

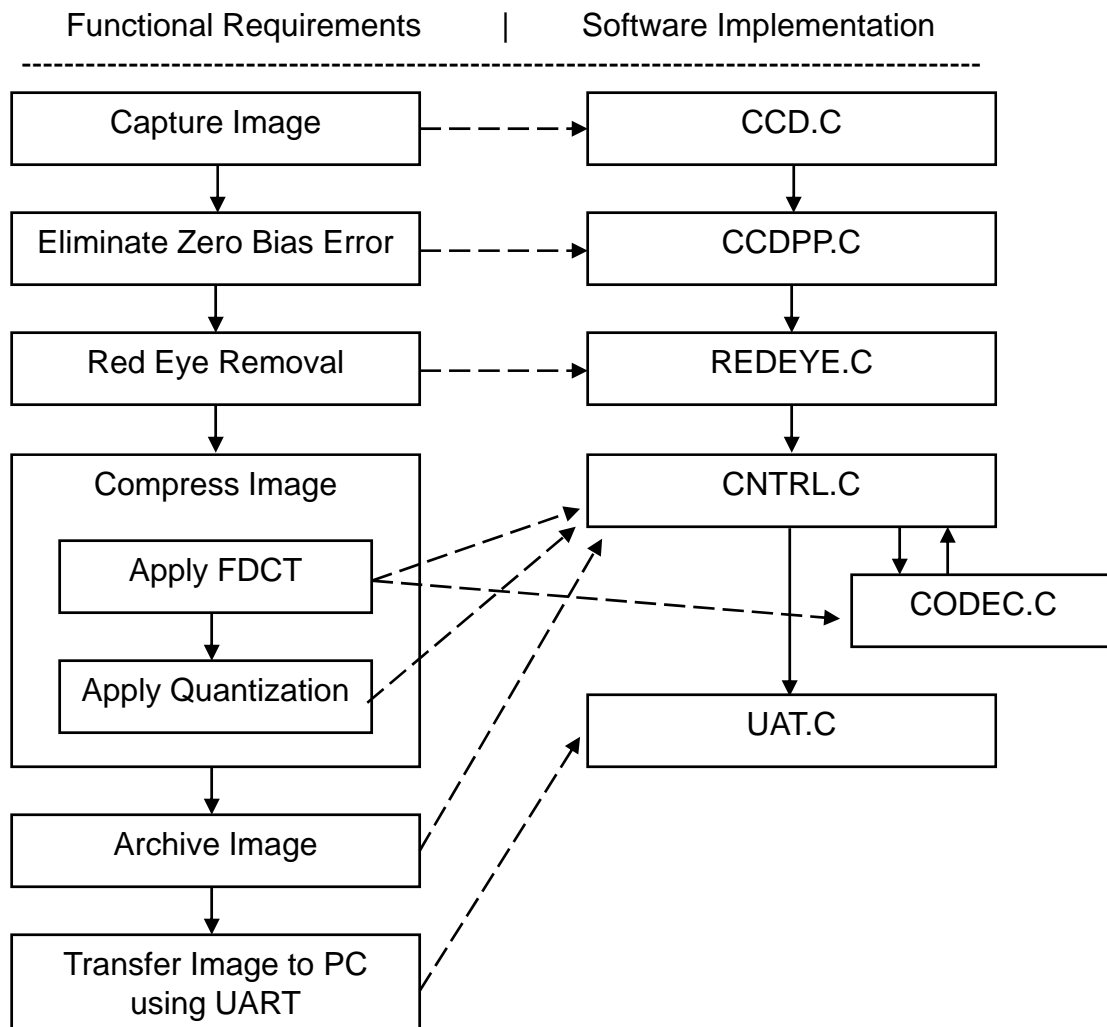
Software Decomposition for Multicore Architectures

Ankit Jain, Dr. Ravi Shankar

Multicore Architectures: Software Reuse Challenge

- Reverse Engineer existing (legacy) software to port it on next generation Multicore Architectures
- Partition existing code over Multiple concurrent cores
- Transform existing Sequential Model based software to Concurrent Model based software

Benchmark Chosen



The digital camera software (DCS) was chosen from all the available options mainly due to its application to Multicore Architectures and easy availability of software for it. Digital cameras have grown in popularity and are being integrated with several consumer electronic products such as phones, portable entertainment devices (game consoles), and watches. This requires DCS to be scalable so that it can be used in multiple domains. The implementation was not detailed; yet the implementation covers all the functional requirements of a basic digital camera by making certain assumptions.

Top-Down Representation

Control/Data Flow Diagrams

+ Coupling and Cohesion properties

+ Computation and Communication Cost

Graphical representation of the software structure

Loops are manually unrolled.

Klocwork's inSight tool provides functionality for static annotation-capable graphical representation of the software architecture.

The analysis methods are use-case based and data driven.

Bottom-Up Annotation

- Computation and Communication Cost Analysis
 - Estimates number of reads, writes, executes and multiply instruction by analyzing every LoC
 - For e.g. `i = i + 1; //` 1 – Read, 1 – Write, 1 – Execute, 0 – Multiply
- Coupling and Cohesion Analysis
 - Complexity analysis by utilizing multiple level analysis provided by Klocwork inSpect

Concurrency Modeling

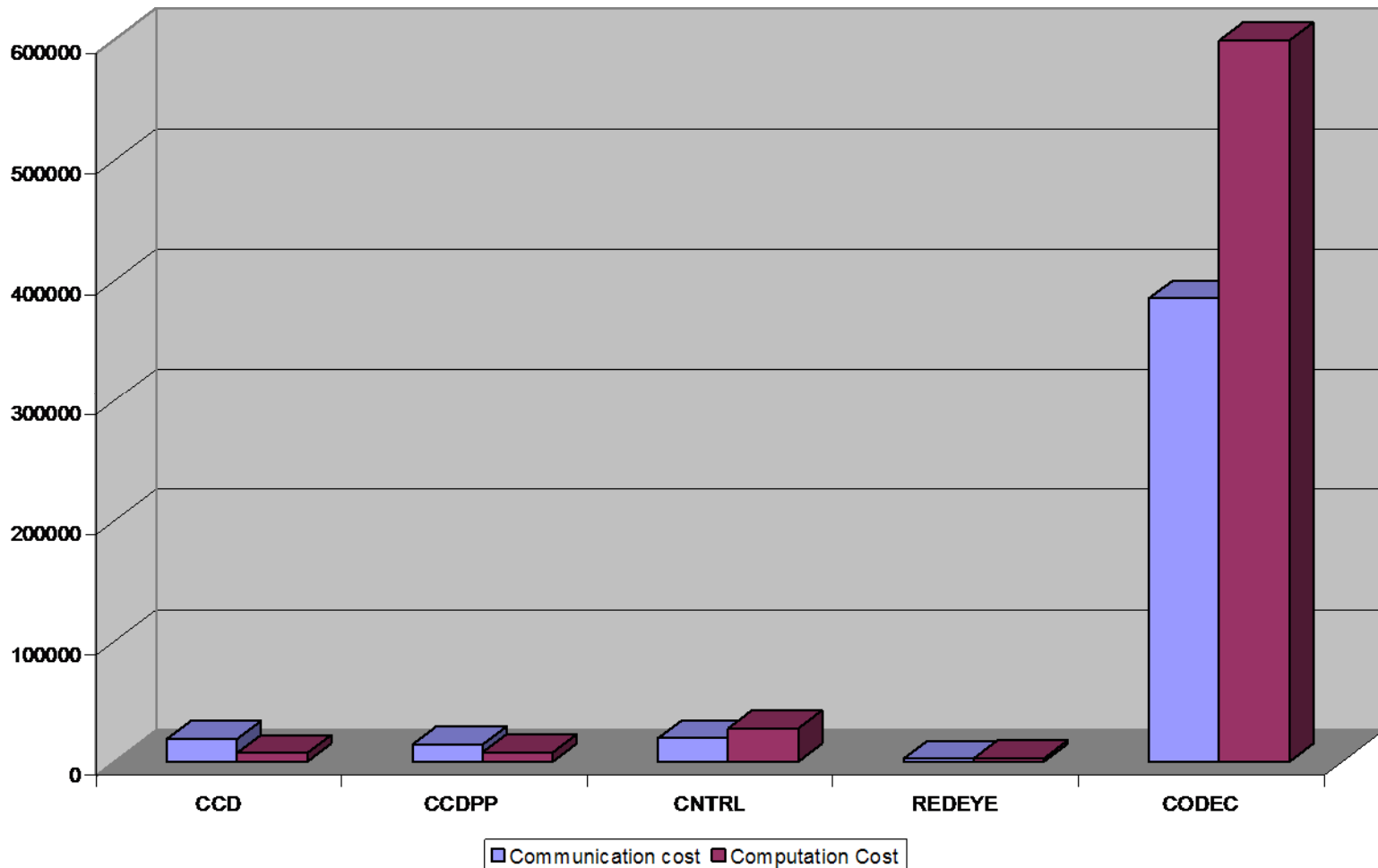
- Used Labeled Transition System Analyzer (LTSA) 2.2 provided by Jeff Magee and Kramer
- Needed to resolve concurrency issues while designing
 - Multicore Architecture
 - Bus / Interconnection Network
 - Memory
 - Cores
 - Multicore Architecture Scheduler (OS Layer)
- Benefits
 - Dramatically reduces development time
 - Produces scalable, reusable and reliable design

Middle-Out Analysis

- Middle-out analysis is performed by combining bottom-up annotation and top-down representation together with intermediate level system modeling.
- High-level software architecture and Multicore hardware architecture are combined by mapping software onto the hardware using MLDesigner.
- Performance evaluation is performed and Concurrency Cost is computed.
- Discrete Event Model of Computation is used in Simulations.

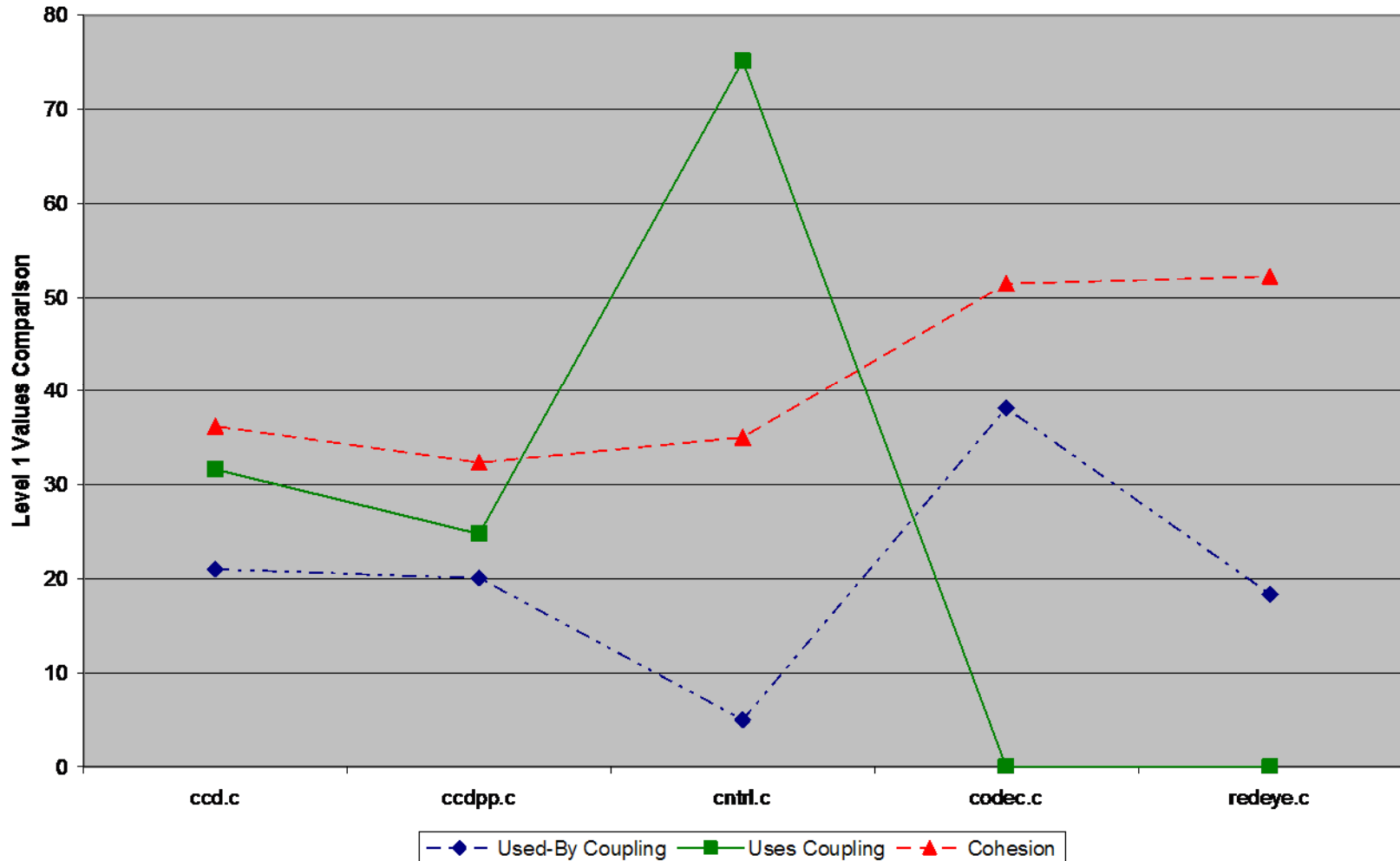
Computation and Communication Analysis

Overall Initial Distribution

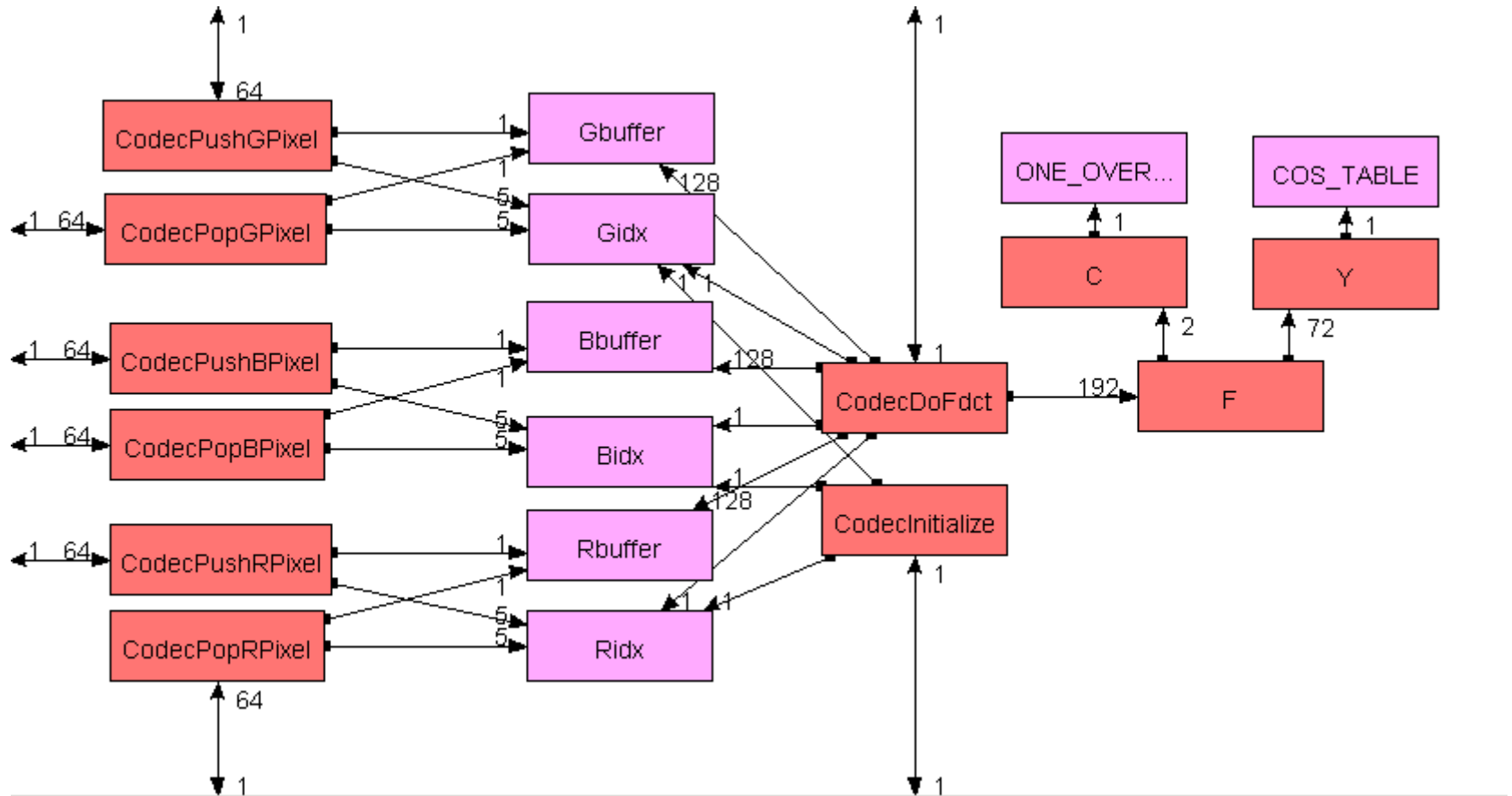


Complexity Analysis

Combined Chart

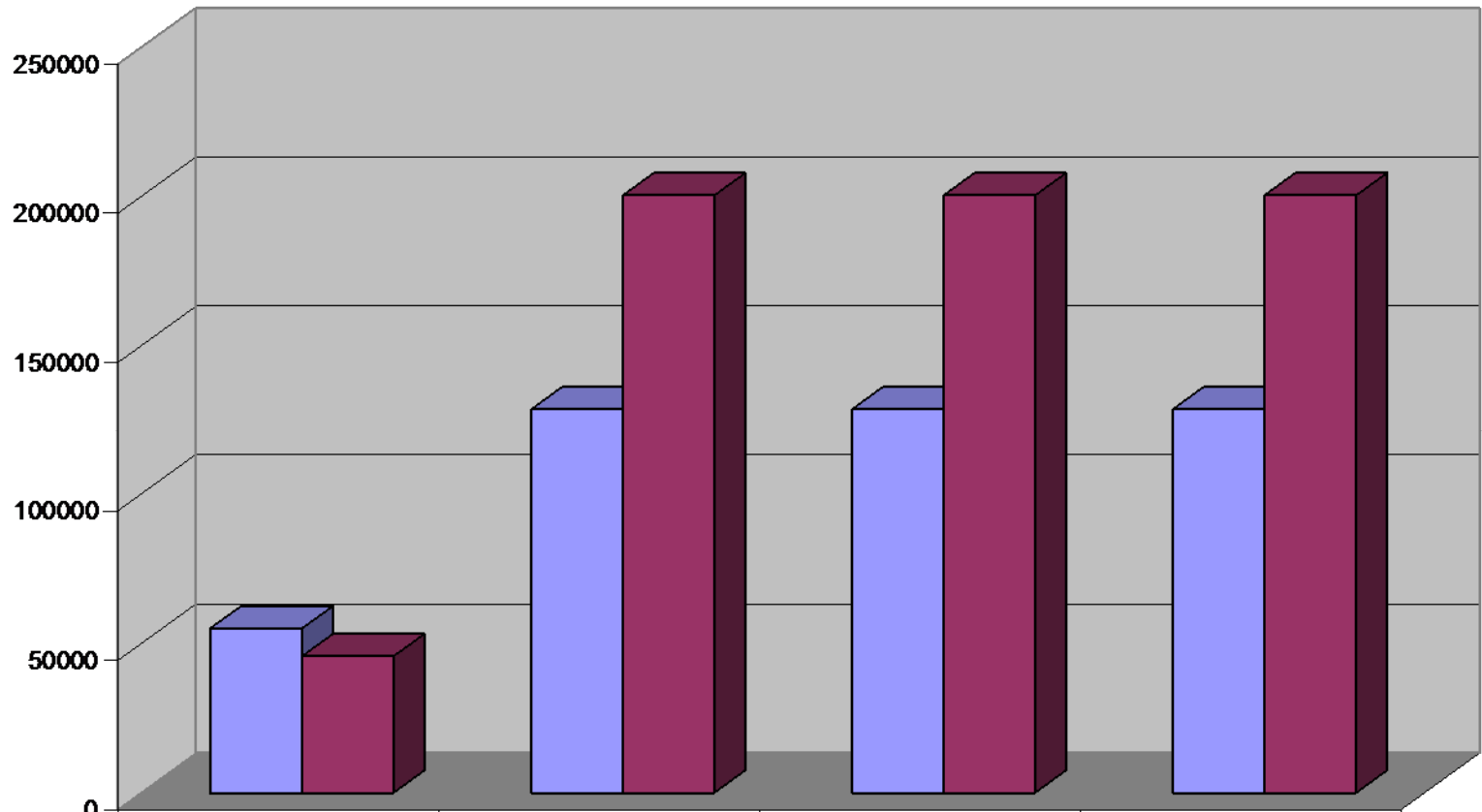


CODEC Graphical Analysis



4-Core Decomposition Option

4 Core Decomposition



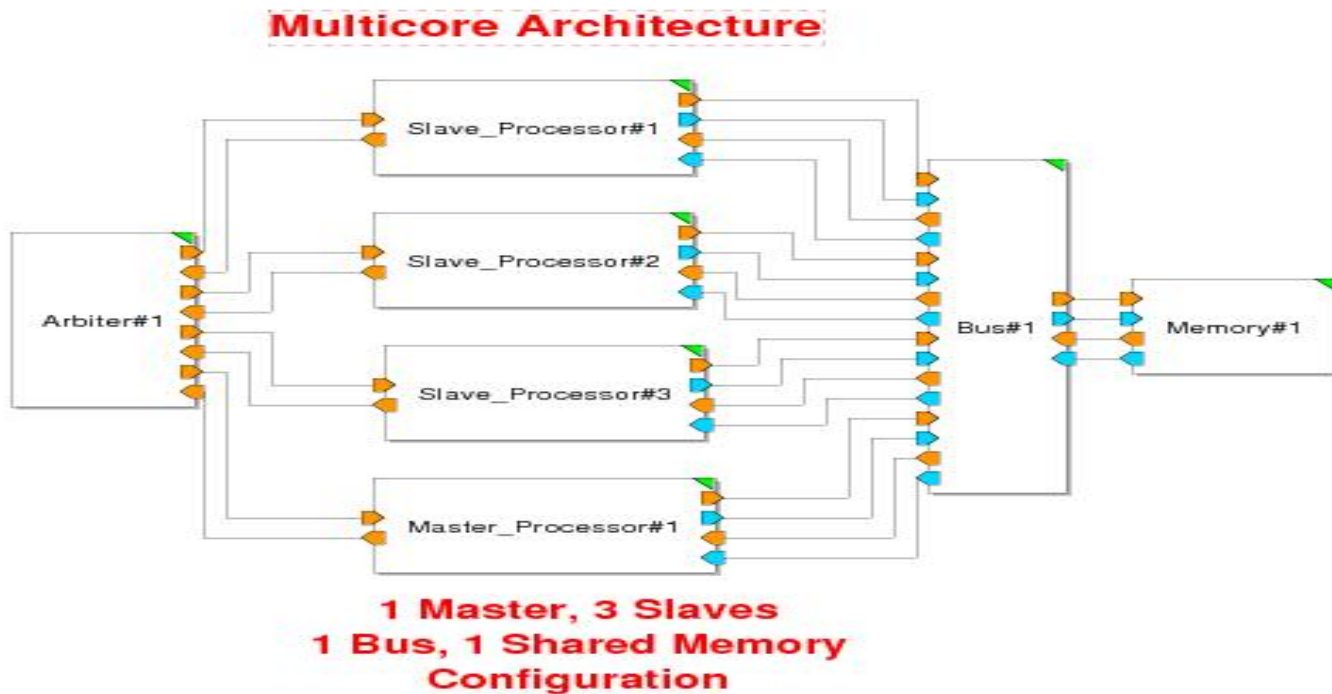
Communication cost	54966	128625	128625	128625
Computation Cost	45651	200336	200336	200336

Concurrency Modeling Exhaustive Analysis

```
Composition:
MULTICORE_SYSTEM = s1:SLAVE_PROCESSOR.SLAVE_DATA || s1:SLAVE_PROCESSOR.ATEND ||
s1:SLAVE_PROCESSOR.SLAVE_INTERFACE || s1:SLAVE_PROCESSOR.SLAVE_BEHAVIOR ||
s1:SLAVE_PROCESSOR.PROCESS_STATE || s2:SLAVE_PROCESSOR.SLAVE_DATA || s2:SLAVE_PROCESSOR.ATEND ||
s2:SLAVE_PROCESSOR.SLAVE_INTERFACE || s2:SLAVE_PROCESSOR.SLAVE_BEHAVIOR ||
s2:SLAVE_PROCESSOR.PROCESS_STATE || m:MASTER_PROCESSOR.MASTER_DATA || m:MASTER_PROCESSOR.ATEND ||
m:MASTER_PROCESSOR.MASTER_BEHAVIOR || m:MASTER_PROCESSOR.PROCESS_STATE ||
m:MASTER_PROCESSOR.MASTER_INTERFACE || {s1,s2,m}::BA.ARBITER || {s1,s2,m}::BA.bus:COUNTER(1) ||
{s1,s2,m}::BA.count:COUNTER(3) || {s1,s2,m}::BA.TIMEOUT(2) || b:BUS || {b}::MEMORY.TUPLE(task) ||
{b}::MEMORY.TUPLE(result) || {b}::MEMORY.TUPLE(task.stop)
State Space:
  2 * 2 * 9 * 10 * 3 * 2 * 2 * 9 * 10 * 3 * 9 * 2 * 11 * 3 * 10 * 29 * 2 * 4 * 4 * 22 * 3 * 3 * 3 =
  2 ** 60
Progress Check...
-- States: 10000 Transitions: 61329 Memory used: 11009K
-- States: 20000 Transitions: 122987 Memory used: 10003K
-- States: 30000 Transitions: 185229 Memory used: 10109K
-- States: 40000 Transitions: 247608 Memory used: 11037K
-- States: 50000 Transitions: 310286 Memory used: 12186K
-- States: 60000 Transitions: 373054 Memory used: 13832K
-- States: 70000 Transitions: 435663 Memory used: 14078K
-- States: 80000 Transitions: 498293 Memory used: 15275K
-- States: 90000 Transitions: 560983 Memory used: 15847K
-- States: 100000 Transitions: 623888 Memory used: 15399K
-- States: 102290 Transitions: 638244 Memory used: 14583K
No progress violations detected.
Progress Check in: 7266ms
```

102,290 possible states and 638,244 possible actions of the Multicore Architecture exhaustively analyzed for Deadlocks and Livelocks

Multicore Architecture MLDesigner Implementation



Results

- Analysis was performed on 2-Core and 4-Core architecture solutions
- 4-Core showed 23% performance improvement over 2-Core Solution
- 4-Core showed 40% performance improvement over 1-Core solution (following similar scheduling algorithm to the 4-Core Solution)

Conclusion

- Developed 10-Step High level abstract Methodology to choose QoS Driven:
 - Optimum Concurrent Model based Software Decomposition
 - Optimum Multicore Architecture
- We decompose large sequential code to concurrent code by maximizing software reuse