CMA: Chip Multi-Accelerator

Dominik Auras[†], Sylvain Girbal[†], Hugues Berry[§], Olivier Temam[§], Sami Yehia[†]
[†] Thales Research and Technology, France   [§] INRIA Saclay, France
{dominik.auras, sylvain.girbal, sami.yehia}@thalesgroup.com   {hugues.berry, olivier.temam}@inria.fr

Abstract

Custom acceleration has been a standard choice in embedded systems thanks to the power density and performance efficiency it provides. Parallelism is another orthogonal scalability path that efficiently overcomes the increasing limitation of frequency scaling in current general-purpose architectures. The use of custom acceleration in multiprocessor systems has been limited by the programming complexity they induce compared to homogeneous chip multiprocessors. In this paper we propose a multi-accelerator architecture that combines the best of both worlds, parallelism and custom acceleration, while adressing the programmability inconvenience of heterogeneous multiprocessing systems. A Chip Multi-Accelerator (CMA) is a regular parallel architecture where each core is complemented with a custom accelerator to speed up specific functions. Furthermore, by using techniques to efficiently merge more than one custom accelerator together, we are able to cram as many accelerators as needed by the application or a domain of application.

We demonstrate our approach on a Software Defined Radio (SDR) case study. We show that starting from a baseline description of several SDR waveforms and candidate tasks for acceleration, we are able to map the different waveforms on the heterogeneous multi-accelerator architecture while keeping a logical view of a regular multi-core architecture, thus simplifying the mapping of the waveforms onto the multi-accelerator.

1 Introduction

Chip-Multiprocessors (CMP) architectures have become the industry standard of processor architectures during the last decade [26, 18] even in the embedded industry [13]. Technology scaling, the increasing design complexity of single-core architectures and power requirements have led the industry to naturally adopt this path. Homogeneous multi and many-cores offer a substantial programmability advantage thanks to the regularity of their architectures. Still, many-core architectures may not scale on the longer term because the communication overhead, the limited bandwidth, the limited chip power budget and the effect of Amdahl's law will be such that increasing the number of cores will not add to the performance anymore.

Custom acceleration is another complementary path to parallelization which can provide power efficient processing for specific application tasks and is inexorably making its way in general purpose computing because of the efficiency advantage they offer for compute-intensive tasks such as video processing [21], texture logic for graphic processing [26] and cryptographic functionalities [18]. While SoCs embed an increasing number of accelerators, these chips are notoriously difficult to program, which will ultimately limit their scalability and popularity.

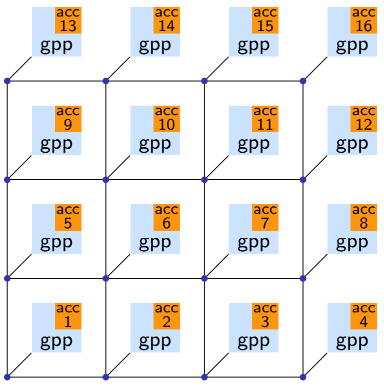


Figure 1: *CMA architecture.*

In this paper we propose to combine the best of both worlds (parallelism and acceleration) by leveraging the efficiency of customization together with the programmability of regular multi-core architectures. A *Chip Multi-accelerator (CMA)*, or a many-accelerator, is *regular* from a user or programming model perspective and *heterogeneous* from an architecture perspective for performance and power efficiency: Figure 1 shows the general concept of the proposed architecture. The building block here is a many-core layout in which a custom accelerator is coupled with a general-purpose processor. The custom accelerators in the different tiles need not be identical and each of them can be tailored to a specific function. The main idea here is that specialization comes after parallelization from a programming perspective. So the accelerators coupled with each processor will accelerate the individual threads of the parallel program.

According to this baseline, a parallel program developed for a homogeneous multi-core architecture having the same fixed infrastructure (same memory architecture, NoC, programming model, etc.) would run transparently on the CMA architecture, though at the cost of not taking advantage of the accelerators, thus widening the application scope of the target architecture. In some sense, the same way general-purpose processor designers find it more efficient to trade additional computational units for cache size, a many-accelerator architecture trades additional cores for accelerators. In fact, with the current technology scaling trends, we can foresee many-accelerators with hundreds of different accelerators where only few of them need to operate at the same time according to the application needs, thus saving substantial energy with appropriate power gating techniques [6].

The nature and specifics of the accelerators on the CMA depend on the nature of the target applications. However for such an approach to become mainstream, accelerators have to be generated in a generic way from the application to lower the non recurring engineering (NRE) costs and allow a reasonable time to market. Loop based accelerators [33, 3, 7, 8] allow to efficiently execute frequent computational patterns while having relatively small footprints compared to FPGA because they only implement specific data paths of the target computations. Compound circuits [33] offer the possibility to merge several loop accelerators into one accelerator with a minimum overhead and loss of efficiency. Compounding therefore offers a scalable way of acceleration on the proposed CMA architecture: for an N-tile CMA, if the target application (or a specific domain of applications) has M loops to accelerate, and M > N, we can appropriately merge more than one circuit in order to fit into the N accelerators' placeholders of the CMA architecture.

To illustrate the CMA concept, we use the Software Defined Radio (SDR) domain of applications as a case study. Starting from several target waveforms and their software implementation, we design a many-accelerator that can efficiently implement the several waveforms on our *regular but heterogeneous* architecture. One essential aspect of the proposed methodology is that the mapping of the different tasks is done as if the application is mapped to a homogeneous multicore architecture. Then, according to the resulting mapping we generate the appropriate accelerators for those tasks that need to be accelerated (in this particular case to meet the real-time requirements of the waveform). Next, thanks to compound circuits, we are able to fit the several accelerated tasks on the available tiles of the CMA. Consequently, for a given "Tile budget", acceleration is done after parallelization and independently from the underlying parallel programming model.

The main contributions of this paper are the following: (1) We combine both parallelism and specialization in a many-accelerator architecture which provides the programmability of a parallel architecture and efficiency of a specialized architecture, (2) we use compound circuits to increase the scope of the CMA architecture in spite of a limited number of tiles and (3) we illustrate the benefits of our approach in a Software Defined Radio case study.

2 A CMA example: SDR

Wireless protocols in embedded systems have particularly stringent requirements in terms of performance and power consumption. Software Defined Radio (SDR) [28] is the concept of performing digital signal processing as software on flexible hardware. SDR enables the implementation of several wireless protocols at the physical layer without the need for a hardware-only (ASIC) implementation for each protocol; and is considered as the enabling technology for future multi-mode and cognitive radios [1].

SDR waveforms inherently consist of parallel (concurrent and pipelined) and sequential tasks (stateful algorithms). In this context, a CMA is a natural solution where the parallel parts can be accelerated with the multi-tile structure, and simultaneously, the sequential parts can be accelerated with the custom circuits within each tile.

In this section we show how to implement the physical layer of two wireless protocols: Orthogonal Frequency Division Multiplex (802.11a [16]) and Wideband Code Division Multiple Access (WCDMA) [14] on a CMA architecture. For each protocol, we consider both transmitter (TX) and receiver (RX) waveforms. We first demonstrate how to implement a single waveform (802.11a TX) on a four-tile CMA architecture. Next, we illustrate the case for two waveforms when more than one accelerator needs to be architected within one tile. Finally we show how to implement the four waveforms on a CMA.

2.1 802.11a TX implementation on a 4-tile CMA

Figure 2(a) shows a 802.11a TX (Transmitter) waveform consisting of 13 pipelined-dependent tasks. Achieving the required real time throughput is the essential constraint (in addition to the traditional power

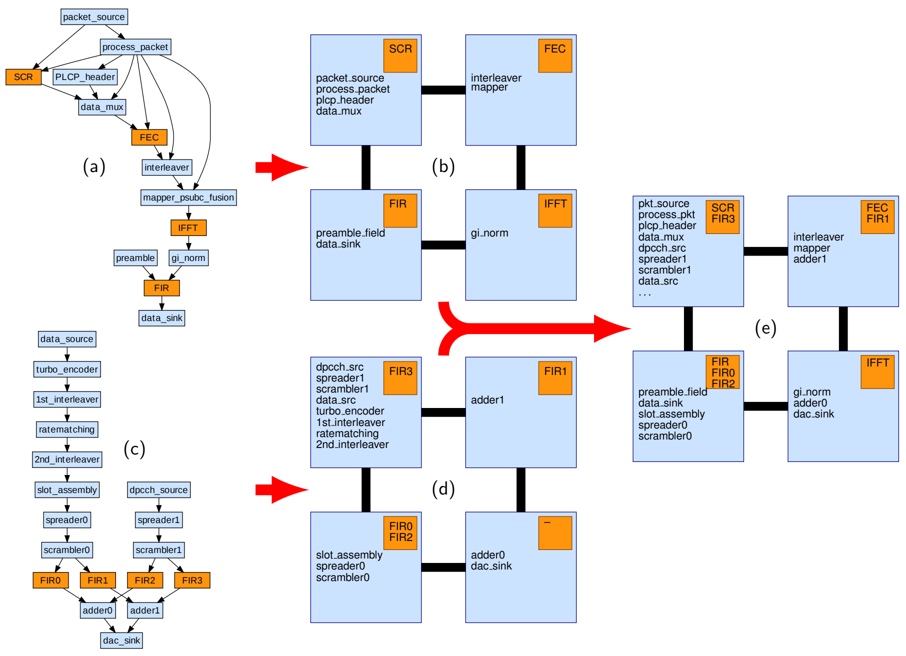


Figure 2: *802.11a tx waveform (a) and mapping (b), WCDMA tx waveform (c) and mapping (d), and final mapping of both transmitters (e).*

budget constraint of embedded systems) any wireless protocol implementation has to fulfill in order to function properly. The required data throughput of the 802.11a TX waveform is 24Mbps. In order to achieve this throughput, each stage of the pipelined waveform must process one OFDM symbol every 4 μs. An OFDM symbol is a finite-length signal representing the modulated bits within a fixed-time slot (4 μs).

On an embedded PowerPC405 running at 650 Mhz, the FEC(Forward Error Correction), FFT and FIR tasks process an OFDM symbol in 6, 19.6 and 61 μs respectively and therefore do not meet the real time requirements.

Those three tasks (shaded in Figure 2(a)) can be accelerated using custom hardware to fulfill the real time and power requirements of the system. Accelerators can be designed by hand or generated automatically from C code using tools such as ROCCC [12], CatapultC [4] or GAUT [27]. In this work we used the tool chain proposed by Yehia et al. in [33] to generate the circuits for each individual task.

The circuits are converted from C to an intermediate representation, and then into Verilog. Critical parts of the circuits were also handcrafted where C language was not expressive enough, e.g., counting the number of ones in a register. In the translation process, array references are converted into stream buffers, and the task is thus converted into a data flow circuit. A major asset of our conversion process is to create a circuit that will efficiently manage data transfers to/from memory or other accelerators, thanks to these streams. These stream buffers come in two sorts: with an address generation unit and counters to automatically fetch strided references, or with an address queue for irregular addresses generated by the rest of the circuit (e.g., indirect addressing). The intermediate representation explicits not only the data flow part of the circuit but also the circuit control. See Figure 3(a) for the circuit representation of 802.11a FFT task.

The task mapping process is iterative. Tasks are first partitioned and mapped to the cores of the CMA as if it were a homogeneous multi-core. The bottleneck tasks are then selected for acceleration, and the corresponding accelerators are generated and placed within a tile, resulting in a mix of software and accelerated tasks. The process is repeated until real time constraints are met. The goal is to minimize the number of accelerators. If the number of necessary accelerators exceeds the number of tiles, accelerators are merged using the compounding process explained in Section 2.2.

The task mapping can be done by hand or using automatic parallel mapping tools such as SCOTCH[23] or METIS[17] assuming a homogeneous CMP but incorporating profiling information *after* acceleration of critical tasks. Figure 2(b) shows the mapping of each task to each of the 4 tiles of the CMA using SCOTCH [23] mapping tools. From a programming perspective, each call to one of these tasks is replaced with a call to the corresponding accelerator as in an ASIP architecture [3].

In this example, the profile after acceleration showed that the resulting mapping could not meet throughput requirements because of many software tasks mapped to Tile 0. We therefore accelerated task SCR on Tile 0 (given the profile information) to meet the real time requirements.

2.2 Mapping several accelerators to one tile using compound circuits

Let us now assume we want to implement a WCDMA TX waveform, as shown in Figure 2(c), on the same CMA architecture. In Figure 2(d), we show a possible mapping of the waveform on the 4-tile CMA architecture where the 4 tasks FIR0, FIR1, FIR2 and FIR3 are identical FIR filters mapped to three of the 4 accelerators of the CMA (in this mapping, FIR0 and FIR2 tasks execute sequentially on one tile).

In order for the 4-tile CMA to support both waveforms (802.11a TX and WCDMA TX), tiles need to contain more than one accelerator. For instance, task SCR of 802.11a TX and task FIR of WCDMA TX are both mapped to Tile 0 of the CMA. One solution is to map both accelerators within the tile, and to activate either one when needed; the processor interface would also have to support both accelerators. However, due to chip area constraints, this solution only enables the implementation of a limited number of accelerators.

An attractive alternative is to aggregate the circuits of several accelerators into a single circuit. While the 802.11a and WCDMA circuits are different, they often contain many similar data flow constructs, e.g., the FIR filter. Instead of implementing such common constructs twice, once in each accelerator, with one being useless in the idle accelerator, while the other one is used in the active accelerator, we create an aggregate accelerator which *factors in* these common constructs. Consider, for example, aggregating FFT (in 802.11a TX waveform) in Figure 3(a) and the Backward State Metric Unit (BSMU in WCDMA RX Waveform) in Figure 3(b) into one accelerator. The input data streams, the adder, the substracter, the output data stream as well as the register file of the FFT are all used by the BSMU as well, and thus can be factored in. The BSMU only requires a few additional operators (MAX, delay, adder and substracters) so the aggregate circuit is complemented accordingly, as shown in Figure 3(c). While manually aggregating accelerators would be tedious, Yehia et al. have shown that it is possible to entirely automate the process [33]: the data flow and control flow graphs of target accelerators are compared using efficient pattern matching techniques, and all redundant elements are discarded. The resulting aggregate circuits are called *compound* circuits, and require much less area than the sum of the areas of the aggregated accelerators. As part of the aggregation process, some multiplexers are introduced to enable the compound circuit to behave as either one of the original accelerators, see *Circuit Select* in Figure 3(c). While these multiplexers induce some area and latency overhead, Yehia et al. [33] have shown that this overhead remains low, even when 9 accelerators are aggregated together. As a result, this aggregation technique significantly increases the scalability of the CMA by allowing to tackle many more tasks than if accelerators are simply added up.

Figure 2(e) shows the result of mapping both waveforms, 802.11a TX and WCDMA TX to a 4-tile CMA according to the mappings of Figures 2(b) and 2(d). In this example, implementing both waveforms on a 4-tile CMA results in three compounds. The lower right tile has only one accelerator (IFFT) belonging to the 802.11a TX waveform because no accelerated task from the WCDMA TX is mapped to that tile.

Implementing remaining waveforms of 802.11a and WCDMA. Table 1 shows the final mapping of the four waveforms, 802.11a TX, 802.11a RX (only decoding state), WCDMA TX and WCDMA RX. The waveform 802.11a RX has two software tasks (implemented on the GPP) and 8 accelerated tasks. The Equalizer and Detector circuit (Equ. & Detct.) is a division-free equalizer implementation, delivering the detected data as soft bits [9]. The interpolator circuit used by 802.11a RX for synchronization on the transmitter's clock is a cubic 8-tap interpolator with a Farrow structure [22].

We implemented the decoding state of WCDMA RX in 45 software tasks and 10 Hardware tasks. The accelerated tasks *Backward State Metric* and *Forward State Metric* are part of the Max-log-MAP decoders inside the Turbo Decoder, computing the backward and forward recursion of the MAP algorithm [29], where the Fwd. State Metric simultaneously computes the forward recursion and the extrinsic Log-Likelihood Ratios (LLR). WCDMA RX's correlator is used in the searcher that synchronizes the receiver to the three base stations [19].

Table 1 shows the area in thousands of gate equivalents (kGE), latency, power consumption and the speedup of the hardware tasks and their mapping on the four tiles. The simulation and synthesis infrastructures are detailed in Section 4.

The 6 hardware tasks of Tile 0 result in a compound

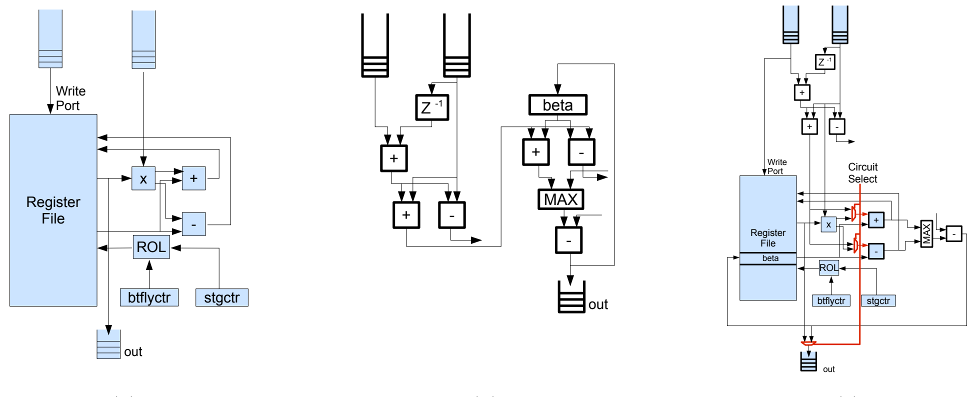


Figure 3: *Compound circuits: (a) FFT circuit, (b) Backward State Metric Unit, (c) resulting Compound*

circuit of 347 kGE (0.98mm^2) using a 90nm TSMC standard cell library for a power consumption of 207 mw. The sum of the area of the 6 individual tasks of Tile 0 is 687 kGE (1.94mm^2), which is twice the size of the compound circuit.

	Tile 0					Tile 1				
	Task	Area [kGE]	Lat. [ns]	Power [mW]	Speedup	Task	Area [kGE]	Lat. [ns]	Power [mW]	Speedup
802.11a TX	Scrambler	12.14	2.99	8.76	2.36	Fwd. Err. Corr.	22.20	3.03	11.40	29.29
802.11a RX	FIR filter	251.80	3.56	81.94	82.54	Interpolator	209.08	4.04	82.17	14.96
WCDMA TX	Ups. FIR filter	286.21	3.26	88.30	25.15	Ups. FIR filter	286.21	3.26	88.30	25.15
WCDMA RX	Correlator	21.93	3.02	14.97	11.02	Correlator	21.93	3.02	14.97	11.02
	Bwd State Metric	44.29	3.64	23.93	3.23	Bwd State Metric	44.29	3.64	23.93	3.23
	Fwd State Metric	71.78	4.33	37.45	8.57	Fwd State Metric	71.78	4.33	37.45	8.57

	Tile 2					Tile 3				
	Task	Area [kGE]	Lat. [ns]	Power [mW]	Speedup	Task	Area [kGE]	Lat. [ns]	Power [mW]	Speedup
802.11a TX	Ups. FIR filter	373.65	4.17	129.93	24.32	FFT	171.27	5.72	93.82	11.69
802.11a RX	Viterbi Decoder	349.21	3.16	174.35	241.52	Mixer	102.08	3.03	52.15	50.41
	Scrambler	12.14	2.99	8.76	2.36	FFT	171.27	5.72	93.82	11.69
	Deinterleaver	15.50	2.98	12.26	2.60	Equ & Detct.	61.25	8.55	28.48	4.48
WCDMA TX	Ups. FIR filter	286.21	3.26	88.30	25.15	-				
WCDMA RX	Correlator	21.93	3.02	14.97	11.02	FIR filter	251.80	3.56	81.94	304.21
	Bwd State Metric	44.29	3.64	23.93	3.23					
	Fwd State Metric	71.78	4.33	37.45	8.57					

Table 1: *Accelerator mapping of 802.11a TX, 802.11a RX, WCDMA TX and WCDMA RX on the CMA*

3 Implementation

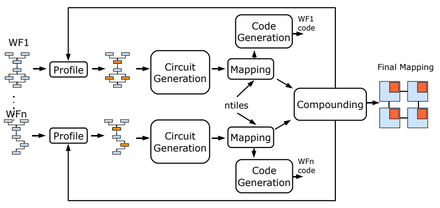


Figure 4: *CMA work flow.*

Our general work flow is depicted in Figure 4. Starting from the description of each target waveform represented by a task flow graph as shown in Figure 2(a) and 2(c), we profile each of the waveform to identify tasks that need to be accelerated. We then generate the corresponding custom circuits for those tasks and repeat the profiling (using simulation). At the end of the iterative process, we (1) generate the appropriate compound circuits for each tile, and (2) generate the target code for each waveform including calls to accelerators and inter-tile communications.

3.1 Architecture of the CMA

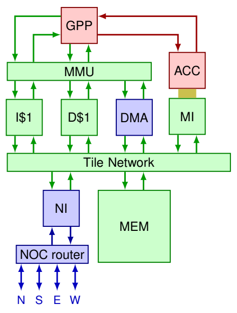


Figure 5: *CMA tile.*

The CMA is a multi-tile, distributed memory architecture on a mesh toplogy NoC, see Figure 1. The choice of the memory architecture and the NoC architecture is orthogonal to the scope of this paper.

Figure 5 shows the details of a single tile of the mesh. As previously explained in Section 1, each tile is essentially composed of a general purpose processor (GPP) coupled with a custom accelerator (ACC) and a local memory (MEM). In this configuration, each processing unit (GPP or ACC) only accesses its own local memory to perform computations and explicit data transfers between tiles are performed by the DMA controller. The DMA is programmed by the GPP through memory-mapped registers and the Memory Management Unit

(MMU) is responsible for mapping memory accesses either to the cache units or the DMA registers.

The Memory Interface (MI) [10] is responsible for supplying the custom accelerator with the necessary data. The MI is specially designed for multi-stream accelerators. It combines the determinism of DMAs, while retaining several of the performance advantages of general-purpose processors memory systems. The MI can accommodate multiple concurrent requests, out of order requests, and it can take advantage of reuse across concurrent streams, significantly improving the observed bandwidth. It is composed of streams capable of fetching complex memory patterns (non-monotonic references, sparse accesses, etc), and a small Stream Table to manage concurrent accesses and to enable short-distance temporal reuse [10].

Finally, each tile is interfaced to the NOC router through a Network interface (NI) and all communications between tiles are done through the DMA. We used wormhole routing for communications over the mesh topology.

We used a 650Mhz PowerPC405 architecture for the GPP with a 4-way associative 32KB instruction cache and a 4-way associative 32KB data cache. We augmented the PowerPC ISA with instructions to call and configure the accelerator [33]. In our implementation, we used a regular SDRAM memory for local memories.

3.2 Programming the CMA

```
tile #0
int tile0_main(void)
  uint8_t in0[SIZE_IN];                              //scrambler input
  uint8_t out0[SIZE_scrambler_out];         //first scrambler output
  uint8_t out1[SIZE_scrambler_out];        //second scrambler output
  int32_t fir_out[SIZE_fir];                //fir accelerator output

  for( ;; )
    // ... in0 = output from spreader1
    scrambler(0,in0,out0,out1);
    // send first scrambler output to tile #2
    dma_put(TILE2,DMA_ID_scrambler_out,out0,SIZE_scrambler_out);

    // start the fir accelerator on 2nd scrambler output
    accel_init(ID_txfir);                        //select fir circuit
    accel_mta(out1,in_stream0);               //init of input stream
    accel_mta(fir_out,out_stream0);          //init of output stream
    accel_start();                                  //start circuit
    // send fir accelerator output to tile #1
    dma_put(TILE1,DMA_ID_fir_out,fir_out,SIZE_fir);
    // ...
```

```
tile #1
int tile1_main( void )
  int32_t fir_out[2][SIZE_fir];                  // fir double buffer
  int fir_buffer_idx=0;            // current fir double buffer index

  // configure dma and double buffer to process tile#0 output
  dma_config(TILE0,DMA_ID_fir_out,SIZE_fir,fir_out[0],fir_out[1]);

  for( ;; )
    // get fir accelerator output from tile #0
    dma_get(TILE0,DMA_ID_fir_out,fir_out[fir_buffer_idx],SIZE_fir);
    adder1(fir_out[fir_buffer_idx],adder_out);
    // ...
    fir_buffer_idx = 1 - fir_buffer_idx;
```

Figure 6: *Programming example of CMA.*

The CMA concept is orthogonal to the programming model. In this study, we used a distributed-memory model; the task mapping and data transfers are detailed below.

The application task flow is distributed among the tiles as tile binaries. Figure 6 shows a partial view of the WCDMA TX waveform code. The code implements the three tasks `scrambler, FIR3` and `adder1`. In `Tile 0`, the `scrambler` task is a software task with 2 outputs, `out0` and `out1`. The output of task `out1` is passed to the FIR accelerator of the same tile. This accelerator sends its output `fir_out` to `Tile 1` for the task `adder1`.

To invoke the FIR accelerator, we call the `accel_init()` function to configure the compound circuit so that it executes the desired task. Function `accel_mta` (move to accelerator) initializes the different registers and streams of the accelerator (in the example, the input and output streams of the accelerator are initialized with the addresses of the input and output buffers of the task).

To transfer data from one tile to another, a pair of `dma_put`/`dma_get` functions is used. On the sender side, we specify in the `dma_put` function the destination tile, data source, size as well as a unique ID (`DMA_ID_fir_out`) that defines the communication channel. This ID is useful when more than one `dma_put` call from the same tile target another tile.

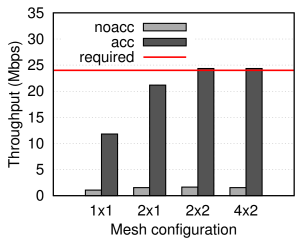
On the receiver side, we use a double buffering scheme to allow parallel reception and processing of data. In the example, we use two identical buffers (initialized in `dma_config`, `fir_out[0]` and `fir_out[1]`), and alternate between receiving data in one while processing data of the other (See Figure 6).

4 Performance Evaluation

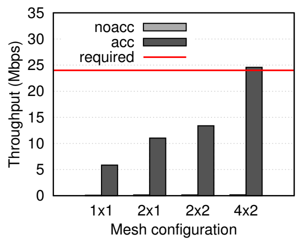
Simulation infrastructure. In order to evaluate the CMA architecture, we built a cycle accurate, distributed memory multi-core simulator using the UNISIM [5] infrastructure environment. For the GPP architecture of the tiles, we consider a regular 90nm PowerPC405 [15] architecture running at 650 MHz. We assume 3-cycle (˜4.6ns) latency circuits for simulation, this assumption is somehow pessimistic because circuits can be further pipelined [24]. The processor interface, the tile details and the NoC infrastructure are described in Section 3.1.

Synthesis infrastructure. We developed a tool chain which automatically creates compound circuits and generates Verilog HDL based on our intermediate circuit representation. We then synthesized all circuits using Synopsys Design Compiler [2] and TSMC 90nm standard library, with the highest mapping effort of the design compiler (`-map_effort high -area_effort high` options). Dynamic power consumption is calculated assuming 50% switching activity.

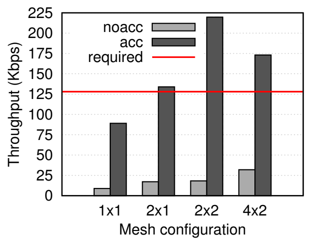
As mentioned in Section 2.1 the main goal of an SDR



(a) 802.11a TX



(b) 802.11a RX



(c) WCDMA TX

Figure 7: *CMA Performance: (a) 802.11a TX, (b) 802.11a RX, (c) WCDMA TX*

implementation is to achieve the required real-time throughput at the physical layer. We evaluate 802.11a and WCDMA waveforms on three different configurations of a CMA in addition to a baseline configuration (1x1) which consists of one tile. The three configurations tested are a dual-tile configuration (2x1), a 4-tile configuration (2x2) and an 8-tile configuration (4x2). For each configuration, we simulate the architecture, with `(acc)` and without `(noacc)` accelerator.

Figure 7 shows the resulting throughput for the three waveforms. In all cases, running the different tasks of the waveforms without acceleration leads to throughputs far from the required real-time throughput (on a 2x2 CMA, 1.5 Mbps TX, 0.155Mbps RX in 802.11a for a required throughput of 24Mbps, and 31Kbps in WCDMA TX for a minimum of 128Kbps required); and adding more cores to map more tasks contributes little to the performance, as shown in Figure 7. This is mainly because the SDR consists of pipelined tasks and the throughput is bounded by the task of lowest throughput. For example, the throughput of the software FIR task in 802.11a TX is 1.57 Mbps and the throughput of a sequential software implementation of the viterbi decoder is 0.27Mbps. These tasks bound the overall pipeline throughput.

More importantly, Figure 7 shows that both parallelism and acceleration must be combined to achieve performance scalability. On the 1x1 configuration (no parallelism), speedups due to acceleration vary from 10x (WCDMA TX) up to 68x on the 802.11a RX waveform. Further scaling is then achieved with task parallelism. Waveform 802.11a RX, for example, scales well: using `(acc,1x1)` configuration as a baseline, the speedups are respectively 1.8, 2.2 and 4.1 on the 2x1, 2x2 and 4x2 configurations, achieving the required real-time throughput on the 4x2 configuration. On the WCDMA TX waveform, we achieve the required throughput on a smaller configuration (2x2), however the benefit of parallelism decreases on the 4x2 configuration due to communications overhead.

5 Related Work

The CMA differs from MPSoC architectures [32] especially on the programming approach: the regular template allows to use any of the existing programming models designed for homogeneous multi-cores, while programming for MPSoC is notoriously difficult [30]. In the context of SDR, it also allows to use existing waveform development environments such as [25].

Several approaches [20, 31, 11] previously proposed specialized architectures for SDR. Lin et al. [20] proposed a parallel architecture, SODA, consisting of very wide programmable SIMD processors. They later proposed in [31] to add a dedicated hardware turbo decoding unit to increase the computational and energy efficiency of the architecture. Our approach generalizes the customization process by generating the necessary custom units "on-demand" for performance- and energy-intensive tasks. Compounding provides the necessary flexibility to implement several wireless protocols.

Finally, loop based accelerators [3, 7, 8] are alternatives to the compound accelerator as proposed in this study.

6 Conclusions and Future Work

We have proposed the CMA, an architecture which provides both the regular template, and thus ease of programming, of homogeneous multi-cores, and the efficiency of custom acceleration. Combined with circuit (accelerator) compounding, which enables to merge multiple circuits into one with little overhead, a CMA can either tackle complex tasks with few tiles and thus at a cheap cost, or it can become a flexible architecture capable of tackling a broad range of applications.

For now, tasks are mapped statically to tiles. By combining a run-time system with accelerators replicated across multiple tiles (at almost no cost thanks to compounding), we will be able to implement dynamic task mapping and thereby minimize the number of required tiles.

References

[1] Joint Tactical Radio System. http://jpeojtrs.mil.

[2] Synopsys design compiler. http://www.synopsys.com.

[3] Tensilica. http://www.tensilica.com/.

[4] Designing high-performance DSP hardware using Catapult C synthesis and the altera accelerated libraries. Mentor Graphics Technical Library, October 2007.

[5] D. August, J. Chang, S. Girbal, D. Gracia-Perez, G. Mouchard, D. A. Penry, O. Temam, and N. Vachharajani. UNISIM: An open simulation environment and library for complex architecture design and collaborative development. *IEEE Comput. Archit. Lett.*, 6(2):45–48, 2007.

[6] K. Chakraborty, P. M. Wells, and G. S. Sohi. Over-provisioned multicore processor, September 2009. Patent application, 11/867508.

[7] N. Clark, A. Hormati, and S. Mahlke. Veal: Virtualized execution accelerator for loops. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 389–400, Washington, DC, USA, 2008. IEEE Computer Society.

[8] K. Fan, M. Kudlur, G. S. Dasika, and S. A. Mahlke. Bridging the computation gap between programmable processors and hardwired accelerators. In *15th International Conference on High-Performance Computer Architecture (HPCA-15 2009), 14-18 February 2009, Raleigh, North Carolina, USA*, pages 313–322. IEEE Computer Society, 2009.

[9] S. Fechtel and A. Blaickner. Efficient FFT and equalizer implementation for OFDM receivers. *IEEE Transactions on Consumer Electronics*, 45(4):1104–1107, 1999.

[10] S. Girbal, S. Yehia, H. Berry, and O. Temam. Stream and memory hierarchy design for multi-purpose accelerators. In *SAW-1: 1st Workshop on SoC Architecture, Accelerators and Workloads (SAW-1), in conjunction with HPCA-16*, 2010.

[11] J. Glossner, D. Iancu, M. Moudgill, G. Nacer, S. Jinturkar, S. Stanley, and M. Schulte. The sandbridge sb3011 platform. *EURASIP J. Embedded Syst.*, 2007(1):16–16, 2007.

[12] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers. Optimized generation of data-path from c codes for FPGAs. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 112–117, Washington, DC, USA, 2005. IEEE Computer Society.

[13] K. Hirata and J. Goodacre. ARM MPCore; the streamlined and scalable ARM11 processor core. In *ASP-DAC '07: Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 747–748, Washington, DC, USA, 2007. IEEE Computer Society.

[14] H. Holma, A. Toskala, et al. *WCDMA for UMTS: Radio access for third generation mobile communications*. Citeseer, 2000.

[15] IBM. PowerPC 405 CPU Core. Sept. 2006.

[16] IEEE. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (ANSI/IEEE Std 802.11, 1999 Edition (R2003))*. Institute of Electrical and Electronics Engineers, Inc., June 2003.

[17] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359, 1999.

[18] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE micro*, 25(2):21–29, 2005.

[19] H. Lee, Y. Lin, Y. Harel, M. Woh, S. Mahlke, T. Mudge, and K. Flautner. Software Defined Radio–A High Performance Embedded Challenge. *High Performance Embedded Architectures and Compilers*, pages 6–26, 2005.

[20] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. SODA: A low-power architecture for software radio. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 89–101, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Y.-K. Lin, D.-W. Li, C.-C. Lin, T.-Y. Kuo, S.-J. Wu, W.-C. Tai, W.-C. Chang, and T.-S. Chang. A 242mw, 10mm21080p h.264/avc high profile encoder chip. In *DAC '08: Proceedings of the 45th annual Design Automation Conference*, pages 78–83, New York, NY, USA, 2008. ACM.

[22] H. Meyr, M. Moeneclaey, and S. Fechtel. *Digital communication receivers: synchronization, channel estimation, and signal processing*. John Wiley & Sons, Inc. New York, NY, USA, 1997.

[23] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. *Lecture Notes in Computer Science*, 1067:493–498, 1996.

[24] L. Pozzi and P. Ienne. Exploiting pipelining to relax register-file port constraints of instruction-set extensions. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 2–10, New York, NY, USA, 2005. ACM.

[25] V. Ramakrishnan, E. M. Witte, T. Kempf, D. Kammler, G. Ascheid, H. Meyr, M. Adrat, and M. Antweiler. Efficient and portable sdr waveform development: The nucleus concept. *CoRR*, abs/0906.3313, 2009.

[26] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):1–15, 2008.

[27] O. Sentieys, J. Diguet, and J. Philippe. GAUT: a high level synthesis tool dedicated to real time signal processing application. *EURO-DAC, September*, 2000.

[28] W. Tuttlebee. Software-defined radio: facets of a developing technology. *Personal Communications, IEEE*, 6(2):38 –44, apr 1999.

[29] M. Valenti and J. Sun. The UMTS turbo code and an efficient decoder implementation suitable for software-defined radios. *International journal of wireless information networks*, 8(4):203–215, 2001.

[30] P. van der Wolf, E. de Kock, T. Henriksson, W. Kruijtzer, and G. Essink. Design and programming of embedded multiprocessors: an interface-centric approach. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 206–217, New York, NY, USA, 2004. ACM.

[31] M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder, and K. Flautner. From SODA to scotch: The evolution of a wireless baseband processor. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 152–163, Washington, DC, USA, 2008. IEEE Computer Society.

[32] W. Wolf, A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(10):1701 –1713, oct. 2008.

[33] S. Yehia, S. Girbal, H. Berry, and O. Temam. Reconciling specialization and flexibility through compound circuits. In *15th International Conference on High-Performance Computer Architecture (HPCA-15 2009), 14-18 February 2009, Raleigh, North Carolina, USA*, pages 277–288. IEEE Computer Society, 2009.