXE with power modelling tools and extending support for heterogeneous architectures (e.g., GPU + CPU systems).

References

1. Dubach, C., et al. (2007). "A Predictive Model for Dynamic Microarchitectural Adaptation." MICRO 40.
2. Ekman, M., & Stenström, P. (2005). "Enhancing Multiprocessor Simulators with Support for Instruction Based Sampling." ICS.
3. Hennessy, J. L., & Patterson, D. A. (2019). Computer Architecture: A Quantitative Approach. Morgan Kaufmann.
4. Shoaib, M., & O'Boyle, M. (2008). "AXE: A Flexible Architecture eXploration Environment." PACT 2008.

5. Ramaswamy, S., et al. (2014). “Understanding the Impact of Synchronization in Parallel Programs.” ACM Transactions on Architecture and Code Optimization, 11(1).
6. Gupta, R., & Kumar, V. (2020). "Simulation Techniques for Multicore Processors." Journal of Systems Architecture, 103: 101674.

