

# Software Decomposition for Multicore Architectures

Ankit Jain, Ravi Shankar

[ajain2@fau.edu](mailto:ajain2@fau.edu), [ravi@cse.fau.edu](mailto:ravi@cse.fau.edu)

Dept of Computer Science and Engineering  
Florida Atlantic University, Boca Raton, FL-33431

## Abstract

Current multicore processors attempt to optimize consumer experience via task partitioning and concurrent execution of these (sub)tasks on the cores. Conversion of sequential code to parallel and concurrent code is neither easy, nor feasible with current methodologies. We have developed a mapping process that synergistically uses top-down and bottom-up methodologies. This process is amenable to automation. We use bottom-up analysis to determine decomposability and estimate computation and communication metrics. The outcome is a set of proposals for software decomposition. We then build abstract concurrent models that map these decomposed (abstract) software modules onto candidate multicore architectures; this resolves concurrency issues. We then perform a system level simulation to estimate concurrency gain and/or cost, and QOS (Qualify-of-Service) metrics. Different architectural combinations yield different QOS metrics; the requisite system architecture may then be chosen. We applied this 'middle-out' methodology to optimally map a digital camera application onto a processor with four cores.

The computers and electronics industry will increasingly use multicore architectures (MA) in the near future. MA comprises of multiple cores on one physical chip. Cores also be known as System on Chip (SoC) or processors and are interconnected with each other using bus-based or packet-based interconnection network. Multiple medium or low-speed cores working together can achieve higher performance than a single high-speed core. MAs are expected to lower power dissipation and increase performance. MAs use GALS (Globally-asynchronous, locally-synchronous) design.

MA requires software developers to know underlying hardware architecture to address concurrency issues. These are:

- In design phase: Identify sections which could be parallelized; Reduce the use of locks while programming
- In development phase: Data Corruption, Deadlocks, and Livelocks.

Developers will require new tools and methodologies to synchronize among dynamic, concurrent, and real-time processes while writing multi-threaded code in a concurrent environment. Software designers will have to work with hardware designers to ensure proper software and hardware integration.

Multi-threaded software executes without issues on a single core, but when it is introduced into a MA it can result in data corruption, deadlocks and livelocks during execution. As the number of cores increase on a chip, it will not be physically feasible to synchronize all the tasks. In order for software to exist in an asynchronous environment the software developer will have to come up with innovative ways to design software. This thesis illustrates an integrated methodology on how to restructure existing sequential code into concurrent code which executes efficiently on MA.

The methodology is language independent. A ten step middle-out process was used that combined bottom-up annotation and decomposition, top-down modeling and middle-out analysis:

Step 1: Source Level Computation (COMP) and Communication (COMM) Cost Estimation: A module's COMM and COMP are calculated in terms of high-level attributes of RWXM (Read, Write, Execute and Multiply). COMP is calculated by determining the number of execute and multiply instructions and their cycle usage. COMM is calculated by determining number reads and writes instructions and their cycle usage

Step 2: Complexity analysis: This is performed using coupling and cohesion metrics. Modules with high coupling and cohesion properties are identified. The type of coupling is determined in step 4. The aim is to produce tightly coupled cohesive modules in sub-systems that have low coupling between them.

Step 3: Control and Data Flow Diagram (C/DFD) Analysis: Independent control flows can help determine concurrent tasks in the system. Data flow diagrams can help determine objects and data synchronization needs in the system.

Step 4: Top-Down Graphical Decomposition Analysis: We analyze graphical decomposition of a software system by graphically combining results from steps 1 and 2, and C/DFD from step 3. Software artifacts vulnerable to concurrency issues and types of coupling between modules are identified in this step.

Step 5: Software Grouping: Modules with low COMP and high coupling are grouped together. Control flow diagrams are utilized to determine multiple possible sub-systems so that behavior is not altered.

Step 6: Software Decomposition: Modules with high COMM and COMP are decomposed into smaller modules

to lower individual module cost. Data flow diagrams and concurrency issues are considered while performing this step.

Step 5 and 6 yield many potential software architectures. While performing steps 5 and 6, we made sure that the functionality of the software is not compromised. We accomplish this by making incremental changes. A subset of optimum architectures is decided upon only after computing concurrency costs (CONC) at a later step.

Step 7: System Level Concurrency Modeling: A concurrency model is developed using FSP language in the LTSA tool. This helps in designing the concurrent system. Note that both software and hardware exploit concurrency, and high-level modeling help uncover and resolve any resulting concurrency issues.

Step 8: Mid Level System Simulation: This is a discrete event based system simulation model which is developed using MLDesigner.

Step 9: Concurrency Cost (CONC) Analysis: Software architecture is mapped onto hardware architecture using scheduler capabilities of the hardware model. The hardware architecture is fine tuned in order to produce optimum results. Memory access time, cache size, etc. can be optimized in order to create optimum conditions on the hardware. This generates concurrency costs of the concurrent system, for a given software architecture.

Steps 7-9 are executed for each of the hardware architectures in combination with each of the software architectures obtained from an earlier step of software decomposition.

Step 10: System Architecture Selection: A subset of system architectures which meet the QoS (Quality of Service) requirements are selected, as appropriate for our given application.

In the methodology shown here, ways to decompose large sequential code to concurrent code by maximizing software reuse have been extensively researched. A ten-step methodology which uses techniques such as bottom-up annotation, middle-out analysis, and top-down representation was presented. The methodology provides a mechanism for software decomposition in concurrent system. Concurrency modeling tools FSP and LTSA were used to model and validate concurrent hardware and software models. COMM, COMP, and CONC were obtained related to every decomposition option. The CONC of all the decomposition options were compared and optimum solution was chosen after comparing CONC to QoS requirements. We can choose decomposition which is aimed at performance gain or lower energy consumption.

We evolved a high level representation of DCS by use of bottom-up annotation on DCS source code. DCS was used

as a benchmark due its widespread application across multiple domains, which requires scalability.

For software decomposition, we added a 10-step methodology to sequential DCS to concurrent code. Three different decomposition options were generated to map to the following multi-core architectures:

- 4-core architecture
- 9-core architecture
- 27-core architecture

COMM, COMP, and CONC for the 4-core decomposition options were computed. Six variations of 4-core architecture were designed to find optimum bus interrupt timer in the architecture. 4-Core architecture decomposition was compared to a 2-core decomposition. CONC obtained by comparing the two showed that 4-core architecture decomposition was 23% faster than the other. The results obtained from 4-core decomposition were compared to our QoS requirements. Our QoS requirements were to lower energy consumption and gain performance. We chose the 4-core architecture with the bus interrupt timer period of 16 x CMM as the optimum decomposition.

By using this methodology, we have shown that restructuring existing sequential software to concurrent execution was possible. CONC for 9-core and 27-core architecture were not computed, so 4-core architecture decomposition cannot be claimed as an overall optimum solution. We found ways to lower energy consumption and gain performance for execution of DCS.

## Acknowledgements

This work was funded by iDEN, Motorola.

## References

- [1] Sutter, Herb, *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*, Dr. Dobb's Journal, 30(3) March. Internet. <http://www.gotw.ca/publications/concurrency-ddj.htm>. 2005.
- [2] Gomaa, Hassan, *Structuring Criteria for Real Time System Design*, Proceedings of the 11th international conference on Software engineering. 1989.
- [3] Gomaa, Hassan, *Use cases for Distributed Real-Time Software Architectures*, IEEE. 1997.