Balancing LTE Protocol Load on a Multi-Core Hardware Platform Using EFSM Migration

Anas Showk, Shadi Traboulsi, David Szczesny
Institute for Integrated Systems,
University of Bochum,
44801 Bochum, Germany
Email: [Anas.Showk, shadi.traboulsi, david.szczesny]@rub.de

Attila Bilgic
KROHNE Messtechnik GmbH,
Ludwig-Krohne-Str. 5,
47058 Duisburg, Germany
Email: A.Bilgic@krohne.com

***Abstract*—In the past decade the mobile communication data rate is increased dramatically. Therefore, mobile terminals must have enough processing capability by migrating to multi-core processors. To exploit the multi-core processing power, we focus on a critical component in the future mobile terminal which is the load balancer. Its main role is to partition and balance the load so as to achieve an optimal sharing of load between cores and to eventually reduce power consumption. In this paper, we show how a model driven layered protocol stack can be parallelized and run on a multi-core modem. For instance, we illustrate how the Extended Finite State Machines (EFSMs) concurrency is employed to achieve the protocol stack parallelism. Furthermore, we move the load balancer to the modem subsystem layer by using the EFSM migration between cores. The proposed migration scheme during run time replaces the classic thread migration scheme and reduces the thread context switching overhead which definitely improves the performance. In addition, we present a semi-dynamic load balancer implementation accompanied with customized data pipeline scheduler for future multi-core LTE smart phones.**

***Keywords*-LTE protocol stack, multi-core mobile terminal, parallel embedded software, load balancing, model driven development.**

I. INTRODUCTION

The Long Term Evolution (LTE) is the enhancement of Universal Mobile Telecommunications System (UMTS) and is optimizing its radio access architecture. The targets of LTE are to increase the data rates to 100 $\mathrm{Mbit/s}$ in the downlink and 50 $\mathrm{Mbit/s}$ in the uplink. Due to the exponential growth of data rate in the mobile communication systems during the past decade, more efforts have been invested in order to achieve the required performance that satisfies the increasing processing demand for computational intensive applications. The computational power provided by single processing units, at reasonable power consumption, seems to grow slower compared to the application needs. Therefore, research is focused on migrating to multi-core architectures which can provide a better balance between performance, power consumption, flexibility and scalability. For instance, multi-cores can provide the required computing performance while allowing lower clock rates to achieve power efficiency and offer a second dimension in resource allocation.

Developing next generation mobile communication protocol can gain by reusing established approaches and best practices like Model Driven Development (MDD). For instance, the Specification Description Language (SDL) is commonly used in the previous mobile communication protocols. The dynamic behavior in an SDL system is described in processes using Extended Finite State Machines (EFSM) [1].

Data pipeline scheduling is accomplished by apportioning protocol functionalities into a series independent steps, where the output of one step is the input to the next. However, each step can be executed on a different core in order to constitute separated steps in a pipeline. These parts might be different protocol stack layers or specific protocol functions depending on the design. Pipelining can be very powerful if the degree of parallelization is high. Even though, it may take some efforts in order to fill the pipeline and generate a constant throughput. For instance, pipeline steps should have the same execution time and they must be tuned such that one stage does not become a bottleneck.

A load balancer is a component that distributes the computational load between two or more entities such as cores, clustered computer systems, network links or other resources, in order to get optimal resource utilization. In the embedded domain, load balancing provides an efficient execution of multiple threads on processors with multiple cores for concurrent and parallel applications. Depending on the implementation of the decision making of the load balancer, it can be a static, dynamic, or combination of them [2].

In this paper, we present the parallelization of model driven LTE protocol stack. In addition, we illustrate the design and implementation of a semi-dynamic load balancer accompanied with data pipeline scheduling for future multi-core LTE modems. It is moved to the modem subsystem layer by migrating the EFSMs between cores depending on the required data rate. For example, at very low data rates all protocol EFSMs should run on single core. If the data

 ISBN: 978-1-61208-139-7

rate is increased some of the EFSMs are moved to a second core in order to provide enough processing power. While increasing the data rate further, the load balancer distributes the EFSMs on more cores as needed.

This paper is organized such that, Section II shows the previous work done in this field. The smart phone system architecture is presented in Section III including the software stack and the hardware platform. The data pipelining scheduler for LTE protocol stack is discussed in Section IV. In Section V, the load balancer design and implementation is presented followed by the conclusion in Section VI.

II. Related Work

Embedded multi-core scheduling is investigated by researcher from different perspectives. For instance, the time server technique was developed in order to cope with scheduling several applications on the same platform [3]. This solution is used extensively for single core embedded systems and was also extended for multi-processor systems [4], [5]. Some power-aware scheduling algorithms utilize Dynamic Voltage Scaling (DVS) so as to optimize the power consumption [6], [7]. Moreover, in [8], a scheduling solution was used to achieve fault-tolerance for embedded systems with soft and hard timing constraints.

In [9], an efficient, optimal pipelining algorithm for associating a task chain with a chain of processors was explained. The pipelining algorithm is based on new data structure, called the layered assignment graph. A flow graph scheduling algorithm which considers pipelining, retiming and hierarchical node decomposition is presented in [10]. Research on pipeline scheduling is at a significantly less mature state than on the classical scheduling problem [11].

Researchers investigated load balancing for embedded multi-core systems on the operating system's level using the thread migration technique. However, most of the embedded multi-core system load balancing techniques used the thread level migration. For instance, based on the virtualization concept, the most common approaches in scheduling techniques and load balancing for embedded mobile communication systems are discussed in [2]. However, to the best of our knowledge, researchers have not used the EFSM migration for load balancing. In contrast to the other researchers, we develop an innovative EFSM migration scheme in order to minimize the thread context switching frequency. In addition, we customize data pipeline scheduling and integrate it with our load balancer in the modem subsystem layer so as to have a parallel execution of LTE protocol stack on the multi-core modem of a mobile terminal.

III. System Architecture

A mobile terminal architecture depicted in Fig. 1 is divided into three main parts: the hardware platform, the operating system layer and the modem subsystem layer. ARM processors are widely used in mobile phones and specifically the ARM11 core is representative for a mobile terminal state-of-the-art hardware platform [12]. Therefore, we have chosen ARM RealView® base board including the ARM11 MPCore™ to be our hardware platform. More details on the base board can be found in [13], [14]. In addition the L4/Fiasco microkernel is selected as an operating system. Moreover, the modem subsystem layer is composed of the load balancer and LTE protocol stack as illustrated in Fig. 1. In the next subsections, more details are given about the operating system and the modem subsystem layers.

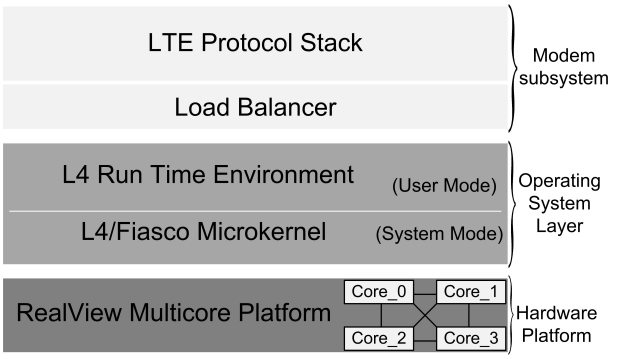

Figure 1. The system architecture of the future mobile terminal.

A. The Operating System's Layer

The L4/Fiasco Microkernel operating system is composed of two layers, the L4/Fiasco microkernel and the L4 runtime environment (L4Re) [15] as shown in Fig. 1. The selection of such a modern operating system is done because it represents a robust Real Time Operating Systems (RTOSs) as well as utilizes the concepts of microkernels [16]. The services of each layer of the deployed OS are illustrated in the following.

The L4/Fiasco microkernel is running in processor privileged mode and is responsible for controlling the underlying hardware. It provides a minimal set of mechanisms like tasks, threads, and Inter-Process Communication (IPC). Fiasco kernel services are implemented in terms of kernel objects. A task constitutes of an address space where one or more threads can execute. Multiple threads are scheduled by Fiasco's priority-based and preemptive scheduler. The scheduler is controlled by the timeslice length, priority and the maximum controlled priority. Each thread is scheduled for maximum time equal to the associated timeslice length. The kernel uses multiple-level round-robin queues such that there is a queue associated with each priority level. The combination of all the queues represents the kernels ready queue. Unlike the timeslice length and priority, the maximum controlled priority is not thread based but rather task based. It is specified for every task at the creation time and all threads in the task will have the same value. In addition, an IPC kernel object provides the basic communication

mechanism in L4-based systems and is used mainly for transmitting arbitrary data between threads.

On the other hand, the L4Re offers a basic set of abstractions and services, which are useful to implement user-level applications on top of the L4/Fiasco microkernel. It consists of a set of libraries mainly responsible for memory and IO resource management. The kernel does not migrate threads dynamically, but only supplies needed functionalities for its applications.

B. Load Balancer

The main job of the load balancer is to monitor the execution of applications and share the load between the available cores. In addition, it should decide where application components are initially started and when they need to be migrated in order to satisfy an optimum criterion like performance, power consumption, etc. Only the load balancer has a global view on the total load of the system and can thus distribute it on cores according to the previously specified requirements.

The load balancing is achieved within the modem subsystem layer by utilizing the EFSM migration scheme. It's design and development procedure can be divided into two main steps offline analysis and online processing. The first step (i.e., offline analysis) is accomplished at compile time to collect the information about LTE protocol stack and generate LTE scheduling tables which include the process-thread mapping tables and LTE configuration table prior to system execution. The process-thread mapping tables define the association of the EFSMs to threads depending on LTE states. LTE states are identified according the required data rate and how many cores should be involved in order to achieve this data rate. On the other hand, during system's run time the online processing includes all the actions needed to reconfigure the system when the LTE state changes. For example, when the load balancer detects an LTE state change, it will flush the pipeline by processing all the scheduled task. Then reconfigure the thread activation scenario and migrate the EFSMs between cores depending on the new LTE state in order to balance the load between cores. In Sect. V detailed explanations of the load balancer design and implementation are given.

C. LTE Protocol Stack

Developing software for LTE mobile terminal can greatly benefit from reusing prevailing approaches and best practices. For over a decade, most global installations have taken advantage of Model Driven Development (MDD) for communication products; oftentimes with tools using the Specification Description Language (SDL). Therefore, an SDL tool is selected to develop the access stratum part of LTE protocol stack in the mobile terminal side. Since the user plane is more computational intensive, the modeling targeted only the user plane part.

The access stratum part of LTE protocol stack includes layer 2 which is divided into three sublayers: Medium Access Control (MAC), Radio Link Control (RLC), and Packet Data Convergence Protocol (PDCP) [17]. Fig. 2 illustrates the LTE data flow from the mobile terminal perspective. In the uplink direction the mobile terminal generates packets and sends them through the air interface to an evolved base station (eNodeB). The building of packets payload and header as well as the downlink processing are modeled according to the 3GPP standard of LTE [18], [19], [20]. On the other hand, the mobile terminal receives Transport Blocks (TBs) in the downlink direction from the air interface and processes them through the MAC, RLC, PDCP and IP layers. The functionalities of the mentioned layers are modeled as described in more details in [21], [22].

The dynamic behavior in SDL systems is described in the SDL processes using EFSMs. Processes in SDL can be created at system start, or created and terminated dynamically at runtime. The concept of process instances that work autonomously and concurrently makes SDL a true real time language. The other advantage of the processes concurrency is making the parallelism easier to identify and exploit in contrast to pure C programming. It is clear, from Fig. 2, that the processing of packets in each layer should be done after the previous layer in a sequential manner. Therefore, the LTE protocol stack is modeled by dividing its functionalities into several EFSMs, which are communicating using the SDL asynchronous messages communication facility. For instance, the output from an EFSM is the input to the next EFSM and the latter's output represents the input to the one after, and so on, formulating a chain of EFSMs. Even though, the protocol stack processing is parallelized by exploiting EFSMs concurrency together with data pipeline scheduling.

IV. Pipeline Scheduling

In general, scheduling algorithms suitable for embedded systems are mapped to two major classes: static and dynamic. The scheduling involves three steps: assigning tasks to processors, ordering execution of these tasks on each processor, and determining when each task fires such that all data precedence constraints are met. To reduce the run-time computation, all the three steps are performed at compile time by a static scheduler. On the other hand, the scheduler which accomplishes these steps during run time is called a dynamic scheduler.

Generally, pipeline schedulers aim to efficiently divide a task into stages, allocate some cores to stages, and create schedules for each pipeline stage. As a consequence, the stage with the longest execution time in the pipeline determines the throughput of the multi-core system. In general, pipelining can considerably increase the throughput beyond what is obtainable by the classical (minimum-make-span) scheduling algorithms. However, this increase in throughput

 ISBN: 978-1-61208-139-7

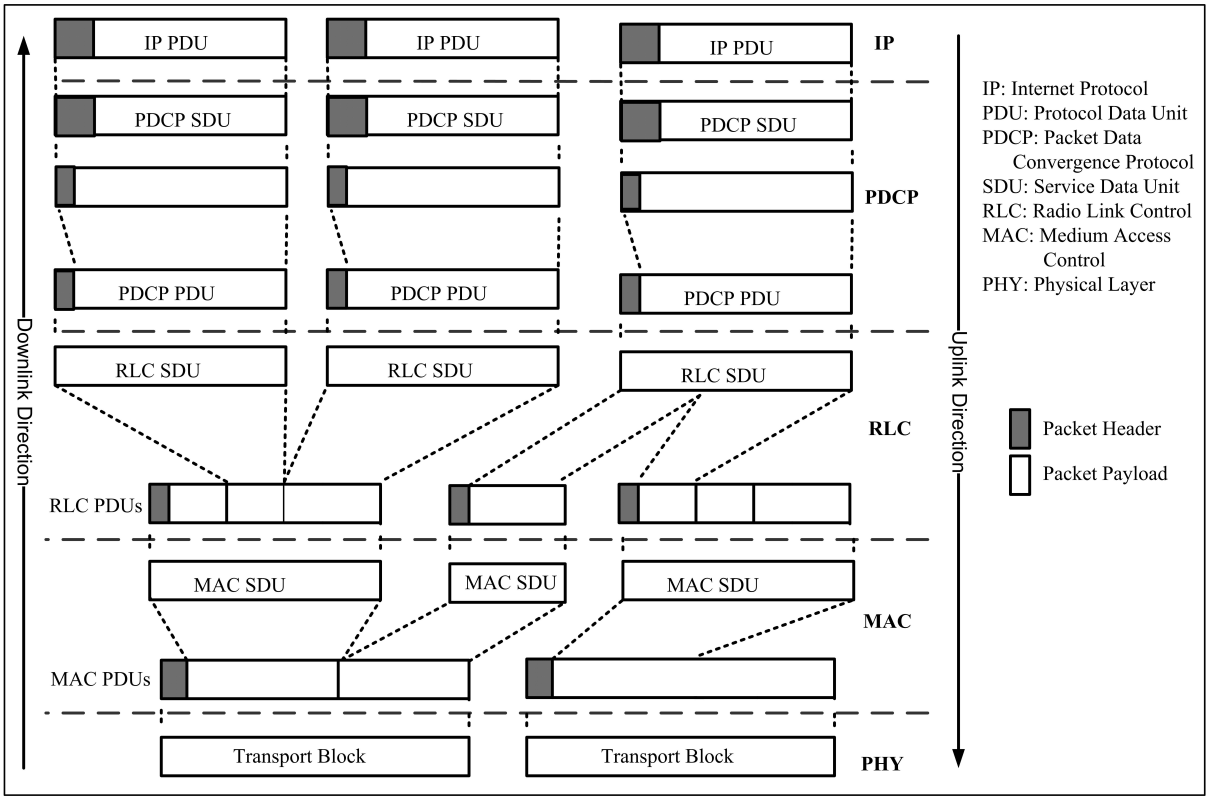

Figure 2. The LTE protocol data flow in the mobile terminal side.

may cost a significant increase of latency compared to the classical (minimum-make-span) schedulers [11].

The new direction that is relevant to embedded multi-core scheduling is the exploration of pipeline scheduling algorithms, which concentrate on throughput as the main performance metric. Hence, the presented data pipeline scheduler is adopted and customized for LTE protocol stack in order to improve the performance of the multi-core hardware platform. In addition, the scheduler is implemented on the SDL level by exploiting the message communication between EFSMs for several thread activation scenarios depending on number of needed cores.

As an example, a thread activation scenario of the LTE protocol running on four cores is depicted as a message sequence chart in Fig 3. All LTE EFSMs are distributed on four cores in such a way that processing a packet within the same core is illustrated as one task before the control is given to the other core for further processing of the same packet. For instance, the first packet (P_1) is processed by the first core (i.e., $Core_0$) while other cores are idle. After that, $Core_0$ sends the packet P_1 to $Core_1$ for activating the thread on $Core_1$ to do further processing and sends the message $Trigger$ to itself in order to start processing the second packet (P_2). consequently, both packets P_1 and P_2 are processed in parallel on cores $Core_1$ and $Core_0$, respectively. After processing P_1, $Core_1$ and $Core_2$ hand it over to the next core and send the $Trigger$ message to $Core_0$ to start working on a new packet. The last core $Core_3$ sends a $Trigger$ message to $Core_0$ when ever it finishes processing a packet in order to keep the pipeline full all the time. The only exception arises when there is an LTE state change and the pipeline should be cleaned thus it should not send this message. This technique of asynchronous SDL message communication is used for multi-core synchronization at a high abstraction level. One of the disadvantages of this scenario is the long latency of filling the pipeline (i.e., from start processing P_1 on the first core up to finish processing of the same packet on the last core). However, the throughput will increase if the load is distributed evenly between cores.

V. Load Balancer Design and Implementation

Power consumption and performance are very important factors for embedded multi-core systems dedicated to wireless communication protocol processing. Thus, in order to have an efficient multi-core mobile modem we concentrate on the load balancer which is one of the most critical components of the system. For example, a load balancer aims to share the load evenly between cores in order to ensure an optimal performance as well as to reduce power consumption. According to the load balancer design, it can

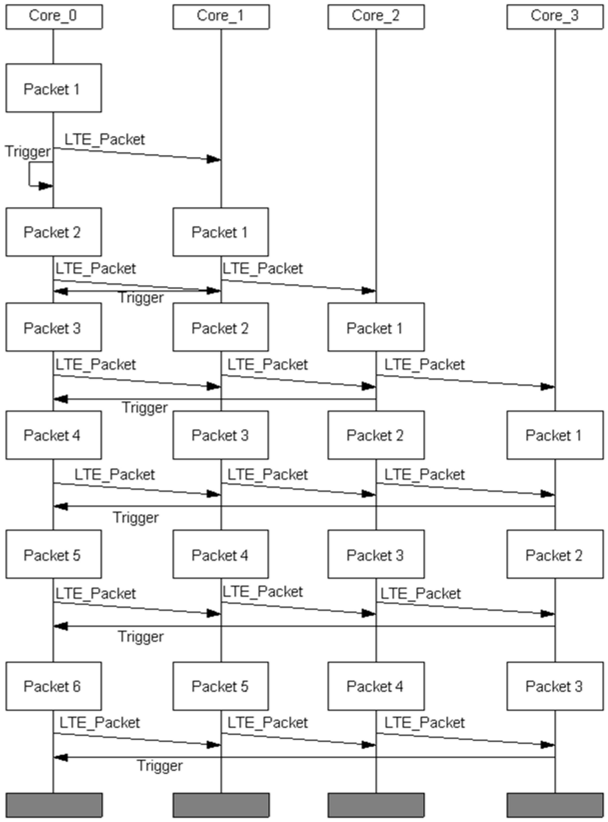

Figure 3. The message sequence chart of a thread activation scenario with four cores.

be a static, dynamic or combination of them. In this work, the load balancer development is divided into two main steps: offline analysis and online processing as depicted in Fig. 4. In the next sub-sections, more explanations of these steps are given.

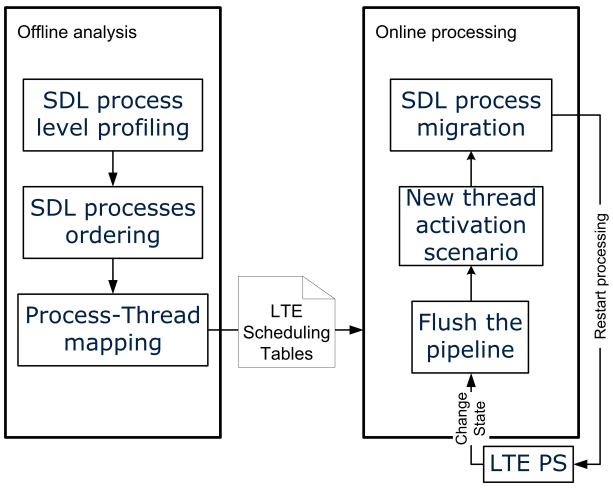

Figure 4. The offline analysis and online processing to balance LTE protocol load.

A. *Offline Analysis*

The LTE protocol stack design is accomplished using the hierarchical decomposition of SDL with system, block, sub-block and process as the main building blocks. Every protocol sublayer is realized by a sub-block (that is MAC, RLC,PDCP and IP). The latter is divided into sub-sub-blocks or processes depending on role or mode of the target layer. The behavior of the protocol is implemented in the SDL process level using concurrent EFSMs. Each EFSM is idle in the current state until it is triggered by an event to execute a transition and move to the next state. This event can be a message from another EFSM or even itself, an expiration of a timer or a change of internal variable.

A directed multi-graph is an ordered pair (A, E), where A is a set of actors (sometimes called nodes or vertices) and the set E is order pair of nodes called edges. Graphically, actors are represented by circles and edges are represented by arrows connecting the circles. Each edge is an ordered pair (a_1, a_2) where $a_1, a_2 \in A$. If $e = (a_1, a_2) \in E$, we say that e is directed from a_1 to a_2; a_1 is the source actor of e, and a_2 is the sink actor of e. In a directed multi-graph two or more edges can have the same source and sink actors and loops from the actor itself are allowed.

The data-flow graph is a directed multi-graph which is a conceptual notion for expressing the function of a system. The actors are the computations and the edges are First-In-First-Out (FIFO) queues. The latter direct the data (or token) as an output from one computation to be an input to another one. In terms of SDL and EFSMs, an actor is an EFSM (e.g., the active transition of a state machine form current state to the next state), an edge is an SDL queue and a token is an SDL message. Therefore, the collection of all EFSMs that constitutes the LTE protocol can be represented by data-flow graphs.

During offline analysis, the LTE protocol stack is profiled to measure the cost of every EFSM. Afterward, the EFSMs are ordered depending on which one will be executed first according to the data precedences to look like the data-flow graph example depicted in Fig 5. In the example, a data-flow graph is illustrated by a set of pairs (A, E), where set A is bounded by the big circle and E is not shown for sake of clarity. The set A includes all the nodes which are part of LTE protocol models where:

$$A = \bigcup_{i=1}^{N} a_i \ .$$

The execution time of each actor is represented by C_i where $i = [1, 2, 3, ..., N]$ is computed so as to calculate the cost of a group of actors or even the total cost of the system. For simplicity, we assume that the cost C_i includes the edges delay (i.e., the transition time between actors when running on the same core). Therefore, the cost of the chain of actors between actor a_x and actor a_y can be calculated by the delay

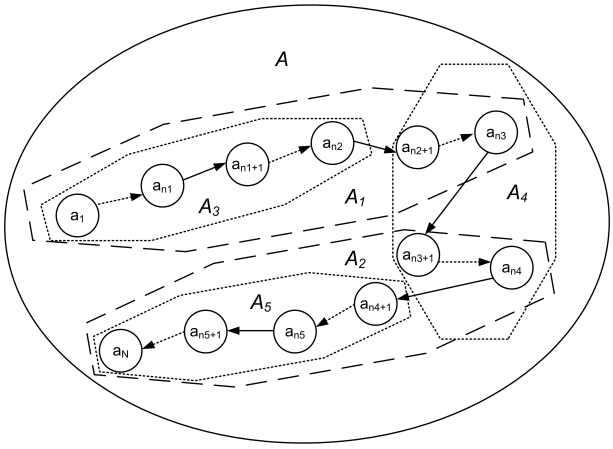

Figure 5. The data-flow graph example illustrates the mapping of actors into different sets in order to allocate them to one, two or three cores.

D_{xy} where:

$$D_{xy} = \sum_{i=x}^{y} C_i \ ; \ where \ \ 1 \leq x, y \leq N \ and \ x \leq y \ .$$

Since the ARM RealView® base board, which is used as a hardware platform, has four cores, the costs C_i are used to partition the set A in order to share the load evenly when one, two, three or four cores are needed. For example, subsets A_1 and A_2 in Fig. 5, which are bounded by octagons with dashed lines, are chosen in such a way that they should have almost equal costs (i.e., delay D_{1n_3} is equal to delay $D_{(n_3+1)N}$). Furthermore, the intersection between both sets should be equal zero and the union of them is equal to set A (or mathematically, $A_1 \cap A_2 = \phi$ and $A_1 \cup A_2 = A$). The same conditions are valid for sets A_3, A_4 and A_5 bounded by the octagons with dotted lines. The work can be extended to distribute the load on four cores by dividing the set A into another four sets by applying the same rules.

The system has only four threads which are created at the system startup and associated to each core. As a consequence, the process-thread mapping table for every LTE state in addition to the configuration tables are generated. The process-thread table illustrates which EFSM will run on which thread (core). The LTE state defines the number of cores needed according to the targeted data rate in addition to some other configuration parameters. For instance, a mobile terminal working at low data rate (like voice calls) leads to only one core that should be active. On the other hand, while streaming a very big video file, the system should utilize all the available four cores.

B. Online Processing

The SDL Suite™ tool is equipped with a deployment editor where the SDL system can be divided into separate threads. For example, at the design time the user can decide how many threads will be included and which EFSM will run in which thread. Therefore, at system start SDL system's run time kernel creates threads and associates different EFSMs with different threads according to the created deployment diagram. If an EFSM receives a message or other event occurs, the associated thread will be woken up to execute the transition. In this paper we develop a method to modify the EFSM parameters and move it from one thread to another during run time. In addition, this is also possible even if the two threads are running on different cores.

In this setup, at system start, four threads are created and each ARM11 core processor is allocated to each thread using L4/Fiasco's thread migration facility. When the LTE state changes during the LTE protocol execution, the SDL system will report the situation to the load balancer by calling a load balancer utility function to reconfigure the system. First of all, the load balancer flushes the pipeline by finishing all the scheduled tasks. After that, it reconfigures the thread activation scenario according to the new LTE state. Finally, it migrates some EFSMs depending on the process-thread mapping tables which are generated offline at compile time so as to balance the load between cores before restarting the LTE system processing of a new packet.

The EFSM migration is tested and the thread activities are monitored and printed in Fig. 6. The LTE threads with the names *lte_ps-main*, *lte_ps10001*, *lte_ps10002*, *lte_ps10003* and *lte_ps10004* are executed on cores $Core_0$, $Core_1$, $Core_2$, $Core_3$ and $Core_0$ respectively. The main thread of the system (i.e., *lte_ps-main*) includes the SDL system's run time kernel as well as the load balancer. At the beginning, all EFSMs are associated to thread *lte_ps10001* and run on $Core_1$. Then, for increasing LTE data rate some EFSMs are moved to thread *lte_ps10002* in order to distribute the load between $Core_1$ and $Core_2$ and the system output is verified. The number of used cores is increased to three and four while higher data rates are achieved.

In order to compare the performance of system when using EFSM migration with same system employing classical thread migration, assume the EFSMs are statically associated to different threads. Fig. 7 illustrate how the system load can be divided to threads which are distributed on multi-core. The X-axis represents the number of active cores in each case. On the other hand, the Y-axis shows the portion of the load on each core and the partitioning into threads. It is clear that, the minimum number of threads needed to distribute the load when activating one, two, three and four cores is six threads.

For example, at low data rate all threads should be migrated to run on only one core. As a consequence, the processing time of a TB, on uplink and downlink together, includes thread context switching costs due to the transition from one thread to the other. In contrast, no thread context switching occurs when using our EFSM migration scheme which definitely improves the performance. Moreover, this is still the case while running the LTE protocol on more than

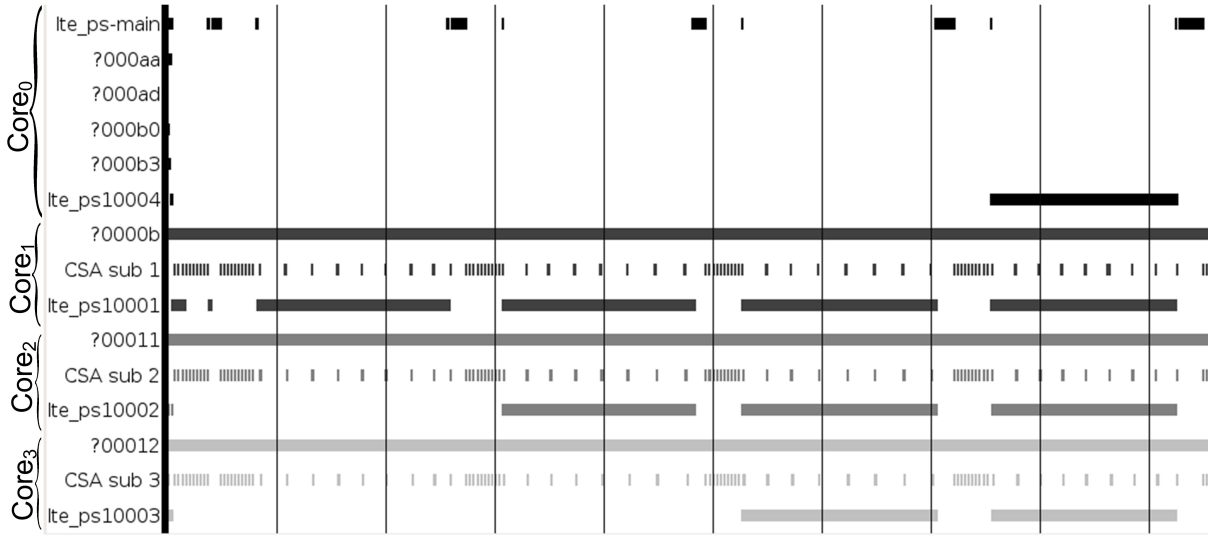

Figure 6. The monitoring of thread activities on ARM11 multi-core.

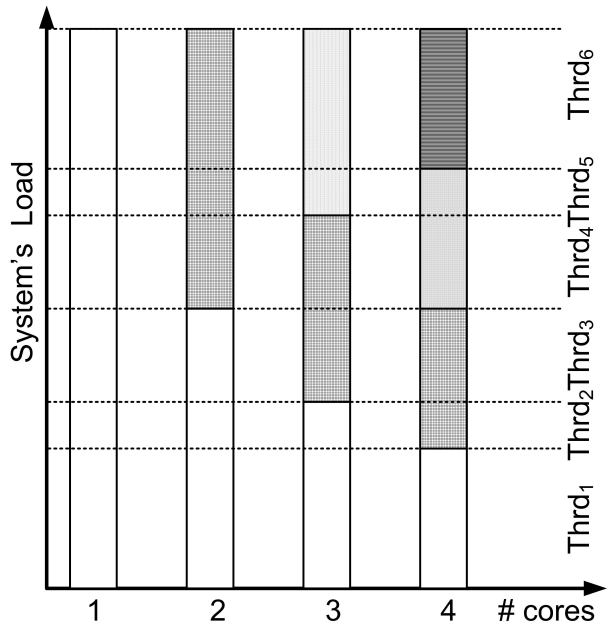

Figure 7. The system load partition into threads and distribution on one, two, three and four cores.

one core. More precisely, six, three and two thread context switches are avoided on one, two and three cores situation respectively. When four cores are active, four threads will execute on two cores (i.e., two threads for each core), thus introduces two thread context switches which can be avoided by employing EFSM migration. After measuring the thread context switching overhead, we found that it costs about 52 μs per switch. As a consequence, by utilizing EFSM migration we save around 312, 156, 104 μs when executing the LTE protocol on one, two, three cores respectively. In addition, in the four cores situation 104 μs from the execution time of processing one TB is reduced.

VI. Conclusion

A light version of the LTE protocol stack for the access stratum user plane is modeled using the SDL Suite™ tool. The SDL model is composed of several EFSMs which are associated with four threads to enable execution in a multi-core platform. The generated code is executed on ARM RealView® baseboard on top of an L4/Fiasco based RTOS. In this paper, we investigate load balancing and scheduling of LTE model driven protocol stack on a state-of-the-art multi-core mobile terminal. In addition, we show how the SDL EFSMs concurrency is exploited in order to achieve a parallel execution of the LTE protocol. As a result, we offer a parallel software architecture for LTE which is not existing today to the best of our knowledge. A new technique for load balancing in the modem subsystem level using EFSM migration is presented and successfully implemented. Moreover, the load balancing is accompanied with an adopted and customized data pipeline scheduling in order to make it suitable for the LTE protocol stack. In addition, we employ the thread activation scenario as a high level synchronization technique for multi-core mobile modem platform by utilizing the asynchronous message communication facility of SDL Suite™ tool. Last but not least, we prove that our innovative EFSM migration technique avoids the thread context switching and therefore, improves the performance in contrast to the thread migration counterpart. For future work, we are planning to continue enhancing the LTE mobile terminal performance by optimizing the IPC cost and decreasing its frequency. Even more, we will consider power consumption

 ISBN: 978-1-61208-139-7

as a parameter for scheduling in order to increase the battery life time of the LTE mobile terminal.

REFERENCES

[1] IBM Rational®, "SDL Suite™ User Manual," SDL Suite™ v6.1.

[2] D. Tudor, G. Macariu, C. Jebelean, and V. Cretu, "Towards a Load Balancer Architecture for Multi-Core Mobile Communication Systems," in *Proceedings of the 5th International Symposium on Applied Computational Intelligence and Informatics*, May 2009, pp. 391–396.

[3] S. Baruah and G. Lipari, "A Multiprocessor Implementation of the Total Bandwidth Server," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS04)*, June 2004, p. 40a.

[4] B. Brandenburg and J. Anderson, "Integrating Hard/Soft Real-Time Tasks and Best-effort Jobs on Multiprocessors," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS07)*, July 2007, pp. 61–70.

[5] S. Baruah, J. Goossens, and G. Lipari, "Implementing Constant-Bandwidth Servers upon Multiprocessor Platforms," in *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS02)*, September 2002, pp. 154–163.

[6] Z. Shao, M. Wang, Y. Chen, C. Xue, M. Qiu, L. Yang, and E. Sha, "Real-Time Dynamic Voltage Loop Scheduling for Multi-Core Embedded Systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 445–449, May 2007.

[7] Y. Chen, Z. Shao, Q. Zhuge, C. Xue, B. Xiao, and E. Sha, "Minimizing Energy via Loop Scheduling and DVS for Multi-Core Embedded Systems," in *Proceedings of the 11th international Conference on Parallel and Distributed Systems - Workshops (ICPADS'05)*, vol. 2, July 2005, pp. 2–6.

[8] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of Fault- Tolerant Embedded Systems with Soft and Hard Timing Constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, March 2008, pp. 915–920.

[9] S. H. Bokhari, "Partitioning Problems in Parallel, Pipelined, and Distributed Computing," *IEEE Transactions on Computers*, vol. 37, no. 1, p. 4857, January 1988.

[10] P. Hoang and J. M. Rabaey, "Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput," *IEEE Transactions on Signal Processing*, vol. 41, no. 6, pp. 2225–2235, June 1993.

[11] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. New York, NY, USA: Taylor & Francis Group, CRC., 2009.

[12] C. van Berkel, "Multi-core for Mobile Phones," in *Proceedings of the Conference on Design, Automation and Test in Europe*, April 2009, pp. 1260–1265.

[13] ARM, "RealView® Platform Baseboard for ARM11 MPCore™ User Guide," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0351c, March 2009.

[14] ARM, "ARM11 MPCore™ Processor Technical Reference Manual," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0360f, October 2008.

[15] TU Dresden, "The Fiasco Microkernel," http://os.inf.tu-dresden.de/fiasco.

[16] S. Traboulsi, F. Bruns, A. Showk, D. Szczesny, S. Hessel, E. Gonzalez, and A. Bilgic, "SDL/Virtual Prototype Co-design for Rapid Architectural Exploration of a Mobile Phone Platform," in *Design for Motes and Mobiles*. Springer-Verlag, September 2009, pp. 239–255.

[17] 3rd Generation Partnership Project (3GPP), "The LTE Protocol Specification, 3GPP Rel8," http://www.3gpp.org/Release-8.

[18] 3GPP TS 36.323, "Evolved Universal Terrestrial Radio Access (E UTRA); Medium Access Control (MAC) Protocol Specification," March 2009.

[19] 3GPP TS 36.322, "Evolved Universal Terrestrial Radio Access (E UTRA); Radio Link Control (RLC) Protocol Specification," March 2009.

[20] 3GPP TS 36.323, "Evolved Universal Terrestrial Radio Access (E UTRA); Packet Data Convergence Protocol (PDCP) Specification," March 2009.

[21] A. Showk, D. Szczesny, S. Traboulsi, I. Badr, E. Gonzalez, and A. Bilgic, "Modeling LTE Protocol for Mobile Terminals using a Formal Description Technique," in *Design for Motes and Mobiles*. Springer-Verlag, September 2009, pp. 222–238.

[22] A. Showk, F. Bruns, S. Hessel, A. Bilgic, and I. Badr, "Optimal resource management for a model driven lte protocol stack on a multicore platform," in *Proceedings of the 8th ACM international symposium on Mobility management and wireless access (MobiWac'10)*. ACM, October 2010, pp. 91–98.

 ISBN: 978-1-61208-139-7