SPECTRUM: A Software Defined Predictable Many-Core Architecture for LTE Baseband Processing

Vanchinathan Venkataramani, Aditi Kulkarni, Tulika Mitra, Li-Shiuan Peh
National University of Singapore
{vvanchi, aditi, tulika, peh}@comp.nus.edu.sg

Abstract

Wireless communication standards such as Long Term Evolution (LTE) are rapidly changing to support the high data rate of wireless devices. The physical layer baseband processing has strict real-time deadlines, especially in the next-generation applications enabled by the 5G standard. Existing base station transceivers utilize customized Digital Signal Processing (DSP) cores or fixed-function hardware accelerators for physical layer baseband processing. However, these approaches incur significant non-recurring engineering costs and are inflexible to newer standards or updates. Software programmable processors offer more adaptability. However, it is challenging to sustain guaranteed worst-case latency and throughput at reasonably low-power on shared-memory many-core architectures featuring inherently unpredictable design choices, such as caches and network-on-chip.

We propose *SPECTRUM*, a predictable software defined many-core architecture that exploits the massive parallelism of the LTE baseband processing. The focus is on designing a scalable lightweight hardware that can be programmed and defined by sophisticated software mechanisms. *SPECTRUM* employs hundreds of lightweight in-order cores augmented with custom instructions that provide predictable timing, a purely software-scheduled on-chip network that orchestrates the communication to avoid any contention and per-core software controlled scratchpad memory with deterministic access latency. Compared to a many-core architecture like Skylake-SP (average power 215W) that drops 14% packets at high traffic load, 256-core *SPECTRUM* by definition has zero packet drop rate at significantly lower average power of 24W. *SPECTRUM* consumes 2.11x lower power than C66x DSP cores+accelerator platform in baseband processing. *SPECTRUM* is also well-positioned to support future 5G workloads.

LCTES '19, June 23, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6724-0/19/06…$15.00
https://doi.org/10.1145/3316482.3326352

CCS Concepts • **Computer systems organization → Real-time system architecture**.

Keywords Time-predictable architecture, LTE, 5G, baseband processing, many-cores, low-power

ACM Reference Format:
Vanchinathan Venkataramani, Aditi Kulkarni, Tulika Mitra, Li-Shiuan Peh. 2019. SPECTRUM: A Software Defined Predictable Many-Core Architecture for LTE Baseband Processing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '19), June 23, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3316482.3326352

1 Introduction

The last decade has seen unprecedented growth in the number of wireless-enabled devices including Internet of Things (IoT) end-nodes and handhelds. These devices expect the newer generation of cellular networks to support their low-latency, high data-rate requirements. Long Term Evolution (LTE) is the current communication standard utilized in the cellular networks and 4G LTE is the fastest connection available for wireless networks. The current LTE lays the foundation for the upcoming 5G networks [16], the specifications of which are being finalized. The vision of 5G is to achieve 1000X throughput improvement and 100 billion connections. Most importantly, there will be strict deadlines on access latency for latency-critical and mission-critical applications enabled by 5G networks such as the tactile Internet, intelligent transportation, telesurgery, and real-time control. These will require ultra-reliable connectivity, low end-to-end latency provided with an extremely high level of certainty [48]. For example, the latency requirement for factory automation applications is between 0.25-10 ms with a packet loss rate of 10^{-9} [53].

The LTE architecture [12] establishes a wireless radio connection between a UE (User Equipment, e.g, mobile phone) and an eNodeB (Evolved Node B) base station from the mobile operator. The eNodeB then connects to the core network and the Internet through the backhaul. The wireless radio communication latency is defined as the packet transmission time between UE and eNodeB and is mainly due to the physical layer communication, which in turn, is the sum of the transmit time at the UE, the propagation delay over the radio

interface, and the processing time at the base station (eNodeB). Our focus is on real-time guarantees for the physical layer processing at the base station.

Existing base stations utilize customized DSP cores in conjunction with fixed-function ASIC accelerators for baseband processing as DSPs by themselves cannot meet the performance [9]. These solutions incur significant Non Recurring Engineering (NRE) costs that cannot be easily amortized. According to Huawei's Wireless Network Market Insight statistics, global mobile operators have about 6 million physical base stations [41], which is orders of magnitude smaller than 5 billion mobile phones and 20 billion IoT devices acting as UEs. Moreover, ASICs do not provide any flexibility in the context of continuously evolving wireless network standards. While FPGAs do offer flexibility (unlike ASICs), they are still hard to program, cannot meet the performance-power-area goals, and testing, verification contribute significantly to the NRE cost. Thus, there is increasing interest from the telecommunications equipment manufacturers to use software programmable solutions that can be easily updated with the changing standards.

Prior studies have attempted to map the LTE physical layer processing at base stations on GPUs [68] or general purpose CPUs [55, 57]. However, these architectures are built with components (processor, caches, networks) that provide high performance at the cost of low time-predictability, making it difficult to meet the real-time latency guarantees as well as throughput constraints [55]. Besides, thermal design power (TDP) of these architectures are rather high, a key concern for mobile operators as temperature control accounts for 45% of total energy [4].

The shortcomings of the existing approaches led us to the design of *SPECTRUM*, a *predictable* many-core architecture tailor-made for cellular base stations. This architecture exploits the abundant parallelism in baseband processing with many lightweight in-order cores, several completely software-controlled on-chip networks (NoC), and software programmable scratchpad memories (SPM). Our concrete contributions are the following:

- We believe *SPECTRUM* is the first many-core architecture to provide holistic software-defined time-predictable design (end-to-end) in every component (compute, memory, NoC) and their interactions (e.g., data transfer between SPMs via NoC); hence it does not need any backup mechanism (e.g., dynamic routing) to handle uncertainties.
- As all the components in *SPECTRUM* are inherently predictable and operate under software control, the worst-case execution time of the baseband computation can be easily estimated through static analysis and hence we can provide guaranteed worst-case latency for processing at the base station for latency-critical applications.
- *SPECTRUM* is completely software programmable in high-level programming languages with standard multi-threaded libraries. The compiler/runtime management layer orchestrates configurations and application execution hiding complexities from the programmer, thereby improving productivity in the face of updates in LTE standards.
- *SPECTRUM* consumes 2.11x lower power than existing C66x DSP cores based platform that is hard to program and relies heavily on custom hardware accelerators for meeting the baseband processing requirements.
- *SPECTRUM* provides guaranteed latency with 0% packet drop, while contemporary many-cores like Intel Xeon Gold (Skylake-SP) incur 14% packet drops at high network traffic.
- *SPECTRUM* avoids hardware complexity by relying on sophisticated software techniques to provide high performance at low power of 24W compared to Skylake-SP at 215W.

2 LTE Baseband Processing

The uplink physical layer baseband processing at the base station transceiver (eNodeB) is one of the most computationally demanding components in the LTE protocol [22, 36]. We describe how the uplink data sent through the wireless transmitter of the UE is processed at the base station transceiver. For details, see [14].

2.1 Overview

For the uplink data signals from the UE to the base station, the LTE standard uses Single Carrier Frequency Division Multiple Access (SC-FDMA) modulation scheme. Figure 1 describes the overview of frequency allocation in LTE base stations. Each LTE base station is allocated a frequency band between 1.25 MHz to 20 MHz, split into 15 KHz sub-carriers (SC). The time domain is split into frames (lasting 10 ms), and each frame is further divided into ten sub-frames. Each sub-frame of 1 ms duration comprises two slots (0.5ms per slot), each containing seven SC-FDMA symbols. This includes six data symbols (SYM) and one reference symbol for channel estimation. The minimum unit allocated to a UE is called a Physical Resource Block (PRB), consisting of 12 sub-carriers (frequency domain) lasting one sub-frame (1 ms in the time domain). As per [14], the frequency spectrum is divided into 100 PRB, totaling 1200 sub-carriers, that can be allocated to a maximum of 10 UEs.

2.2 LTE Uplink Baseband Processing

The current LTE standard requires a throughput of one sub-frame per ms. Assuming 20ms worst-case end-to-end latency for mission-critical applications [53] leaves us with a bound of 2.5ms [14, 27] on subframe processing latency. If a subframe processing time does not meet the deadline, it is

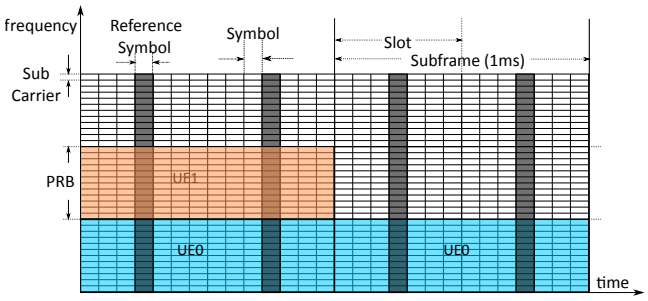

Figure 1. Frequency allocation in LTE

dropped [10] leading to substantial packet loss that cannot provide the ultra-reliable connectivity demanded by latency-critical applications.

We use the LTE PHY benchmark [55] from Ericsson as our baseline reference implementation. The benchmark captures the characteristics of time-varying realistic uplink workloads such as variations in the number of UEs, PRBs allocated to each UE, modulation scheme of each UE, etc. The WiBench project [67] restricts the number of UEs to one and is not suitable for realistic LTE uplink workloads. OpenAir-Interface [10], a standard-compliant open-source software-based implementation of LTE needs to interface with the UEs in real-time and hence cannot be incorporated with our *gem5* [23] architectural simulation environment used to evaluate *SPECTRUM*.

In a typical base station performing uplink baseband processing [55], the radio receiver obtains combined signals from multiple UEs, filters them, removes the cyclic prefix and performs FFT to convert back from time to the frequency domain. In the frequency domain, user extraction recovers individual UE signals and baseband processing is initiated.

Figure 2 shows the baseband processing per UE that has three main components (i) *Channel estimation*, (ii) *Data Demodulation*, and (iii) *Decoding*, each comprising multiple computational kernels. Of these three components, the *Decoding* component with *Turbo Decoding* and *Cyclic Redundancy Check (CRC)* kernels have fixed algorithms that do not change rapidly with each generation of baseband processing. Hence, it is customary to use ASICs for this component to achieve better power-performance efficiency [20, 58]. We thus design *SPECTRUM* to target the other two components.

Channel Estimation. User signals experience distortion and propagation losses during transmission. A weight factor is calculated based on the reference symbol to recover the other symbols in the sub-frame. The weight factor is computed in *Channel Estimation* where for each UE, the reference symbol per sub-carrier undergoes *matched filter* computation followed by *inverse FFT (IFFT)* transforming the input back to time domain. In *windowing*, a subsequence of the time domain samples is extracted. *FFT* is then used to transform the data back to frequency domain. As UE signals are received by different receiver antennae in base stations, the original user signal is recovered by combining signals received from different antennae and adjusting for channel conditions. This is performed in the *combiner weights* kernel. For a MIMO system, channel estimation must be performed for each receiver antenna and layer, the results of which are used to calculate combiner weights.

Data Demodulation. The combiner weights are used to merge the signals from multiple antennae in *antenna combining*. An *IFFT* is then performed to convert the signal back to the time domain. Next, in *de-interleave*, the bits are brought back to the correct sequence. This step is performed as the sequences of bits is dispersed to minimize the effect of burst errors (introduced in transmission) before sending data to the base station. Finally, the original bits are recovered by mapping the output into the corresponding constellation in *soft symbol demap*. Existing modulation schemes include QPSK, 16QAM, 64 QAM and 256QAM. For example, with 64QAM demodulation scheme, each symbol is decoded into six bits.

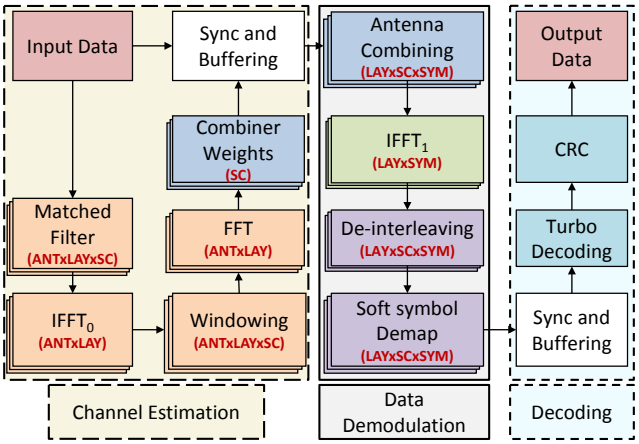

Figure 2. Per-UE data processing flow. The parallelization factor per kernel is indicated within brackets.

2.3 Parallelization Strategies

In LTE baseband processing, after the user extraction phase, processing for different UEs (Figure 2) can proceed in parallel. LTE PHY benchmark from Ericsson [55] parallelizes Matched Filter, $IFFT_0$, Windowing, FFT (MIWF) across *Layers* (LAY) and *Antennae* (ANT), and Antenna Combining and $IFFT_1$ across *Layers* and *SC-FDMA data Symbols* (SYM). We identified additional parallelization factors in each kernel through code analysis. From this analysis, we observe that all the kernels in *channel estimation* and *data demodulation* except FFT and IFFT can be parallelized across sub-carriers (SC). The parallelism factor per kernel is stated within brackets in Figure 2. For example, using maximum values of different parameters as per LTE specifications [15] (SC-1200, SYM-6, ANT-8, and LAY-8), *FFT* has a maximum parallelization factor of 8 ANT X 8 LAY = 64.

3 Architecture Design

We propose *SPECTRUM*, a predictable software defined many-core architecture for LTE baseband processing to achieve the latency and throughput objectives. The streaming subframe data arrives to *SPECTRUM* system along multiple high-speed SerDes (Serializer, Deserializer) links from the RF (Radio Frequency) module. The data flows through and gets processed by *SPECTRUM* cores but never spills to the off-chip memory. After physical layer processing, the data is passed on to the higher layer of the protocol stack.

SPECTRUM differs from conventional shared memory multi-/many-core architectures along multiple dimensions. In shared memory architectures, the hardware automatically brings the data into the caches using an underlying replacement policy. Thus, based on the kernel access patterns, the data may frequently move between on-chip cache and off-chip memory. Additionally, output produced by a kernel is used in subsequent stages. This generates a number of coherence messages, increasing the NoC traffic. Finally, significant overheads and non-deterministic execution can occur due to synchronization with parallel processing. Thus, it is challenging to provide real-time guarantees in conventional shared-memory architectures. In contrast, *SPECTRUM is designed to handle the worst-case workload configuration, ensuring that the latency and throughput constraints are always met. SPECTRUM orchestrates the data placement and movement carefully through software and ensures that the data always stays on-chip.*

An overview of the *SPECTRUM* architecture is presented in Figure 3. The current prototype consists of 256 tiles arranged in a 16x16 2D mesh connected by 8 software-scheduled NoCs. Section 4 explains our architectural parameter choices. Each tile employs time-predictable design decisions in all components including (i) light-weight in-order cores with custom instructions, (ii) software-scheduled network switches with dedicated links for inter-core communications, and (iii) software controlled scratchpad memory for instructions, data. We describe these components and then present our software scheduling approach to ensure efficient use of these resources.

3.1 Lightweight In-order Cores

SPECTRUM deploys lightweight, time-predictable, single-issue in-order cores [2] with local scratch-pad memory (SPM) for both instruction and data. It relies on massive parallelism to meet performance. The input data signals in LTE processing are complex numbers consisting of 16-bit real and 16-bit imaginary components [10, 14, 55]. The baseband processing primarily operates on these complex numbers and involves operations such as complex addition, subtraction, multiplication, scaling, and conjugation and radix-2 butterfly operations. We identify these operations through careful profiling of the kernels and realize them as custom instructions through custom functional units that are added to the processor pipeline (Figure 4). Note that the chosen custom instructions are committed in silicon and are immutable. For example, we add a custom functional unit to compute the product $(a*c-b*d)+(a*d+b*c)i$ of two complex numbers $a+bi$ and $c+di$. This custom unit implemented with four multipliers and two adders, reduces the latency to one integer multiplication followed by one addition. The speedup due to customization varies from 1.5x to 7.5x for different tasks (Section 4.3). As SPM accesses have fixed latency and individual tasks run on single-issue in-order cores, we are able to estimate the worst-case execution time (WCET) easily and accurately [43, 66].

3.2 Software programmable memory

SPECTRUM uses software controlled scratchpad memory (SPM) [19] in each core for on chip instruction and data storage. SPMs have been extensively used as an alternative to caches in embedded systems for their timing predictability, area- and energy-efficiency. An SPM consists of an array of SRAM cells with no tag arrays. Each SPM is mapped to an exclusive portion of the entire memory address space. Thus, given a memory access, there is a one-to-one mapping to a particular SPM. But a core can access data in its own SPM as well as remote SPMs (access latency includes NoC latency), through load/store instructions. However, the compiler should map the data and the instructions in the SPMs across the chip appropriately to ensure the locality of the memory accesses. Given the data, instruction mapping, each memory access has deterministic latency. In the current prototype, we introduce 4-way SPM banking for lower area, power and support multiple writes per cycle boosting throughput. The set of bits selected by Local Bank ID Calculator and Destination Bank ID Calculator to determine the local/destination bank ID are configurable.

For the baseband processing application, the code for each kernel is statically mapped into the instruction SPM. For data, we first map all private/stack variables in each thread's local SPM. Next, we observe that the shared variables within a phase always have read-only accesses. Thus, we replicate each shared variable in the data SPM of all the cores that access that variable.

3.3 Software-scheduled Network-on-Chip

The physical layer processing proceeds in phases where each phase consists of a set of tasks executing in parallel. The different phases are executed in a pipelined fashion on *SPECTRUM* with a set of cores exclusively assigned to each phase. We use double buffering for data communication from one phase to another. We copy all the required data produced in phase i to the SPMs of the cores in phase $i+1$ using software-scheduled NoC with deterministic data transfer cost between SPMs before executing phase $i+1$ computations. This explicit data copy also ensures that all data accesses are serviced from the local SPM.

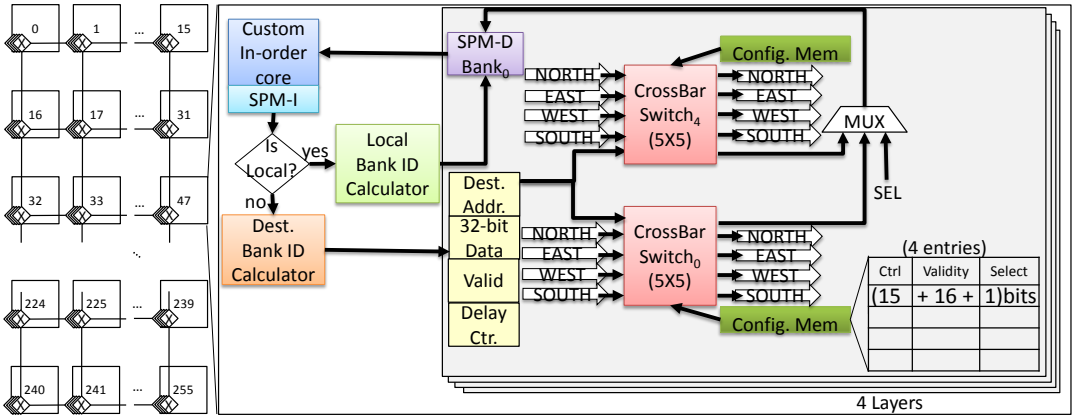

Figure 3. Schematic of *SPECTRUM*: a Software Defined Predictable Many-core architecture

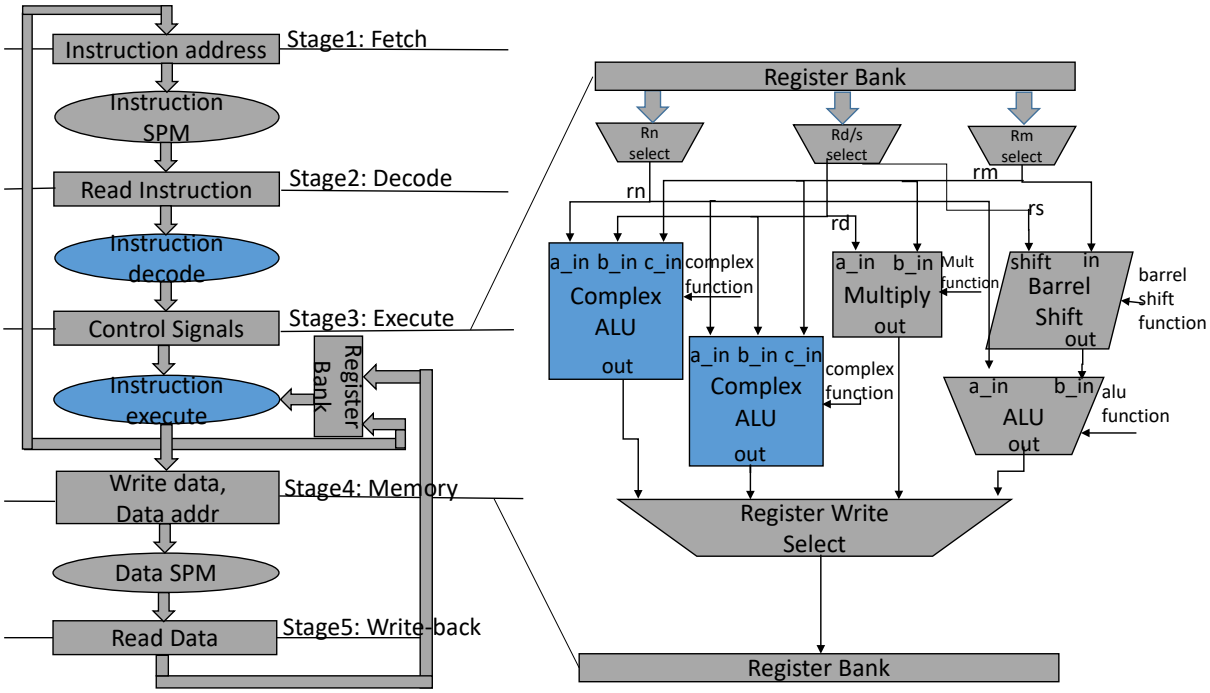

Figure 4. Pipeline of lightweight in-order core with custom instructions. The highlighted components in blue are modified for customization.

The software scheduled NoC with buffer-less switches connected in a flat mesh topology forms the backbone of *SPECTRUM* (Figure 3). The connections settings at each switch are configured by the software scheduler and are stored in the switch's configuration memory. In typical NoCs employing sophisticated routers [45, 56], the data packet header contains the destination address and the router logic determines the crossbar switch's configuration. Multiple buffer queues (called virtual channels [29] (VC)) are typically employed at each router input port to temporarily store the data when there is contention. Such NoCs do not guarantee predictability as contention can lead to packets being stalled. Buffers and routing logic also add complexity and overhead to the NoC. In *SPECTRUM*, as software guarantees contention-free transmission, buffering is not needed within the NoC and time predictable data movement is guaranteed. Different routing protocols are realized by the software choosing alternative routes through NoC, maximizing utilization of the wires. Effectively, this reduces the NoC to its bare minimum in hardware, with just wires and switches. This leads to lower power consumption, opportunities to schedule as per application needs and ease of perform analysis for real-time guarantees.

Switch configuration. The configuration memory contains four entries (Figure 3): $5 * 3 = 15$ control bits, 16-bit validity counter and 1 select bit per entry. The control bits determine the connection of the five input links (North, South, East, West, Packet) to five outputs (North, South, East, West,

SPM), while the validity counter determines the longevity of the connection. The crossbar connections change to the next entry in the configuration memory on expiry of the validity counter. In *SPECTRUM*, each packet is a single flit taking one cycle per hop; thus we use packet and flit interchangeably. The select (SEL) bit in the multiplexer (MUX) determine which switch is connected to the SPM bank. The configuration memory is memory mapped and hence the application configures it only once throughout its lifetime using store instructions based on the network schedule.

Topology. We chose to connect the bufferless switches in a flat mesh topology. A generalized flat mesh enables simple reconfiguration of communication patterns that change dramatically in different versions of baseband processing. Its grid layout provides many alternative paths, so the software is highly likely to find an available route. For LTE and 5G benchmarks, our offline communication synthesis algorithm completes within 60 seconds (executed on single-core in Intel Xeon Gold 6126) and never fails to find an available route. Though clustered or hierarchical topology provides better average latency than mesh, it may not match the actual communication patterns of the application as the cores involved in a phase may not fit within the predetermined cluster size. Section 4.6 shows our software-defined bufferless mesh NoCs taking up just 3% of *SPECTRUM*'s power due to their simplicity, while providing a predictable, configurable fabric for achieving the real-time targets.

Network width. In LTE baseband processing, communicating tasks send the SPM address and 32-bit complex data. If we deploy 32KB data SPM, it can be accessed with 15 bit address. For N-way banked SPM and 4-byte data granularity, 15 - ($log_2$4) - ($log_2 N$) address bits are sent. For 4-way banked SPM, the NoC is 32 + (15-2-2) = 43 bits wide.

Network count. As mentioned earlier, *SPECTRUM* works best with applications leveraging pipelining. Since two phases (e.g. Phase i to phase $i+1$ and phase $i+1$ to phase $i+2$) can send data in the NoC at the same time, contention occurs when two packets need to utilize the same link. To avoid contention, we connect two switches to each SPM. Thus *SPECTRUM* deploys 4 banks and 8 networks, so that different phases can use dedicated channels for simultaneous data transfers. For example, tasks in phase 1 can send data to phase 2 through networks $4-7$ while phase 2 can send data to phase 3 through networks $0-3$.

3.4 Inter-phase communication scheduling

We describe the communication pattern seen in baseband processing and present our network scheduling strategy.

3.4.1 Application model.

In baseband processing application, we define kernel as the smallest computational component. For example in Figure 2, the kernels are stated within rectangular boxes, i.e. Matched Filter, $IFFT_0$, etc. A number of kernels are combined to form a task, which are executed sequentially in a core. Since, parallelism is present in this application, tasks containing the same set of kernels can execute in parallel as there are no dependencies. Thus, we group tasks executing the same set of kernels, but operating on different input data as a phase. Section 4.3 states the details on how we group kernels to form tasks and the number of tasks executed in a given phase.

3.4.2 Task mapping.

To confine data traffic within a NoC region, tasks belonging to a given phase and consecutive phases are placed in adjacent cores. Since two switches are connected to an SPM, contention resolution can be separately solved for tasks within a phase. For example, in Figure 5 (a), tasks belonging to two phases $P_2 : P_{20}, ..., P_{25}$ and $P_3 : P_{30}, ..., P_{33}$ are placed on consecutive cores. Tasks in P_2 utilize two networks (denoted as B0 and B1) for communication. If there was another phase P_4, tasks in P_3 would have used network banks B2 and B3 for sending data.

3.4.3 Communication patterns.

In our application, each task has initial computation followed by communication per iteration. As the phases are pipelined, the output produced from compute in each iteration may need to be sent to the next phase. This communication latency can be hidden if it is less than compute latency. Let S denote the set of source tasks that sends data to destination tasks D. As all the tasks in a phase execute the same code (but operate on different data), each $s \in S$ writes one or more data to a given $d \in D$ sequentially until all destinations are covered.

3.4.4 Objective of Communication Scheduler.

The software scheduler defines NoC routing by setting the NoC switch's configuration appropriately (see Section 3.3). The scheduler needs to ensure that (i) the packets reach the correct destination, (ii) there is no network contention, and (iii) the packets incur low latency.

3.4.5 Approach

The software scheduler computes the configuration memory settings utilizing the communication pattern, arrival cycles, and YX routing. All sources are clock synchronized to send the data in the same cycle. Unique routing paths, dedicated channels for communication across phases and delayed injection guarantee contention-free routing in buffer-less crossbar switches. Given a memory request, we first determine if it is a local or remote access (Figure 3). For remote access, we create a packet with the data to be sent, the destination address, valid bit and counter value. The packet is injected into the network when the counter reaches the 16-bit delay counter value set using a special instruction.

3.4.6 Contention Resolution.

Definition 3.1. Contention is present in NoC link l, if two or more data packets access l at the same time.

For example, in Figure 5 (a), sources P_{20}, ..., P_{25} utilize two networks (denoted as B0 and B1) to send data to each of the destinations P_{30}, ..., P_{33}. Figure 5 (b) shows the number of hops required by the source threads to reach P_{31} under YX routing. Here P_{22}, P_{24} contend at the destination as they both want to write to the SPM in the same cycle. In general, two packets P_m and P_n sent at the same time from sources m and n respectively, can never contend in the NoC if they take a different number of hops to reach destination d.

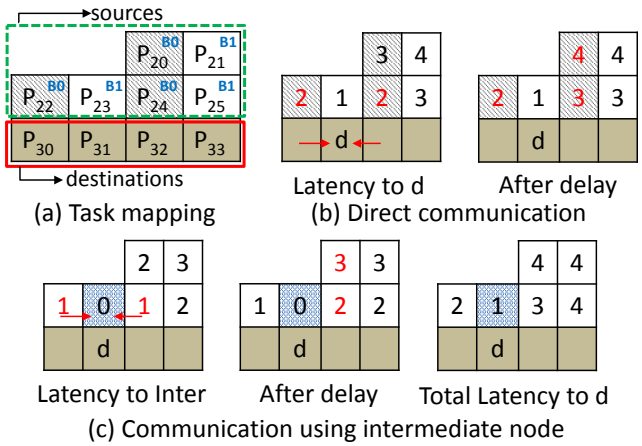

Figure 5. Resolving network contention using delayed packet injection.

Lemma 3.2. *Any packet arriving at switch s will follow the same path and number of hops to reach destination d under the deterministic YX routing protocol.*

Proof. Each switch in a 2D mesh can be represented as (i, j) with i denoting the row id and j denoting the column id of the switch. In YX routing, packets first traverse along the Y-axis to reach the destination's row. Next, they traverse along the X-axis to reach the destination. Based on this, every packet that reaches switch s will follow the exact path to reach destination d. □

Theorem 3.3. *Two packets P_m, P_n sent at the same time from sources m, n can never contend in the NoC if they take different number of hops to destination d.*

Proof. Let l_{md} be the number of hops to reach d from source m. Let l_{nd} be the number of hops to reach d from source n. Let switch x be the first point at which packets P_m and P_n contend. Any packet from x to d will follow the same path and number of hops under YX routing protocol (Lemma 3.2). Let this value be $dist$. As m and n start at the same time, they can contend only if $(l_{md} - dist) = (l_{nd} - dist)$ (By Definition 3.1).

$\implies l_{md}$ equals $l_{nd} \implies m$ and n take the same number of hops to reach d. □

We utilize Theorem 3.3 to design an algorithm that delays packet injection and guarantees contention free routing (Algorithm 1). We first find the ideal latency between each source and destination pair (hops under YX routing scheme) and add them to the priority queues (one SPM per bank ID at the destination). As stated in Theorem 3.3, packets sent with the same bank should reach the destination at different cycles to avoid contention. Thus, for each bank id, we examine the entries in the priority queue and increase the latency if any other source reaches the destination in the same cycle. This value is denoted as computed latency. Finally, the delay value is obtained as the difference between computed latency and ideal latency. In Figure 5 (b), P_{24} is delayed by one cycle as it contends with P_{22}. P_{20} is next delayed by one cycle as it now conflicts with P_{24}.

Algorithm 1: Packet injection delay computation

Input : S - set of sources, d - destination, NB - number of SPM banks
Output: Delay - packet injection delay per source

```
1  //Initialize priority queue PQ with size NB
2  foreach s in S do
3      bid ← getBankID (s);
4      hops ← getLatency (s, d);
5      PQ[bid].push(hops, s)
6  end
7  for bid ← 0 to NB do
8      max_val ← −1;
9      while PQ[bid] ≠ ∅ do
10         (hops, s) ← PQ[bid].top();
11         Delay [s]← MAX(max_val, hops) − hops;
12         max_val ← MAX(max_val, hops) + 1;
13         PQ[bid].pop();
14     end
15 end
16 return Delay;
```

Although Algorithm 1 can be used to calculate the delay counter values, having one delay value that works for every destination increases the communication latency dramatically. Hence, we may be required to store the delay counter values for each destination core in every source's SPM. Alternatively, we may also run the algorithm online to calculate the delay value every time the destination changes. But this approach has huge timing overhead. Inspired by Valiant Load Balancing [63], to ensure that one injection delay value (per task) works across destinations, we propose that all the sources belonging to a phase send the data first to an intermediate core. For an application in which tasks belonging to consecutive phases are mapped on adjacent cores, the intermediate node is chosen to be the core that is closest to

the middle column (amongst the set of source cores that has the largest row ID). The data packets are then re-routed from here to the appropriate destination. Thus, tasks belonging to different phases have different intermediate nodes. In Figure 5(c), the intermediate core for communication between tasks in phases P_2 and P_3 is chosen to be P_{23}. Note that as stated in Section 4.3, the total latency when using an intermediate core may be slightly more than directly sending data to the destination (Figure 8(c)).

With this strategy, the crossbar switches for all sources are configured to receive from the south port to the north port except in case of the last row in the source core set. The cores to the left of the intermediate node are set to send data from West->East while the cores to the right of the intermediate nodes are set to send data from East->West. Once the data is received by the intermediate node, it is routed to the destination. At the destination, switch configuration is (North/South/East/West)-> SPM. Thus, only four entries are required in the switch's Config. Memory. Note that our algorithm receives only one packet per cycle. Algorithm 1 has linear-time complexity on the number of destinations. The configuration of the NoC and the SPM are performed only once per application task graph when the system is installed in a basestation and/or when there is an update in the LTE standard.

In baseband processing application, all the tasks within a phase send data simultaneously to a destination. Consecutive data from a source can only be sent when all the previous packets have reached the destination. Given this communication pattern, we claim that Algorithm 1 provides a data injection schedule that yields minimum latency under fixed YX-routing scheme.

3.5 Run-time Power Optimization

We describe the mechanism utilized in saving power at run-time within the core and the NoC.

Core. Computations in tasks are based on the subframe input. With pipelining, each core starts execution at a regular interval and finishes within the pipeline cycles, irrespective of the input. Thus, cores can be woken up at the pipeline interval and put to sleep mode immediately after computations are completed, saving power.

NoC. When tasks produce data packets, they set the valid bit and write data into the buffer. The packet is then injected into the network and the valid bit is reset when the counter value is reached. The NoC switches are gated and utilized only when packets are produced.

3.6 Software Toolchain

SPECTRUM is supported by a software tool-chain (Figure 6) that automates the mapping of the baseband processing workload to a large extent. Within a generation (4G), updates (e.g., change in modulation scheme to 256 QAM [15]) would involve simply mapping the new task graph to *SPECTRUM* through our tool chain. Across generations (4G to 5G), if performance cannot be satisfied, we need to use higher clock frequency and/or map computations corresponding to different UEs (that are completely independent) within a sub frame to different chips (baseband processing is embarrassingly parallel across users without any dependencies). Details on LTE application mapping using this toolchain are presented in Section 4.3.

(1) **Task Graph.** Whenever there are updates or new communication standards, the task graph is generated for the baseband processing specification (application model described in Section 3.4.1). The throughput is determined by the execution cycles for the longest phase in the pipeline, while the latency is determined by the summation of the execution cycles of the individual phases. The minimum pipeline cycles that are required to meet the performance requirement is obtained using the number of phases, latency and throughput requirements. Note that the complex number operations are realized using custom instructions.

(2) **Phase Profiling.** For each phase in the application, per-task memory profiling is performed for local SPM allocation and obtaining communication patterns (using variables accessed by consecutive phases), followed by per-thread WCET computation [59]. Based on memory profiling, data and instructions are allocated in SPM appropriately (details in Section 3.2). Next, the minimum parallelization factor, i.e., the number of cores allocated per phase, that provides reasonable slack for communication while meeting performance requirements is determined.

(3) **Task Mapping.** Next, we map the tasks in sequential order of phases. In each phase, tasks are first mapped onto cores belonging to the same row (from column zero) in the 2D mesh. Cores present in the next row are utilized when there are no more cores available in that particular row. Thus, tasks belonging to the same phase are mapped in close proximity. We follow the same mechanism for the other phases by continuing from the next unallocated core.

(4) **NoC Scheduling.** The intermediate node is first identified as the core that has the largest row ID and is closest to the middle column. Next, algorithm 1 with intermediate node strategy, takes in the communication pattern (obtained using memory profiling) as input to generate the values for switch configuration memories and delay to attain contention-free NoC routing. Finally, instructions for configuring network switches and Bank ID are generated. Note that the deterministic latency for SPM-SPM data transfers that cannot be hidden behind computation cycles with double buffering is added to the worst-case execution cycles for a phase.

(5) **WCRT.** Finally, the Worst-case Response Time is calculated to validate that the application mapping can meet the latency and throughput requirements.

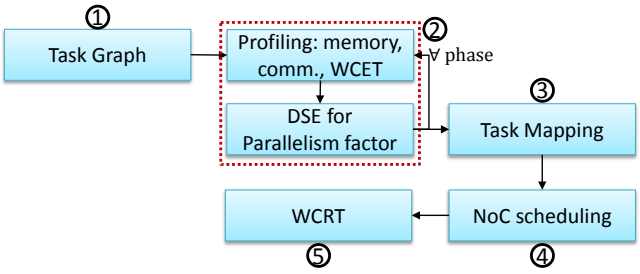

Figure 6. Software Tool chain for application mapping on *SPECTRUM*

Example. Figure 7(a) shows a scaled down version of the LTE baseband processing task Graph containing four phases $P_0 - P_3$ (4, 2, 6 and 4 tasks). This task graph is mapped onto consecutive cores (Figure 7(b)) present in a 4x4 tile, each containing two SPM banks connected using four software scheduled NoCs after phase profiling step. As only consecutive phases communicate with each other in pipelined execution, we configure tasks in Phases P_0, P_3 to receive data from networks 0,1 and tasks in Phase P_2 to receive data from networks 2,3. Figure 7(c) shows the NoC schedule when P_{23} is used as the intermediate node for communication from $P_2 \rightarrow P_3$.

4 Evaluation

4.1 Simulation methodology

To ensure tractability and fidelity of simulating *SPECTRUM* running the LTE benchmark, we took a two-step approach: (i) implement and verify the functionality and correctness of each micro-architectural component and obtain traces, (ii) the traces are then fed into a scalable event-based simulator that models all hardware components. As *SPECTRUM* is software orchestrated and every hardware component is predictable, trace-driven simulation faithfully models the architecture and application.

Micro-architectural Simulation. To validate the functional correctness of our simulator while keeping the simulation time reasonable, we realize a 4x4 version of *SPECTRUM* on *gem5* [23] micro-architectural simulator. The *gem5* simulator was modified to handle custom instructions, SPM and software scheduled NoC. Thus, different system components like light-weight in-order core (ARM ISA, MinorCPU, single issue) with custom instructions, 32KB data SPM, 16KB instructions SPM, crossbar switch based NoC routing are accurately modeled and verified on Syscall Emulation (SE) mode. The task execution cycles, the timing behavior of hardware components like memory access, NoC hop latency, etc. are obtained from *gem5*. These values are then used to build the trace based simulator, which was verified cycle by cycle against the *gem5* simulation.

Trace-based Simulation. Having predictable software defined components allow us to rely on a trace-driven simulation that faithfully models the architecture and application in a reasonable time. We implement a priority-queue based event-driven simulator to model the application execution and communication.

LTE Specifications. In this work, we target the requirements from Section 2.1 of LTE standard [15]. As per this standard, the base station should have the capability to handle 10 users at a time. Each user can be encoded using different modulation schemes namely: QPSK, 16QAM, 64QAM and 256QAM. We assume that UE and radio receivers in base stations have 8 antennae each and thus use 8x8 MIMO. Each user can have up to 100 Physical resource Blocks. Each PRB consists of 12 subcarriers leading to a total of 1200 subcarriers.

LTE Traffic Generation. We evaluate the performance of *SPECTRUM* on realistic subframe workload by combining user activity traces from mobile applications and channel quality indicator estimates from ns-3 [47]. The mobile traffic trace was obtained by tracking background network traffic on an Android phone with no user interaction, while the foreground traffic was generated by allowing random users to upload files of different sizes at random intervals. The packet traces were obtained for over 16 hours. Finally, the network traces and channel quality indicators were used in a Proportional Fair algorithm [21] to produce per subframe network schedule. We consider different traffic situations: (i) Low load with an average throughput of 81.03 Mbps, (ii) High load with average throughput of 445.26 Mbps. We also evaluate performance on a synthetic worst-case workload with the highest throughput of 774.4 Mbps [12, 42]. This workload allocates all 1200 sub-carriers to one UE per subframe, 8x8 MIMO, 256QAM and introduces the maximum computation cost.

Evaluation metric. We define drop rate as the percentage of subframes that cannot be processed as the time from arrival until processing completion is greater than the latency requirement. For LTE network standard, the latency requirement is 2.5 ms.

4.2 Comparison with alternative platforms

Current state-of-the-art baseband processor deployments based on ASIC or DSP cores are proprietary and detailed information is not publicly available. Thus, we rely on qualitative comparisons like power consumption and flexibility in handling newer standards, when details are unavailable for ASIC and DSP. Additionally, we perform a thorough evaluation against commodity Intel Xeon Gold 6126 (Skylake-SP) many-core architecture. We chose Skylake-SP to show that even a server-grade platform may not be able to meet the performance requirements due to unpredictable components present in the system.

Application specific architectures. For stand-alone base stations, Alcatel-Lucent 9926 Base Band Unit with ASIC [1] has 370W TDP for the entire system including the transmitter and receiver power and conservatively 185W for the

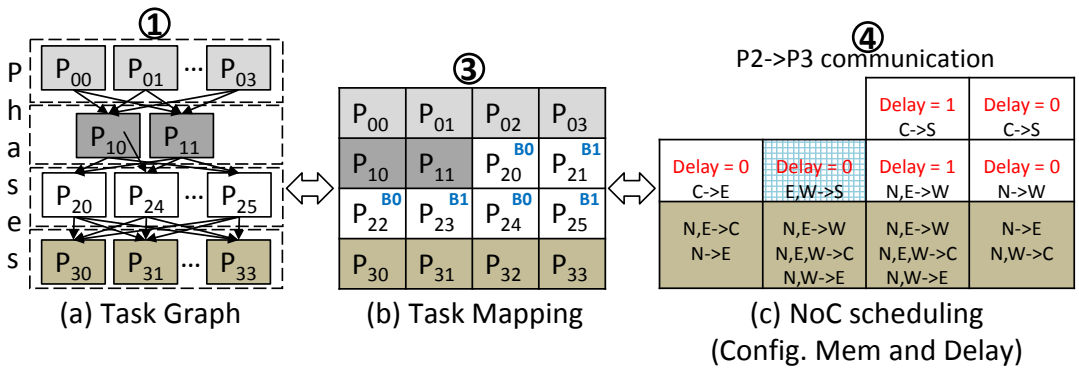

Figure 7. Scaled-down Task mapping example

receiver. However, ASICs have non-recurring engineering costs as they are non-programmable leading to a high cost for upgrades. FPGA based solutions for baseband processing [3, 5] have good flexibility. However, their adoption by industry in base stations did not gain traction due to lower power-efficiency and relatively high cost [13, 68]. Also, FPGAs suffer from programmability challenges.

A number of DSP based PHY deployments exist [6–8, 11, 33]. DSPs are difficult to program [50]. and rely heavily on accelerators for key kernels [6, 8, 33], restricting flexibility. For example, *QorIQ Qonverge B4860 platform* [8] (30W power) deploys additional accelerators when compared to the older version for matrix inversion, multiplication to meet the performance needs. On the other hand, [7, 11] are deployed to support smaller network ranges like Femto, Micro or Pico cells and 3GPP/LTE standards compared to *SPECTRUM* that supports 8x8 MIMO with 4G/5G.

Table 1 states the performance of DSP only and (DSP + accelerator) platforms [33] for downlink processing with 4x4 MIMO [54]. Downlink processing has significantly lower computational complexity than uplink processing targeted by *SPECTRUM*. Hardware accelerators are crucial to meet performance for DSPs and DSP+accelerator platform consumes 2.11x higher power than *SPECTRUM*.

Programmable architectures. To compare *SPECTRUM* against contemporary many-core architectures, we have implemented completely parallelized LTE baseband application on Intel Xeon Gold 6126 (Skylake-SP) and evaluated the performance using the different LTE traffic loads (specs available in Table 2). Table 2 shows that the Skylake-SP platform suffers up to 14% drop rate on real loads and have 100% drop rate for the worst-case load, as opposed to *SPECTRUM* with 0% drop rate. Besides, Skylake-SP also consumes 215W power, as opposed to 24W for *SPECTRUM*. Moreover, cache-based many-cores (Skylake-SP) show huge variations in execution time, even when running constant workload. We demonstrate this timing unpredictability by utilizing the worst-case workload trace. We modify the subframe arrival rate to 100 ms, to provide sufficient time for Skylake-SP to complete processing. In this experiment, Skylake-SP takes between 2–80ms per subframe, with more than 6.6% subframes missing 2.5 ms latency. With a faster arrival rate of 1 ms, Skylake-SP drop rate reaches 100% due to queuing delay between the arrival of a subframe and processing completion (Table 2). In contrast, SPECTRUM has zero execution time variation for constant workload and 0% drop rate.

[24, 68] studied the feasibility of using GPUs for physical layer baseband processing with extensive parallelization and achieve high GPU utilization. [68] showed that GTX680 GPU containing 1536 CUDA Cores at 1 GHz, consumes an average power of 188W at 28nm process technology for 75Mbps uplink data rate, i.e., 1x1 MIMO. *SPECTRUM*, on the other hand, supports 8x8 MIMO with 256QAM achieving 774.4Mbps uplink data rate.

4.3 LTE standards changes and updates

We now use the tool chain in Figure 6 to perform design space exploration and obtain the architectural parameters of *SPECTRUM* that meets the requirements with the least power consumption.

Core: Custom instructions. We implemented six common complex number operations in *gem5* simulator as explained in Section 3.1. Phases *MIWF*, *CWAC*, *IFFT*, and *DD* have a speedup of 4.4x, 7.5x, 4.6x and 1.5x when compared to a standard in-order core, due to custom instructions.

Core count and frequency. The LTE baseband processing workflow (Figure 2) is mapped into four phases. Phase 1 consists of Matched Filter, $IFFT_0$ (IFFT), Windowing and FFT (MIWF). Phase 2 is composed of Combining Weights and Antenna Combining (CWAC). Phase 3 is $IFFT_1$ while Phase 4 is De-interleaving and Soft Symbol Demap (DD). We group these kernels together as they can be parallelized along similar parameters. We parallelize CWAC and DD along subcarriers comprising multiple UEs. However, for IFFT and MIWF, we parallelize and execute each user one after another.

The slots have an arrival rate of 0.5ms. Every subframe has a hard 2.5ms real-time deadline and baseband processing needs to achieve a throughput of 1ms. As we employ pipelining with four stages, each phase needs to finish within 0.5ms. We find the minimum number of threads per phase

Table 1. Power-performance Comparison for 4x4 MIMO

	SPECTRUM	C66x DSP Cores	C66x DSP Cores + Accelerators
#Cores	66	16	4
Accelerators	None	None	Six
Latency (ms)	2.5	13.95	2.15
Frequency	0.9 GHz	1 GHz	1.3 GHz
Area (mm2)	64.68	103.68	25.9 + HW Accel (N.A.)
Avg. Power	6.1 W	17.28 W	12.9 W
Process Technology	40nm	40nm	40nm

Table 2. Power-performance Comparison

	SPECTRUM	Skylake-SP
Low Load (Drop-rate)	0%	1%
High Load (Drop-rate)	0%	14%
Worst-case Load (Drop-rate)	0%	100%
Avg. Power	24 W	215 W
#cores	256	48
Frequency (MHz)	900	2600
Process Technology	40nm	14nm

that meets this requirement, by obtaining the execution time for different parallelization degrees using WCET analysis assuming 1.5GHz maximum system frequency. Figures 8(a) and 8(b) show the reduction in execution time with an increasing number of threads in the four phases. From this exploration, we observe that Phase 1 requires a minimum of 64 cores to meet the 0.5ms deadline while Phase 2, Phase 4 require 75 threads each. Phase 3 requires 24 threads.

A total of 238 cores is required for baseband processing. Thus, we decide on a 16x16 configuration in *SPECTRUM*. The unused cores are power gated. We map and run MIWF threads on cores 0-63, CWAC threads on cores 64-138, IFFT threads on cores 144-167 and DD threads on cores 176-250. We next find the lowest frequency that can achieve the real-time target. From Figures 8(a)(b), 900 MHz is the lowest possible frequency that is able to meet the deadline.

Uncore: SPM and NoC. We map data and instructions to local SPM, using replication of read-only variables to ensure that there are no off-chip or remote accesses. Every destination receiving data from the previous phase employs double buffering. Thus, a destination continues to perform computations for slot i while source sends data for slot $i+1$. In case threads require space more than local SPM size, dynamic overlay technique may be applied. Moreover, tasks may also be split into multiple threads to fit within local SPM.

Reduction in communication latency is primarily related to the number of concurrent data transfers and memory banks. Figure 8(c) shows the communication latency for 2, 4, 8, 16 networks in *MIWF* and *IFFT* phases. The communication latency in the other two kernels can be overlapped while the computation of the next iteration takes place. We choose four SPM banks with dedicated send and receive networks (in total 8 networks) to meet the deadline.

NoC scheduling algorithm: Figure 8(c) shows the communication latency when using an ideal zero contention based scheduling algorithm and the proposed intermediate destination-based algorithm. The ideal delay was individually found for each source, destination pair. From this figure we see that the proposed algorithm using intermediate destination has network latency close to the ideal approach. The total communication cycles spent in phases MIWF and IFFT consume 6.3% and 8.4% of the total pipeline cycles, while the total communication cycles spent in the ideal approach in phases MIWF and IFFT consume 5.3% and 8.1% of the pipeline cycles respectively. Our simple intermediate destination algorithm not only restricts the number of configuration memory entries in a switch to four but also enables *SPECTRUM*'s offline communication synthesis to complete within 60 seconds.

Configuration Setup. Loading instructions in SPM and Config. Mem. is one-off for the entire lifetime of a baseband processing standard. The NoC crossbar switch connections loaded in Config. Mem. changes from one configuration to another based on an automata. Thus, run-time configuration takes zero cycles.

Overall runtime. We configure the system parameters obtained from design space exploration and obtain the computation and communication latency for the different phases running at 900 MHz. Figure 8(d) presents the overall execution time of each of the phases in baseband processing, which meet the throughput and latency constraints.

4.4 System Utilization

Figure 9 reports an average system utilization of 24.41%, 49.31% for low and high network loads. The worst-case load with 774.4 MBps data rate has an average system utilization of 64.7%. The highest core utilization comes from MIWF phase (0-63). This is because, MIWF phase executes FFT and

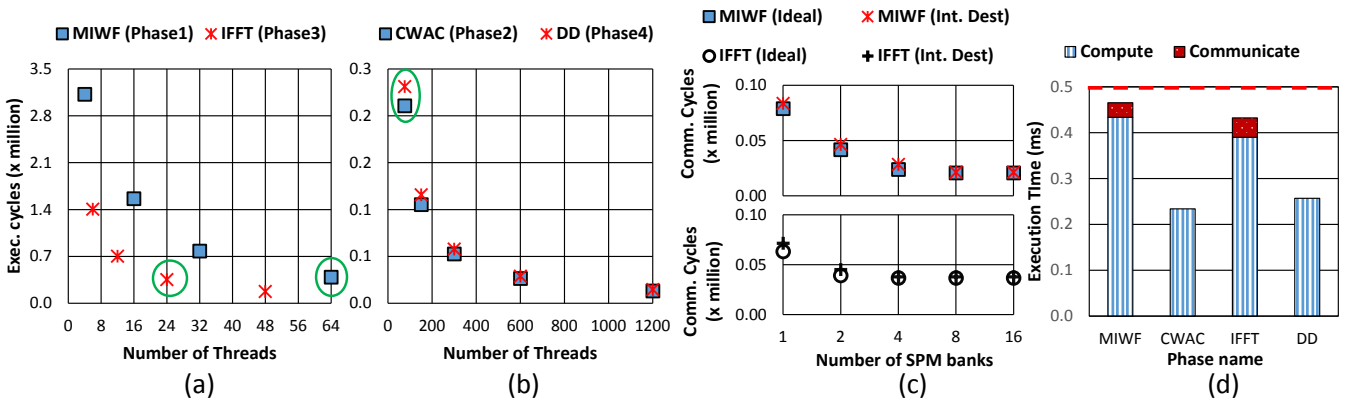

Figure 8. Execution cycles of (a) MIWF, IFFT (b) CWAC, DD phases with varying parallelism with chosen design point circled in green (c) Communication cycles for MIWF and IFFT using different SPM banks (d) Per phase run-time on a 900 MHz system

IFFT kernels, which consume the maximum latency. It has higher utilization than the IFFT phase as it additionally executes Match Filter and Windowing kernels. In low-load scenario, there is variation in core utilization among threads executing CWAC (64-138) and DD (176-250) phases. CWAC, and DD phases are parallelized along subcarriers. Hence, variation in core utilization is attributed by variation in subcarriers allocated per subframe. The *SPECTRUM* parameters are chosen to satisfy worst-case traffic load and hence the drop rate is 0% irrespective of the traffic load. The idle cores are put to sleep to save power.

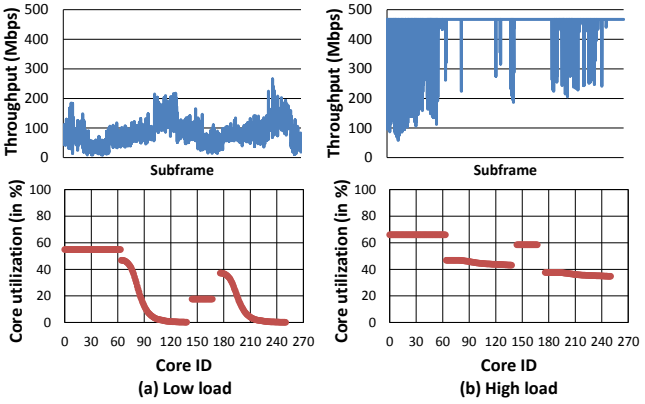

Figure 9. Utilization under different traffic loads

4.5 Scaling to new standards

SPECTRUM can easily be adapted for emerging wireless technologies like *5G*. The *5G* specifications are not yet available. We use one of the proposals for ultra-low latency 5G communication [49] for our evaluation. In this proposal, the slot arrival rate is 0.1ms as opposed to 0.5ms for 4G LTE. A subframe consists of two slots each containing three *OFDM_Symbols* and a total of 560 subcarriers. The data rate is approximately 1.9Gbps. To meet the low-latency and high-throughput requirements, user-level parallelism is additionally exploited in MIWF where each (antenna, layer) combination runs on two cores. In IFFT, each (layer, symbol) pair runs

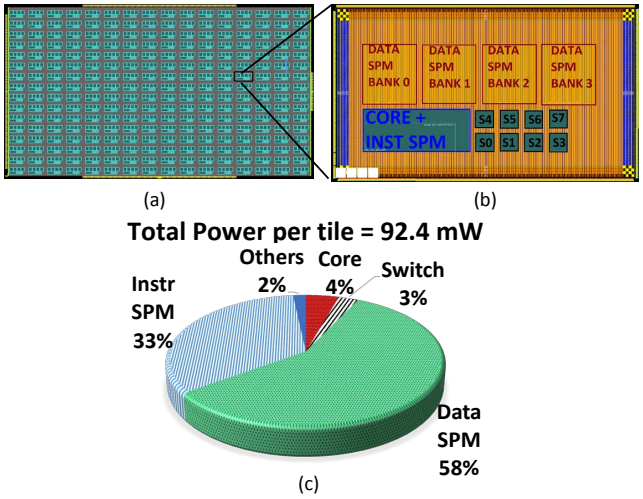

Figure 10. *SPECTRUM* (a) 16x16 floorplan (b) Tile layout (Area per tile is 1.4mm X 0.7mm) (c) Power consumption per tile at 900 MHz, 8 networks, 4 SPM data banks

on individual cores. *SPECTRUM* requires 16x16 tiles running at 1500MHz to achieve the latency and bandwidth requirements with phases MIWF, CWAC, IFFT and DD requiring 128, 75, 16 and 37 cores, respectively. The flexibility offered by *SPECTRUM* allows us to exploit user-level parallelism whenever required to meet the new latency constraints. The compiler transparently takes care of the SPM management and inter-phase communication scheduling.

4.6 RTL design, synthesis and layout

The *SPECTRUM* architecture has been implemented in RTL. Figure 10(a) shows our floorplan comprising 256 tiles arranged in a 16x16 grid. Each tile (Figure 10(b)) consists of an Amber25 processor core[2], 16KB SPM for instruction, 32KB SPM for data and 8 network switches where two networks are connected to each of the four SPM banks of data. Every network switch is bufferless and software-scheduled, and thus is just datapath: a 5x5 crossbar switch with select signals pulled from a 32x4 bits configuration memory. The

configuration memory stores network switch schedules and the associated delay for synchronization and avoiding contention. The RTL design is simulated and synthesized with Synopsys VCS-MX and Design Compiler, with place and route performed using Cadence Encounter. We use Synopsys Prime Time for timing analysis. The width and height of each tile after PnR is 1.4mm x 0.7mm, while the entire 16x16 tiles occupy 22.4mm x 12.3mm.

RTL timing and power analysis. Both the Amber core and the NoC switches are synthesized and carried to layout within the desired frequency of 900MHz. However, the critical path in our design is limited by the Instruction SPM that requires minimum cycle time of 1.26ns as per the 40nm process. We obtained timing and power values of custom-designed SPMs from our industry partner on a commercial 16nm process and use the power numbers by scaling from 16nm to 40nm. These cells have a critical path of 0.66ns meeting our target frequency. The average power per tile with an activity factor of 0.5 is plotted in Figure 10(c). 16x16 tile *SPECTRUM* consumes an average 23.65W power at 900MHz on a commercial 40nm process. The tile power breakdown is: Data SPM 58%, Instr. SPM 33%, Core 4%, NoC Switches 3%, and Others 2%.

Component-wise analysis. The hardware components for compute, memory and communication have been chosen to ensure that there is an end-to-end predictability. We perform component-wise power and area evaluation. The in-order core proposed in this work with custom instructions consumes 6% more area and 5% more power when compared to an unmodified in-order core. There are no changes in the number of register ports as the output of complex instructions are 32-bits long and only one complex operation is performed at a time. Conventional routers with 4 buffers per port have 20% more power and area than crossbar switches proposed in this work with Config. Mem. 32KB Data SPM consumes 39% lesser power and area when compared to direct-mapped caches of similar size containing (17+1) bit tag. 16KB Instr SPM consumes 20% lesser power and 17% lesser area when compared to direct-mapped caches the same size containing (18+1) tag bits. The comparisons were evaluated on 40nm TSMC process at 900MHz.

5 Related Work

Existing architectures. Existing time-predictable single-core architectures include PRET [32] while time-predictable many-core architectures include Kalray [30], T-CREST [51], Merasa[62]. LTE-specific architectures were discussed in Section 4.2. Among many-cores, Kalray [30] requires sophisticated software timing analysis using network calculus for providing real-time guarantees as its interconnect is dynamically routed. T-CREST [51] contains caches apart from SPM supported by software-based coherence leading to difficulty in programmability. Merasa [62] uses non-scalable bus-based shared caches. WCET analysis on multi-core architectures with shared resources [28] has made tremendous progress in recent years but the precision of the results are not sufficient. Whereas, *SPECTRUM* is designed with scalable predictable light-weight components that are easy to program including task, data placement and NoC communication.

Memory management. For the memory hierarchy, cache locking and software-controlled SPM are the approaches used to provide predictability. [18, 31, 35, 44, 59, 61] present optimal scratchpad allocation for program code/data, while [34, 65] explore instruction/data cache locking for WCET minimization. For multi-cores, cache isolation through partitioning is promising [17, 37]. Unlike these approaches, that handle single-threaded applications, [64] defines the exact data placement into each SPM, and controls local/remote SPM access timing through the NoC configuration for multi-threaded applications. However, *SPECTRUM*'s data scheduler is tailored for baseband processing application.

Software-controlled and Real-time NoCs. Software-scheduled NoCs have been proposed in the past for efficiency and quality of service. The MIT Raw on-chip network was the first to use compiler-scheduled flow control in on-chip networks [60], with its successor in the many-core chip from Tilera, TILEPro64, which has the backup of five dynamically routed networks when faced with unknown traffic pattern to aid the software-scheduled, circuit-switched network. *SPECTRUM*, with its domain-specific application of baseband processing, can afford to have all its NoCs to be software-scheduled.

Real-time NoCs have been explored in the past [40]. In Time Division Multiplexing (TDM) hard real-time NoCs like [39], a periodic schedule is obtained where each source node is allocated a time slot in which it is allowed to inject data into the NoC. Distributed TDM real-time NoCs like [52] have a per-router periodic schedule. Periodic schedules lead to wastage of available bandwidth when the application traffic is not perfectly matched to the periodic schedule. *SPECTRUM*'s NoC is fully programmable and controlled by software, so the compiler can generate any schedule for each node and router that need not be periodic. As *SPECTRUM* targets a specific application of baseband processing, we need not resort to periodic slot-by-slot TDM as it has full knowledge of the application flows and can orchestrate the traffic at each router, cycle-by-cycle, for each phase of the application. Hence, as shown in Section 4.3, *SPECTRUM* consumes a maximum of 1% additional cycles than ideal NoC latency while ensuring hard real-time predictability.

The NoC configuration is customized for applications at chip design time with NoC synthesis algorithms [38, 46, 52] that take in worst-case network traffic as input, then fabricated in silicon. In baseband processing, network traffic is heavily dependent on the number of communicating tasks, which changes across different versions.

[25, 26] tackle the problem of worst-case execution time analysis of a conventional wormhole NoC, ensuring predictability when multiple applications share the NoC. Contrarily, *SPECTRUM* is designed for known application of cellular baseband processing, and hence the NoC can be tailored just for the application without worrying about interference between applications.

6 Conclusion

Software programmable processors offer more adaptability with the rapidly changing requirements of wireless network standards but suffer from timing unpredictability. We propose SPECTRUM, a completely time-predictable software-defined many-core architecture that exploits massive parallelism of the LTE baseband processing computation and the streaming nature of its data flow to meet real-time guarantees for all possible workloads. It is power-efficient and consumes 23.65 W for 256 cores.

Acknowledgment

We would like to thank Dr. Raghavendra Kanakagiri for his help in setting up the evaluation platform and the insightful discussions. This work was supported by the National Research Foundation, Prime Minister's Office, Singapore under its Industry-IHL Partnership Grant NRF2015-IIP003 and Huawei International Pte. Ltd.

References

[1] 2009. Alcatel-Lucent 9926 digital 2U eNodeB baseband unit. Alcatel-lucent product brief.

[2] 2010. Amber ARM-Compatible Core. https://opencores.org/project,amber.

[3] 2011. LTE baseband targeted design platform. Xilinx product brief. http://www.origin.xilinx.com/publications/prod_mktg/LTE-Baseband-SellSheet.pdf.

[4] 2011. Temperature Control Solution of Communication Base Station. https://bit.ly/2Bpa9jH.

[5] 2012. LTE baseband targeted design platform. Xilinx product brief. https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/po/wireless-channel-card.pdf.

[6] 2012. Octean Fusion-M CN73XX. https://bit.ly/2TypyW7.

[7] 2013. 66AK2Hxx Multicore DSP+ARM Keystone II SoC. https://bit.ly/2zgPDjO.

[8] 2013. QorIQ ® Qonverge B4860 Baseband Processor. https://bit.ly/2uT6lnp.

[9] 2013. SoC and ASIC Design At Ericsson. https://bit.ly/2TOMLmP.

[10] 2014. Open Air Interface. http://www.openairinterface.org/.

[11] 2016. Transcede t3K Concurrent Dual-Mode SoC Family Communiation Infrastructure. https://intel.ly/2OvK4aY.

[12] 2017. LTE 3GPP releases Overview. https://bit.ly/2DNNnoh.

[13] 2018. Personal Communication with base station manufacturer.

[14] 3GPP. 2017. *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation.* Technical Specification (TS) 36.211. 3rd Generation Partnership Project (3GPP). Version 14.2.0.

[15] 3GPP. 2017. *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.* Technical Specification (TS) 36.213. 3rd Generation Partnership Project (3GPP). Version 14.2.0.

[16] 3GPP. 2018. *Universal Mobile Telecommunications System (UMTS); Base Station (BS) radio transmission and reception (FDD).* Technical Specification (TS) 25.104. http://www.3gpp.org/release-15 Version 15.4.0 Release 15.

[17] Sebastian Altmeyer et al. 2014. Evaluation of cache partitioning for hard real-time systems. In *ECRTS*.

[18] Oren Avissar, Rajeev Barua, and Dave Stewart. 2002. An Optimal Memory Allocation Scheme for Scratch-pad-based Embedded Systems. *ACM Trans. Embed. Comput. Syst.* 1, 1 (Nov. 2002), 6–26.

[19] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, and Peter Marwedel. 2002. Scratchpad Memory: Design Alternative for Cache On-chip Memory in Embedded Systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES '02)*. ACM, New York, NY, USA, 73–78.

[20] Sandro Belfanti, Christoph Roth, Michael Gautschi, Christian Benkeser, and Qiuting Huang. 2013. A 1Gbps LTE-advanced turbo-decoder ASIC in 65nm CMOS. In *VLSI Circuits (VLSIC), 2013 Symposium on*. IEEE, C284–C285.

[21] Paul Bender, Peter Black, Matthew Grob, Roberto Padovani, Nagabhushana Sindhushayana, and Andrew Viterbi. 2010. CDMA/HDR: A bandwidth-efficient high-speed wireless data service for nomadic users. In *The Foundations Of The Digital Wireless World: Selected Works of AJ Viterbi*. World Scientific, 161–168.

[22] Sourjya Bhaumik, Shoban Preeth Chandrabose, Manjunath Kashyap Jataprolu, Gautam Kumar, Anand Muralidhar, Paul Polakos, Vikram Srinivasan, and Thomas Woo. 2012. CloudIQ: A framework for processing base stations in a data center. In *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 125–136.

[23] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[24] Ouajdi Brini and Mounir Boukadoum. 2017. Virtualization of the LTE physical layer symbol processing with GPUs. In *New Circuits and Systems Conference (NEWCAS), 2017 15th IEEE International*. IEEE, 329–332.

[25] Dai Bui, Alessandro Pinto, and Edward A Lee. 2009. On-time network on-chip: Analysis and architecture. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-59* (2009).

[26] Dai N Bui, Hiren D Patel, and Edward A Lee. 2010. Deploying hard real-time control software on chip-multiprocessors. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*. IEEE, 283–292.

[27] Divya Chitimalla, Koteswararao Kondepu, Luca Valcarenghi, and Biswanath Mukherjee. 2015. Reconfigurable and efficient fronthaul of 5G systems. In *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems, ANTS 2015, Kolkata, India, December 15-18, 2015*. 1–5.

[28] Christoph Cullmann et al. 2010. Predictability considerations in the design of multi-core embedded systems. *RTSS*.

[29] W. J. Dally. 1992. Virtual-Channel Flow Control. *IEEE Trans. Parallel Distrib. Syst.* 3, 2 (March 1992), 194–205.

[30] Benoît Dupont de Dinechin, Pierre Guironnet de Massas, Guillaume Lager, Clément Léger, Benjamin Orgogozo, Jérôme Reybert, and Thierry Strudel. 2013. A Distributed Run-Time Environment for the Kalray MPPA®-256 Integrated Manycore Processor.. In *ICCS*, Vol. 13. 1654–1663.

[31] Angel Dominguez, Sumesh Udayakumaran, and Rajeev Barua. 2005. Heap Data Allocation to Scratch-pad Memory in Embedded Systems. *J. Embedded Comput.* 1, 4 (Dec. 2005), 521–540.

[32] Stephen A Edwards and Edward A Lee. 2007. The case for the precision timed (PRET) machine. In *2007 44th ACM/IEEE Design Automation*

Conference. IEEE, 264–265.

[33] R. Damodaran et al. 2012. A 1.25GHz 0.8W C66x DSP Core in 40nm CMOS. In *VLSID*.

[34] Heiko Falk et al. 2007. Compile-time decided instruction cache locking using worst-case execution paths. In *CODES+ISSS*.

[35] Heiko Falk et al. 2009. Optimal static WCET-aware scratchpad allocation of program code. In *DAC*.

[36] Arnon Friedmann and Sandeep Kumar. 2009. LTE emerges as early leader in 4G technologies. In *White Paper*. Texas Instruments.

[37] Nan Guan et al. 2009. Cache-aware scheduling and analysis for multicores. In *EMSOFT*.

[38] Andreas Hansson, Kees Goossens, and Andrei Rădulescu. 2005. A Unified Approach to Constrained Mapping and Routing on Network-on-chip Architectures. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '05)*. ACM, New York, NY, USA, 75–80.

[39] Andreas Hansson, Mahesh Subburaman, and Kees Goossens. 2009. Aelite: A Flit-synchronous Network on Chip with Composable and Predictable Services. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '09)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 250–255.

[40] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany. 2017. Survey on Real-Time Networks-on-Chip. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (May 2017), 1500–1517.

[41] Huawei. [n. d.]. Base Station Operation Increases the Efficiency of Network Construction. https://bit.ly/2GtCd6N.

[42] Yiming Huo, Xiaodai Dong, and Wei Xu. 2017. 5G cellular user equipment: From theory to practical hardware design. *IEEE Access* 5 (2017), 13992–14010.

[43] Xianfeng Li et al. 2007. Chronos: A timing analyzer for embedded software. *Science of Computer Programming* (2007).

[44] Jing Lu, Ke Bai, and Aviral Shrivastava. 2015. Efficient Code Assignment Techniques for Local Memory on Software Managed Multicores. *ACM Trans. Embed. Comput. Syst.* 14, 4, Article 71 (Dec. 2015), 24 pages.

[45] Timothy G Mattson, Michael Riepen, Thomas Lehnig, Paul Brett, Werner Haas, Patrick Kennedy, Jason Howard, Sriram Vangal, Nitin Borkar, Greg Ruhl, et al. 2010. The 48-core scc processor: The programmer's view. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 1–11.

[46] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli. 2006. A Methodology for Mapping Multiple Use-Cases onto Networks on Chips. In *Proceedings of the Design Automation Test in Europe Conference*, Vol. 1. 1–6.

[47] ns 3. 2010. ns-3 Network simulator. https://www.nsnam.org/.

[48] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif I Sarwat, and Huaiyu Dai. 2017. A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions. *arXiv preprint arXiv:1708.02562* (2017).

[49] Klaus I Pedersen, Gilberto Berardinelli, Frank Frederiksen, Preben Mogensen, and Agnieszka Szufarska. 2016. A flexible 5G frame structure design for frequency-division duplex cases. *IEEE Communications Magazine* 54, 3 (2016), 53–59.

[50] Maxime Pelcat, Karol Desnos, Julien Heulot, Clément Guy, Jean François Nezan, and Slaheddine Aridhi. 2014. Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming. In *EDERC*. 36.

[51] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, et al. 2015. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture* 61, 9 (2015), 449–471.

[52] Martin Schoeberl, Florian Brandner, Jens Sparsø, and Evangelia Kasapaki. 2012. A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems. In *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip (NOCS '12)*. IEEE Computer Society, Washington, DC, USA, 152–160.

[53] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. 2017. Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine* 55, 2 (2017), 70–78.

[54] Silexica. 2016. Multi-core Software Design For an LTE Base Station, WHite Paper. https://bit.ly/2TyE7sx.

[55] Magnus Sjalander, Sally A. McKee, Peter Brauer, David Engdal, and Andras Vajda. 2012. An LTE Uplink Receiver PHY Benchmark and Subframe-based Power Management. In *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS '12)*. IEEE Computer Society, Washington, DC, USA, 25–34.

[56] Avinash Sodani. 2015. Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor. In *Hot Chips 27 Symposium (HCS)*. IEEE, 1–24.

[57] Manikantan Srinivasan, C Siva Ram Murthy, and Anusuya Balasubramanian. 2015. Modular performance analysis of Multicore SoC-based small cell LTE base station. In *Very Large Scale Integration (VLSI-SoC), 2015 IFIP/IEEE International Conference on*. IEEE, 37–42.

[58] Christoph Studer, Christian Benkeser, Sandro Belfanti, and Quiting Huang. 2011. Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE. *IEEE Journal of Solid-State Circuits* 46, 1 (2011), 8–17.

[59] Vivy Suhendra et al. 2005. WCET centric data allocation to scratchpad memory. In *RTSS*.

[60] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, et al. 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE micro* 22, 2 (2002), 25–35.

[61] Sumesh Udayakumaran, Angel Dominguez, and Rajeev Barua. 2006. Dynamic Allocation for Scratch-pad Memory Using Compile-time Decisions. *ACM Trans. Embed. Comput. Syst.* 5, 2 (May 2006), 472–511.

[62] Theo Ungerer, Francisco Cazorla, Pascal Sainrat, Guillem Bernat, Zlatko Petrov, Christine Rochange, Eduardo Quinones, Mike Gerdes, Marco Paolieri, Julian Wolf, et al. 2010. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro* 30, 5 (2010), 66–75.

[63] Leslie G. Valiant. 1982. A scheme for fast parallel communication. *SIAM journal on computing* 11, 2 (1982), 350–361.

[64] Vanchinathan Venkataramani, Mun Choon Chan, and Tulika Mitra. 2019. Scratchpad-Memory Management for Multi-Threaded Applications on Many-Core Architectures. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 1 (2019), 10.

[65] Xavier Vera et al. 2007. Data cache locking for tight timing calculations. *TECS* (2007).

[66] Reinhard Wilhelm et al. 2008. The worst-case execution-time problem-overview of methods and survey of tools. *TECS*.

[67] Qi Zheng, Yajing Chen, Ronald G. Dreslinski, Chaitali Chakrabarti, Achilleas Anastasopoulos, Scott A. Mahlke, and Trevor N. Mudge. 2013. WiBench: An open source kernel suite for benchmarking wireless systems. In *Proceedings of the IEEE International Symposium on Workload Characterization, IISWC 2013, Portland, OR, USA, September 22-24, 2013*. 123–132.

[68] Qi Zheng, Yajing Chen, Hyunseok Lee, Ronald Dreslinski, Chaitali Chakrabarti, Achilleas Anastasopoulos, Scott Mahlke, and Trevor Mudge. 2015. Using Graphics Processing Units in an LTE Base Station. *Journal of Signal Processing Systems* 78, 1 (01 Jan 2015), 35–47.