LTE Uplink Receiver PHY Benchmark

Magnus Själander
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden
hms@chalmers.se

January 31, 2012

1 Introduction

The LTE Uplink Receiver PHY benchmark is an open-source implementation of the baseband processing in a long term evolution (LTE) mobile base station. The benchmark implements a realistic processing chain for incoming data to the base station, the so called uplink. The benchmark exists in several versions: a serial version that does all computations sequentially, a Cilk version that takes advantage of the Cilk framework for parallelizing the computations, and a POSIX threads (pthreads) version where creation of tasks and their distribution are managed by the benchmark itself and can easily be modified.

2 LTE Uplink Baseband Processing

A short overview of the resource allocation and signal processing involved for an LTE baseband uplink is presented in this section. For more detailed information on the LTE baseband standard, refer to the 3GPP specification series 36 [1, 3, 2].

2.1 Frequency and Time Allocation

To understand the processing of an LTE base station uplink we address how the frequency spectrum and time are allocated to the LTE transmitters/users (called UEs) that are communicating via a base station.

Each base station has a certain frequency band allocated to it (1.25 MHz to 20 MHz). This frequency band is divide into 15 kHz-wide sub-carriers. The time is divided into subframes that each last for one millisecond. A subframe is divided into two slots, each with seven SC-FDMA symbols. The symbols are arrange such that first three data symbols are transmitted, followed by one reference symbol that is used for channel estimation, and then three more data symbols are transmitted. The smallest scheduable unit is a physical resource block (PRB), which consist of twelve sub-carriers for the duration of one slot. An illustration of the frequency and time domain is shown in figure 1.

The granularity at which frequency and time is allocated to various LTE transmitters is physical resource blocks (PRBs) for each subframe. A specific LTE user gets a number of PRBs allocated by the base-station to be used for transmission during a subframe. For the next millisecond the transmitter will get a new set of PRBs that it can use for transmitting data. It is the work of the base-station to allocate which PRBs an LTE user can use for transmitting data during a subframe.

2.2 Channel Modulation and MIMO Systems

Depending on the signal quality between a transmitter and receiver various coding schemes can be used. For a low noise channel a higher-order modulation scheme can be used, e.g, 16 QAM or 64 QAM [5], to achieve higher data rates compared to when noise and interference are high.

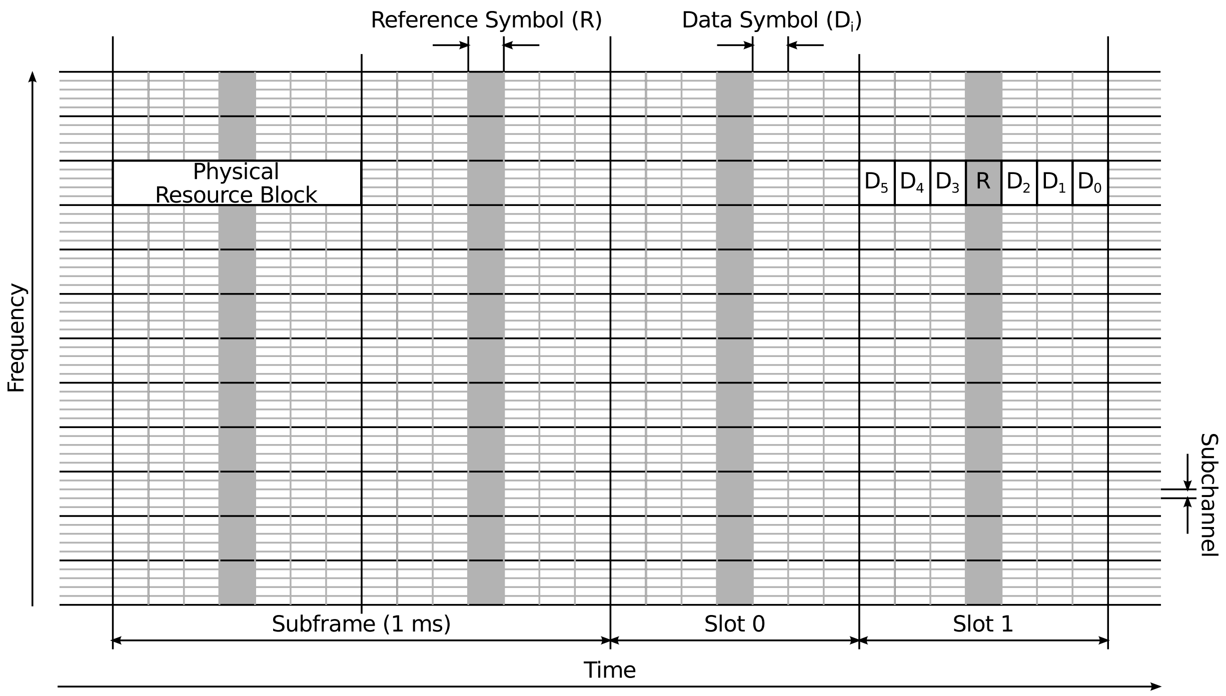

Figure 1: Illustration of frequency and time organization for an LTE base station.

Another technique for increasing data rate is the use of a multiple-input and multiple-output (MIMO) [7]. MIMO employs multiple input and output radio channels to improve the spectral efficiency and link reliability through higher tolerance for fading. Spatial multiplexing (SM), a transmission technique that can be used in conjunction with MIMO [4], allows transmission of several independently encoded data signals via multiple transmit antennas. The independently encoded data signals are called *streams* or *layers*. The latest standards for LTE advance include support for up to four transmission layers in the uplink [6].

2.3 Signal Processing for One User

The front-end of an LTE base-station receiver has a number of components starting with the radio receiver, filters, cyclic prefix removal, and fast Fourier transform (FFT) that is performed on all data that are received. These does not require any scheduling of tasks as all the data are streamed through the different components and treated in the same way.

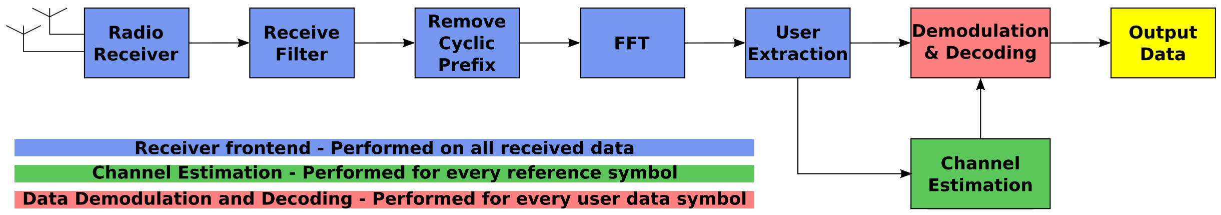
Radio Receiver
Receive Filter
Remove Cyclic Prefix
FFT
User Extraction
Demodulation & Decoding
Output Data
Channel Estimation
Receiver frontend - Performed on all received data
Channel Estimation - Performed for every reference symbol
Data Demodulation and Decoding - Performed for every user data symbol

Figure 2: Illustration of a typical model of an LTE receiver.

When the FFT has been computed the PRBs of one subframe for one user can be processed by a chain of signal processing kernels that typically can be modeled as shown in Figure 3.

For each slot in a subframe the channel needs to be estimated for the user before the data can be demodulated and decoded. This requires that the three first symbols are buffered before the reference symbol is received. Channel estimation first applies a matched filter that multiplies the received reference symbol (which has been distorted by the channel) with the defined reference symbol. An inverse FFT

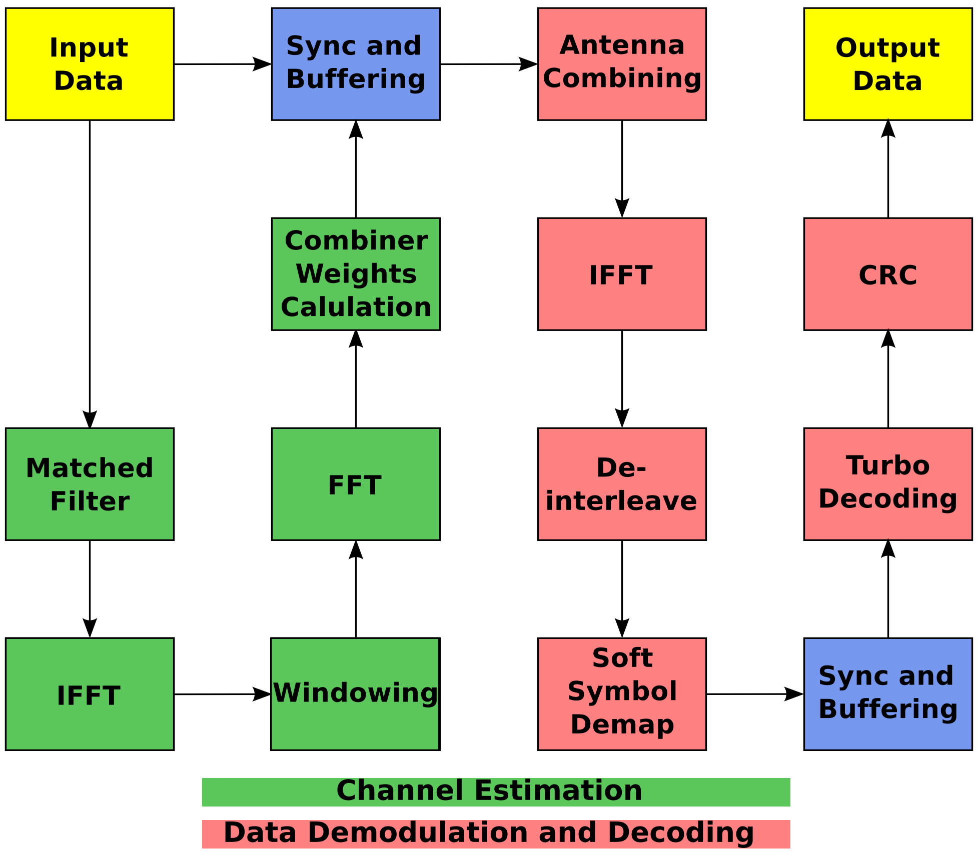

Figure 3: Illustration of a typical model for a one user part of the base band.

(IFFT) follows the matched filter, transforming the data back to the time domain, where a window is used to extract part of the time domain samples. An FFT then transforms these samples back to the frequency domain. For a MIMO system channel estimation must be performed for each receive antenna and for each layer. The results from each antenna and layers are used to calculate a number of combiner weights that are used for demodulation.

The combiner weights are used to combine the symbol data from multiple antennas and adjusts for the current channel conditions. The symbol data is then transformed back into the time domain by an IFFT. In the time domain the data is deinterleaved and soft symbol demapping is performed. The soft demapped symbols are fed through a turbo decoder and the decoded data is checked in a cyclic redundancy check.

3 Parallelism in LTE Baseband Processing

Parallelization is required to meet the performance requirements of processing one new subframe every millisecond. The baseband processing can be parallelized at different stages. The first obvious way of parallelization is to process each user separately (see figure 4). A base station can typically handle 10 to 20 users. However, the processing of different users can vary significantly (one user might use voice over IP at kbits/sec while another user might upload files at Mbits/sec). If the parallelization is only done across users the workload for different cores would become uneven.

The processing of a user is also possible to parallelize. During channel estimation the matched filter, iFFT, windowing, and FFT kernels are processed on the data from each of the receive antennas and layers separately. This is done because the channel is estimated for each individual antenna and layer. For a four antenna receiver and the maximum use of four layers it is possible to process the reference symbols in up to 16 (four antennas times four layers) individual tasks. The combiner weights computation considers

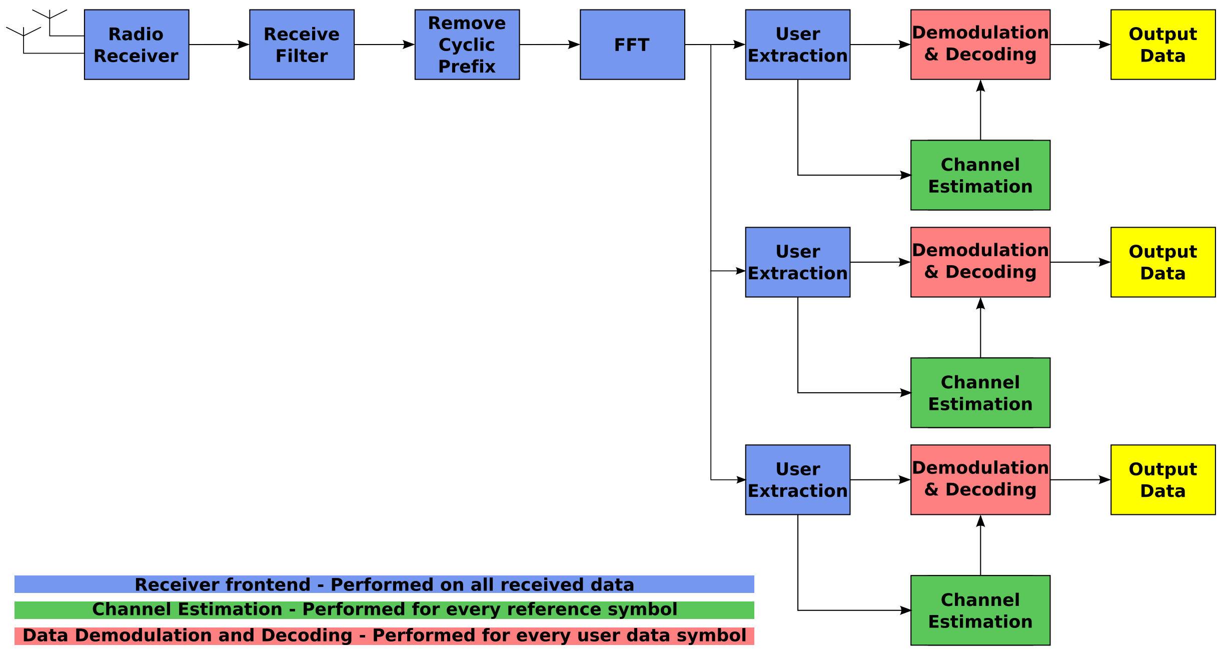

Figure 4: Illustration showing parallelization across three different users.

all the receive antennas and layers and is therefore not easily parallelized into individual tasks. For data demodulation and decoding, antenna combining and FFT is performed for on each separate data symbol and layer. As there are six data symbols in each slot and a maximum of four layers it is possible to process the data in up to 24 individual tasks. The remaining processing of the symbols are performed across all data and are therefore not easily parallelized into individual tasks. Figure 5 shows a schematic illustration of the parallelization of the channel estimation and data demodulation and decoding.

Further parallelization can be achieved at the algorithmic level, e.g., spread the computation of an FFT across multiple processing units. This type of parallelization is orthogonal to the parallelization considered here and could be performed as well to achieve even higher parallelization.

4 LTE Uplink Receiver PHY Benchmark Implementation

The LTE Uplink Receiver PHY benchmark is created to capture the dynamic behavior of an LTE baseband uplink as viewed by a base station. As the frontend (see figure 2) is statically defined and performed on all received data this is not part of the benchmark.

The dynamic behavior comes from the allocation of physical resources blocks (PRBs) to various users, the number of users that are allowed to transmit data, and the quality of the radio channel for each user. The channel quality defines the modulations and the number of layers that a user can use. For low-noise and low-interference channels higher order modulation techniques, e.g., 64 QAM, and multiple layers can be used to increase the rate at which a user can transmit data. The allocation of PRBs is performed by the base station and a new allocation is created for each new subframe. As a subframe lasts for one millisecond this is the period at which the dynamic workload changes. The workload for a subframe is defined by the following set of input parameters:

- number of users;
- number of physical resource blocks (PRBs) allocated to each user;
- number of layers used for each user; and
- modulation technique used for each user.

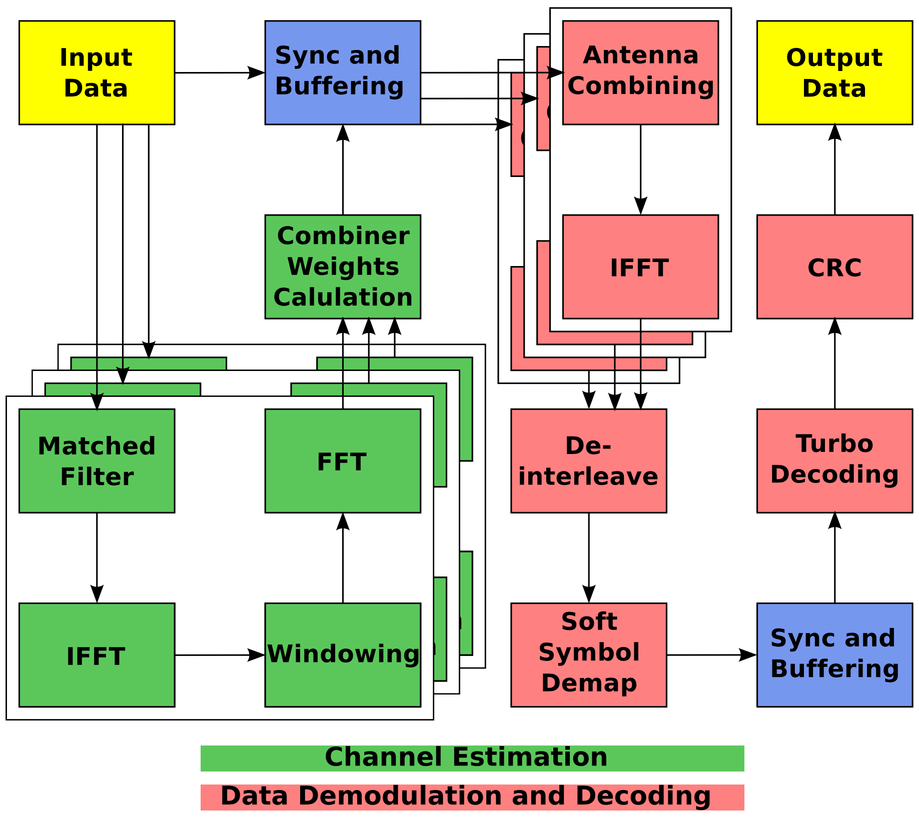

Figure 5: Illustration showing parallelization of the channel estimation and data decoding.

4.1 Parallelization Framework

The LTE Uplink Receiver PHY benchmark exist in three different versions: a serial version that processes a subframe sequentially, a version parallelized using the Cilk framework [9], and a version parallelized using POSIX threads (Pthreads) [8]. The serial version acts as a reference to verify parallelized versions of the benchmark. The Cilk version is implemented as an alternative to Pthreads. The Cilk framework provides an easy means of parallel application implementation, and it hides the need for scheduling individual tasks. Task creation and scheduling are handled by the Cilk runtime [9], thus adapting scheduling and providing resource management for the benchmark would require modifying the runtime. Such modifications has not been performed.

The LTE Uplink Receiver PHY benchmark uses Pthreads in order to create a portable solution. Pthreads is a well defined standard [8] supported by many operating systems and architectures. The Pthread version of the benchmark implements task creation, scheduling, and resource management, allowing for complete control of its behavior and its interactions with the underlying architecture. This is therefore the default version of the benchmark, and it is the version that is mainly described in this document.

4.2 Subframe Generation and Dispatch

The benchmark consists of one maintenance thread and a configurable number of worker threads. The maintenance thread of the benchmark is responsible for producing the inputs for each subframe and to dispatch a subframe every millisecond. In practice the rate at which subframes are dispatched is configurable to allow the benchmark to run on hardware that can not keep up with a rate at one subframe per millisecond. During initialization of the benchmark the main thread creates a defined number of worker threads and initialize all required data structures. After initialization the main thread enters a loop where input data and parameters for a subframe are created and dispatched every DELTA millisecond(s) (where DELTA is configurable, see Sec. 5.9).

4.2.1 Subframe Input Data

The input data is created once for a number of subframes and then reused across all subframes that are dispatched. This is to avoid the overhead of creating new data for each subframe while still having unique data for a number of consecutive subframes. The number of unique input data subframes to generate is configurable and set to ten as default.

4.2.2 Subframe Input Parameters

The input parameters to use for a subframe depends on the usage scenario that is to be modeled. The benchmark accommodates the use of different usage scenarios by providing two function calls for creating your own user parameter model. The first function *init_parameter_model(parameter_model *pmodel)* is used to initialize necessary variables for a model and the second function *user_parameters *uplink_parameters(parameter_model *pmodel)* is called for each subframe to get the number of users and the parameters and input data for the users.

4.2.3 Subframe Dispatch

With the input data and parameters generated for a subframe the maintenance thread waits for a signal alarm that has been configured during initialization to be triggered every DELTA millisecond(s). Immediately after a signal alarm the users of a subframe is written to a global queu for processing after which the main thread starts creating the input parameters for the next subframes and waits for the next signal alarm.

4.3 Subframe Processing

The actual processing of subframes are performed by a configurable number of worker threads. Each worker thread has a local task queue and if no work exist in that queu a worker thread will try to steal work from another worker thread. Before a worker thread tries to steal work from another thread it will first check the global user queu to assure that a new subframe has not been dispatched.

If a user exists in the global user queue an idle worker thread will remove the user from the queue and then start processing the subframe of that user (this thread is from now on referred to as a user thread).

4.3.1 Channel Estimation

The processing of a subframe for a user starts with the user thread creating a number of tasks that equals the number of receive antennas of the system times the number of layers used for the user. The tasks are placed on the local task queue and the user thread then starts processing the tasks until the queue is empty. If there are other idle worker threads they can steal tasks to help with performing the channel estimation. Once the local task queue is empty the user thread waits until the result from all tasks are available. This is necessary as other worker threads might still be processing some of the channel estimation tasks. When all data have been processed the user thread performs the combiner weight calculations.

4.3.2 Data Symbol Decoding

With the channel estimation performed it is possible to start demodulating and decoding the data. Antenna combining and inverse FFT can be performed independently for each data symbol and for each layer. To perform these operations the user thread creates a number of tasks equivalent to the number of data symbols times the number of layers. These tasks are placed on the local queue and the user thread starts processing the tasks until the queue is empty. Also in this case idle worker threads can steal work and help with the processing of the data. Before the data processing can proceed data from both the two slots are required. When all processing has been performed for the two slots the user thread preforms the processing that remains for the subframe, i.e., interleaving, soft demap, turbo decoding and cyclic redundancy check (CRC). Turbo decoding is a challenging task and is commonly performed by dedicated hardware. It is therefore not modeled by the benchmark. The data are simply passed through the call to perform the turbo decoding.

4.4 Verifying the LTE Uplink Receiver PHY Benchmark

For verification purposes of the parallelized uplink benchmarks the serial implementation is used as a golden model. The serial version is used to process a predetermined sequence of subframes and the result from each subframe is recorded and stored in a file. By processing the same sequence of subframes with a parallelized version of the benchmark and then comparing the result from each subframe against the golden data produced by the serial version the computation can be verified.

5 LTE Uplink Receiver PHY Benchmark Structure

This section describes the structure and the most important files of the benchmark.

5.1 uplink_parameters.c

The main purpose of uplink_parameters.c is to create input data and parameters for the subframes. Support for different parameter models is provided by placing them within preprocessor #if's and specifying the define of the model to be used. The main functions of uplink_parameters.c are:

void init_data(void) generates a global array of a structure that stores the input data for a subframe. There are currently two versions of this function. One that generates a number of random input data subframes and one that produces predefined data for a number of subframes. The predefined version is used during verification where known input data are required.

void init_parameter_model(parameter_model *pmodel) initializes the pmodel that is to be used for creating parameters for a subframe, e.g., setting counters.

user_parameters *uplink_parameters(parameter_model *pmodel) generates the users and their input parameters and data for a subframe. The function returns a linked list of *user_parameters* structures. Each *user_parameters* structure contains the input parameters for a user and a pointer to the user's input data.

Supported parameter models are:

PARAMETER_MODEL_STEP_NEW changes the probability for the number of layers and modulation that is used by a user. The probability starts low resulting in almost all users have one layer and QPSK as modulation. The probability is increased over 170 steps until all users will have four layers and use 64 QAM modulation. When maximum load has been reached the probability is then reduced over 170 steps until it reaches the same level as in the beginning. The time to remain in each step is configurable (default is 1 second).

PARAMETER_MODEL_STEP changes the workload in steps of 25% starting at 100% going down to 25% back up to 100% and finally down to 25% before exiting. Each step lasts for a configurable amount of time (default is 10 seconds).

PARAMETER_MODEL_RANDOM randomly generates number of users, layers, and resource blocks.

PARAMETER_MODEL_CORRELATION_RB changes the workload by incrementing the computational effort for a single user. This model can be used to create statistics of core utilization given a users input parameters.

PARAMETER_MODEL_CORRELATION_RB_SEC changes the workload by incrementing the computational effort for a single user. This model can be used to create statistics of core utilization given a users input parameters. In contrast to the above parameter model this will print core utilization statistics every second instead of only once for each parameter input.

PARAMETER_MODEL_CORRELATION_USER changes the workload by incrementing the number of users while keeping numer of resource blocks and layers constant.

PARAMETER_MODEL_VERIFICATION creates a predefined number of parameter combinations that can be used for verification. It also enables the functions that exist in the code for performing the verification.

5.2 uplink_serial.c

The file uplink_serial.c contains the serial version of the benchmark. After initialization the benchmark enters an endless loop where it will call the *uplink_parameter()* function (described in Sec. 5.1) and then process each user in sequence.

5.3 uplink.cilk

The file uplink.cilk contains a version of the benchmark that has been parallelized using the Cilk framework [9]. After initialization of the parameter model, input data, and a signal alarm it enters an endless loop where it will call the *uplink_parameters()* function (described in Sec. 5.1), wait for the signal alarm (triggered every DELTA millisecond, see Sec. 5.9), and then spawn a task for each user to be processed in the subframe.

5.4 uplink_user.cilk

This file contains the signal processing for a user. The functions of uplink_user.cilk are:

cilk void uplink_user_cilk(userS *user) is spawned by the main function in uplink.cilk when a user is to be computed. This function creates tasks for channel estimation and tasks for antenna combination and inverse FFT computation for the two slots of a subframe. It then performs interleave, soft symbol demap, turbo decoding, and cyclic redundancy check on all data of the subframe. Channel estimation for a slot needs to be performed before the data can be processed and the data of slot 0 is available before slot 1. The function therefore starts with spawning channel estimation tasks (equal to the number of receive antennas times the layers used for the current user). Sync is used to assure that all spawned channel estimation tasks are computed before performing weights calculation. Then tasks are spawned to compute the data symbols (equal to the number of layers used for the current user times numer of data symbols). Another synchronization is used to assure that all data has been computed before proceeding. When both slots have been computed the final interleave, soft symbol demap, turbo decoding, and cyclic redundancy check are performed.

cilk void compute_chest(task *taskX) is spawned by *uplink_user_cilk()* and performs matched filter, inverse FFT, channel estimation, and FFT on the reference symbol of the task.

cilk void compute_symbol(task *taskX) is spawned by *uplink_user_cilk()* and performs antenna combining and inverse FFT on the data of the task.

5.5 uplink.c

The file uplink.c is the main file of the version of the benchmark that has been parallelized using POSIX threads (pthreads) [8]. After initializing data and the parameter model it generates a configurable number of worker threads and their input data. Each thread is provided with the global queue for distributing users, the local queue of the worker thread, and pointers to all other worker threads queues for enabling work stealing. After this a signal alarm is configured to be triggered every DELTA milliseconds, where DELTA is configurable (see Sec. 5.9).

The main thread then enters an endless loop where it calls the *uplink_parameters()* function (described in Sec. 5.1), waits for the signal alarm and then places the users on the global user queue.

5.6 uplink_task.c

This file contains the main functionality of the worker threads. The functions of uplink_task.c are:

void *uplink_task(void *args) is the main function and is called when the thread is created. It sets the affinity of the thread, initializes some data, and then enters an endless loop. In the loop the thread will monitor the global user queue and if a user is found start processing it or look for a task that can be stolen from another thread.

void handle_user(user_queue *queue, task_queue *tqueue, task *tasks) is called if the thread detects that the global user queue is not empty. The function tries to remove a user from the queue and if successful it will start processing the user. The removal of a user can fail if the queue would become empty by the removal of a user by another thread. *queue* is the global user queue, *tqueue* is the local task queue that will be used to place tasks during the processing of a user, and *tasks* is an array containing placeholders for created tasks.

int find_work(task_queue *queues, const int id) is called if the global user queue is found to be empty. The function loops through the queues of other worker threads and checks if they contain tasks. If a task is found the *handle_task()* function is called. *queues* are an array of all the queues of the worker threads and *id* identifies which queue is the local queue of the current thread.

void handle_task(task_queue *queue) tries to remove a task from the provided queue and if successful it identifies the type of the task and starts processing it. Currently two types of tasks are supported, that of channel estimation and data demodulation.

5.7 uplink_user.c

This file contains the signal processing for a user. The functions of uplink_user.c are:

void uplink_user(task_queue *queue, userS *user, task *tasks) is called when a worker thread has removed a user from the global user queue. This function creates the tasks for channel estimation and the tasks for antenna combination and inverse FFT computation for the two slots of a subframe. It then performs interleave, soft symbol demap, turbo decoding, and cyclic redundancy check on all data of the subframe. Channel estimation for a slot needs to be performed before the data can be processed and the data of slot 0 is available before slot 1. The function therefore starts with creating channel estimation tasks (equal to the number of receive antennas times the layers used for the current user) that are placed on the local queue. These tasks are then processed by calling the *handle_task()* function with the local queue as argument (described in Sec. 5.6). When the queue is empty the processing of data can not progress until all channel estimation tasks have been completed. There might be other worker threads that has stolen tasks and are still computing the channel estimation. To assure that all channel estimation tasks has been performed each task has a pointer into an array that is written upon completion of the task. The user thread monitors this array by calling *wait_until_computed()* and when all positions in the array has been written it knows that all tasks are completed and can progress to decoding the data symbols of the slot. As each task has a unique address to write into and the user thread only performs reads no lock or critical section is required for writing or reading the array. Task creation and processing of the data are performed in the same way as described for the channel estimation.

void wait_until_computed(bool *computed, int items) this function is used to assure that all tasks have been computed by monitoring an array *computed* of booleans. Each task will write true into a unique location of the array. By waiting until as many trues have been written to the array as the number of tasks to be computed (*items*) it is assured that all tasks have been computed.

void compute_chest(task *task) is called by *handle_task()* when a task of type channel estimation has been found and performs matched filter, inverse FFT, channel estimation, and FFT on the reference symbol of the task.

void compute_symbol(task *task) is called by *handle_task()* when a task of type symbol data has been found and performs antenna combining and inverse FFT on the data of the task.

5.8 lib/kernels

This directory contains all signal processing kernels that are called for processing the data of a user. The different kernels are:

mf_4.c Matched filter.

chest_5.c Channel estimation.

weight_calc_6.c Weight calculation that calls functions in mmse_by_cholcolve_4xX_complex.c.

ant_comb7.c Antenna combining.

fft_8.c Fast Fourier transform and its inverse.

soft_demp_9.c Soft symbol demap.

mmse_by_cholcolve_4xX_complex.c MMSE weight calculator.

interleave_11.c Interleaver.

turbo_dec_12.c Turbo decoding.

crc_13.c Cyclic redundancy check.

5.9 def.h

Much of the behavior of the benchmark can be controlled by a number of defines that can be set in *def.h*. The defines are:

PARAMETER_MODEL_X defines which model to use for generating the parameters of a user for a subframe. The different models are described in Sec. 5.1. Only one of the parameter models are to be set to something other than zero.

PARAMETER_MODEL_CORRELATION defines if core utilization is to be printed on the consol. This is useful when investigating correlation between subframe input parameters and the core utilization.

DEACTIVATE_CORES defines if cores should be put to sleep or not. If enabled and the active signal in the *pmodel* structure is less than the number of worker threads this define will put cores into sleep if supported by the architecture on which the benchmark is executed.

CREATE_VERIFICATION_DATA defines if data is to be created for verification. Used in combination with *PARAMETER_MODEL_VERIFICATION*.

DETAILED_VERIFICATION_DATA defines if only the final result should be verified (set to 0) or also at various steps throughout the computation of a subframe. Used in combination with *PARAMETER_MODEL_VERIFICATION*.

NMB_SLOT defines how many slots a subframe are to contain. Default is two.

OFDM_IN_SLOT defines how many symbols a slot are to contain. Default is seven, one reference symbol and six data symbols. The fourth symbol is always the reference symbol.

RX_ANT defines the number of receive antennas.

MAX_LAYERS defines the maximum number of layers that a user might use.

MAX_SC defines the maximum number of sub-carriers that exists in a subframe.

MAX_RB defines the maximum number of physical resource blocks that exist in a subframe.

SC_PER_RB defines the number of sub-carriers there are in a physical resource block. This of course depends on MAX_SC and MAX_RB.

NMB_DATA_FRAMES defines how many input data sets to be generated on initialization. Default is data for ten unique subframes.

PROB_NEW_USER defines a probability that can be used when creating users during subframe parameter generation.

PROB_EXTRA_LAYER defines a probability that can be used when creating number of layers to be used for a user.

PROB_MOD defines a probability that can be used when creating which modulation to be used for a user.

DELTA defines the length in microseconds of a subframe. This should ideally be 1000, but is commonly higher for an architecture to be able to keep up with the workload.

MAX_USERS defines the maximum number of users for a subframe.

TASK_THREADS defines the number of worker threads to be used.

5.10 uplink_verify.c

This file is used for verifying the correctness of the different versions of the benchmark. The verification is based on the assumption that the serial version of the benchmark *uplink_serial.c* (see Sec. 5.2) is functionally correct and is used as a golden model. Verification is performed by inserting a number of functions at various stages of the execution of a subframe. These functions are used to either record data or to verify the correctness of the data.

To create data to be used as a golden model the *PARAMETER_MODEL_VERIFICATION* and *CREATE_VERIFICATION_DATA* configurations should be set. If more detailed verification should be performed where intermediate results of a subframe are to be verified as well the *DETAILED_VERIFICATION_DATA* should be enabled. When the data of the golden model has been created the *CREATE_VERIFICATION_DATA* configuration can be set to zero to verify the computation of the benchmark. If *PARAMETER_MODEL_VERIFICATION* is set to zero the verification functions are not called to avoid overheads during normal operation.

6 Working With the LTE Uplink Receiver PHY Benchmark

The benchmark comes with an extensive set of make targets that makes it easy to compile and run different versions of the benchmark on different platforms. To compile the benchmark for the Tilera platform the *TILERA_ROOT* (in the Makefile found in the root of the benchmark) needs to be set to the path of the Tilera multicore development environment (MD). The most useful make targets are:

all Compiles the benchmark for the Tilera architecture.

clean Removes temporary compilation files and the binary.

help Prints the complete help text.

cilk_install Installs the Cilk runtime and tools.

compile_linux Compiles the Tilera Linux kernel. This is required if deactivation of cores are enabled as the interrupt routine needs to be modified to be able to wake up cores. This target is called automatically and should not have to be used manually.

uplink Compiles the benchmark. If no argument is given it compiles the benchmark for the Tilera architecture. If argument *x86=1* is given the benchmark is compiled for an x86 architecture (tested on Mac OS X and Ubuntu, Redhat, and Fedora). If argument *serial=1* is given the serial version of the benchmark is compiled for an x86 architecture. If argument *cilk=1* is given the Cilk version of the benchmark is compiled for an x86 architecture.

clean_uplink Deletes all object files and the executable created during compilation of the benchmark.

objdump_uplink Dumps the uplink executable as assembler.

run_pci Compiles the benchmark and launch it on Tilera.

run_sim Compiles the benchmark and launch it in the simulator.

run_x86 Compiles the benchmark and launch it on the local x86 machine.

run_cilk Compiles the benchmark with the Cilk runtime and launch it on the local x86 machine.

run_cilk_pci Compiles the benchmark with the Cilk runtime and launch it on Tilera.

profile Executes the benchmark on Tilera with oprofile enabled.

profile_view View the data collected during a ”make profile” run.

7 Summary

The availability of the LTE Uplink Receiver PHY benchmark provides the community with a realistic software for LTE mobile base stations and the means to perform research in the area of baseband processing. The benchmarks portability and scalability facilitate execution on multiple hardware platforms with different levels of parallelism. This makes the benchmark suitable for benchmarking different hardware platforms to assess their appropriateness for LTE baseband systems.

References

[1] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception. TS 36.104, 3rd Generation Partnership Project (3GPP), September 2008. `http://www.3gpp.org/ftp/Specs/html-info/36104.htm`.

[2] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding. TS 36.212, 3rd Generation Partnership Project (3GPP), March 2010. `http://www.3gpp.org/ftp/Specs/html-info/36212.htm`.

[3] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation. TS 36.211, 3rd Generation Partnership Project (3GPP), March 2010. `http://www.3gpp.org/ftp/Specs/html-info/36211.htm`.

[4] G. J. Foschini. Layered Space-Time Architecture for Wireless Communication in a Fading Environment When Using Multi-Element Antennas. *Bell Labs Technical Journal*, 1(2):41–59, 1996.

[5] L. Hanzo, S. X. Ng, T. Keller, and W. T. Webb. *Quadrature Amplitude Modulation: From Basics to Adaptive Trellis-Coded, Turbo-Equalised and Space-Time Coded OFDM, CDMA and MC-CDMA Systems*. Wiley-IEEE Press, September 2004. ISBN: 978-0-470-09468-6.

[6] C. S. Park, Y.-P. E. Wang, G. Jöngren, and D. Hammarwall. Evolution of Uplink MIMO for LTE-Advanced. *IEEE Communications Magazine*, 49(2):112–121, February 2011.

[7] A. Paulraj, R. Nabar, and D. Gore. *Introduction to Space-Time Wireless Communications*. Cambridge University Press, 2003. ISBN: 0-5218-2615-2.

[8] POSIX.1c. *Threads Extensions*. Information technology—Portable Operating System Interface (POSIX). IEEE Computer Society, 1995. IEEE Std 1003.1c-1995.

[9] Supercomputing Technologies Group, Massachusetts Institute of Technology Laboratory for Computer Science. *Cilk 5.4.6 Reference Manual*, November 2001. `http://supertech.lcs.mit.edu/cilk/manual-5.4.6.pdf`.