Revisiting Accelerator-Based CMPs: Challenges and Solutions

Nasibeh Teimouri
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
nteimouri@ece.neu.edu

Hamed Tabkhi
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
tabkhi@ece.neu.edu

Gunar Schirner
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
schirner@ece.neu.edu

ABSTRACT

Heterogeneous Chip MultiProcessors (CMP)s, which combine processor cores with specialized HW accelerators, are one main approach to high-performance low-power computing. While it is promising for few accelerators, the scalability is a major challenge with increasing number of accelerators. Resources including memory, communication fabric and processor turn into bottlenecks and result in accelerator under-utilization and cripple the performance.

This paper analyzes the scalability of heterogeneous CMPs with many accelerators and identifies bottlenecks and their impacts on system performance. It introduces an analytical method for scalability/bottleneck analysis that is backed up by a simulation-based performance analysis (using automatically generated virtual platforms). This paper proposes a novel architecture template: Transparent Self-Synchronizing (TSS) accelerators for efficient/scalable realization of streaming applications. TSS achieves the efficiency / scalability through configurable point-to-point connections and self synchronization between HW accelerators and efficient management of accelerator's memory.

This article demonstrates the TSS benefits using both analytical and simulation methods. TSS significantly reduces the pressure on the communication fabric, processor load, and memory requirements to improve scalability. Even with increasing number of accelerators, TSS can achieve more than 85% accelerator utilization. In contrast, in ACC-based CMPs the accelerator utilization drops fast; less than 40% with six accelerators or even worse with more accelerators. The scalability benefits of TSS are more pronounced as the number of hardware accelerators increases.

1. INTRODUCTION

The ever-increasing demand for high-performance low-power computing prompted the shift toward more specialization in Chip MultiProcessors (CMPs). Heterogeneous accelerator-based CMPs (ACC-based CMP) combine general-purpose (application) processors with specialized custom-HW accelerators (ACCs) [1, 2]. With underutilized silicon the push for specialization gets even stronger, as it allows proliferating many ACCs within a chip to streamline the embedded realization of high-performance applications [3].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
DAC '15, June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06...$15.00
http://dx.doi.org/10.1145/2744769.2744902.

Streaming applications, such as multimedia, vision computing, software defined radio, radar, and cryptography, particularly lend themselves to ACC-based realization. They all share real-time constraints, very intense computation and communication demands, and are often deployed in a power-limited environment. Streaming applications are mainly constructed out of smaller kernels performing compute-intense (but repetitive) operations over streaming data. The general aim is to map computation-intense kernels into HW ACCs and leave high-level processing and control to the processor cores (SW execution).

To illustrate the context, Fig. 1 outlines a typical ACC-based CMP, inspired by [4, 5]. The ACC-based CMP includes one or more processor cores loosely coupled with many ACCs. System-level components realize the streaming data communication between ACCs and the processor core(s): Scratch Pad Memory (SPMs) for each ACC, a shared memory across all ACCs, a multi-layer communication fabric and many Direct Memory Access (DMA) channels. The processor core(s) is responsible for synchronizing/scheduling transaction across the ACCs. The synchronization load makes the processor core(s) a shared resource from an ACC's perspective.

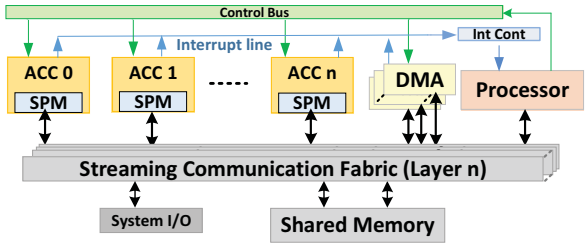


Figure 1: Heterogeneous ACC-based CMPs.

Through specialization, ACC-based CMPs (such as Fig. 1) dramatically increase performance / power efficiency over SW-only approaches. However, ACC-based CMPs do not scale well. As the number of ACCs increases, the shared resources across ACCs become bottlenecks, such as limited on-chip memory, limited concurrent communication channels, and available processor cycles. In result, ACCs wait more on shared resources and become underutilized. Relieving resource pressure is difficult; Increasing SPM size per ACC is limited by area (total on-chip memory) and more importantly by high power consumption [6, 7, 8]. Even a multi-layer communication fabric easily saturates with the traffic between ACCs [9]. Many CPU cycles are just spent for ACC synchronization and coordination. Flooding a core with ACC coordination (many interrupts) may it a bottleneck with rising number of ACCs.

A systematic analysis of the scalability of CMPs with many ACCs is needed to understand the underlying challenges. Novel system-level architecture templates are needed that overcome current scalab-

ility limitations and allow designers to progress on the trend of efficiency improvement through specialization.

This paper analyzes the scalability of heterogeneous CMPs with many ACCs, identifies bottlenecks and their impacts on system performance. It introduces an analytical method for scalability / bottleneck analysis that is backed up by a simulation-based performance analysis (using automatically generated virtual platforms). This paper proposes a novel architecture template: Transparent Self Synchronizing (TSS) accelerators for efficient realization of streaming applications. TSS improves efficiency through configurable point-to-point connections, self-synchronization between ACCs, and efficient management of memory.

This article demonstrates the TSS benefits using both analytical and simulation methods. TSS significantly reduces the pressure on the communication fabric, processor load, and memory requirements. Even with more than 16 ACCs, it can achieve more than 90% ACC utilization. The benefits of TSS are more pronounced as the number of ACCs increases.

In a nutshell, the contributions of this article are:

1. Analytical approach for exploring and quantifying scalability limitations of ACC-based CMPs that reveals the major system bottlenecks.
2. Novel architecture template, Transparent Self-Synchronizing (TSS) ACCs, to improve scalability allowing for an efficient realization of streaming applications using many ACCs.
3. Evaluation of TSS benefits through automatically generated virtual platforms based on vision applications.

This paper is organized as follows: Section 2 briefly overviews related work. Section 3 identifies and quantifies the scalability limitations of ACC-based CMPs. Section 4 introduces our more scalable TSS architecture template. Section 5 evaluates the benefits of the proposed TSS architecture. Section 6 concludes the paper.

2. RELATED WORK

Much effort has been invested toward architecting ACC-based CMPs in both academia and industry, mostly focusing on high-performance low-power realization of streaming applications. Platform 2012 [10], Streaming Accelerator[11] and SARC Architecture [12], and Texas Instruments (TI) DaVinci [13] IBM power EN [14] and Analog Devices (ADI) Blackfin 60x series [15] are only few academic and industrial examples. Compared to homogenous software-only solutions, the ACC-based CMPs achieve much higher performance and power efficiency. On the downside, they all demand a large on-chip memory, significant communication bandwidth and synchronization overhead per ACC making the scalability of ACC-based CMPs highly limited.

Comparatively little work already hints to the scalability issues of ACC-based CMPs [9, 6, 7, 8, 16, 17]. [9] shows that the achievable speedup of ACCs is bounded by the latency of the communication fabric. The speedup reduces exponentially as the communication latency increases. In a different study, [6] tries to tackle the problem of large SPM per ACC. [6] proposes accelerator-store architecture for run-time management of SPM memory per ACC reducing the overall on-chip demand. Similarly, the work in [7, 8] aims to optimize the SPMs by offering runtime memory allocation across many ACCs. In a different approach, [16] and [17] already appreciate the resource limitations when it comes to integrating many ACCs. Following that, they propose analytical approaches to find the optimum set of ACCs either with respect to communication limitation (in [16]) or area limitation (in [17]).

In an orthogonal view, some research aims to improve the architecture efficiency of ACC-based CMPs [4, 18]. Rather than tackling the system bottlenecks, the main focus has been on composability of many ACCs. [4] proposes Accelerator-Rich CMP offering a rich set of accelerators interconnected through a NoC fabric to the processor cores. In an improvement over [4], CHARM [18] decomposes ACCs into smaller blocks and introduces an independent Accelerators Block Composer (ABC) to reduce the synchronization interaction with the processor. However, CHARM still imposes a large volume of traffic and also requires large SPMs to realize inter-accelerator data exchange.

Overall, a holistic view on ACC-based CMPs design is missing. Previous approaches either zoomed into one aspect of design [6, 7, 8, 17, 16], or aimed for new architecture approaches without studying the effects of the bottlenecks and resource limitations [4, 18]. Conversely, this work provides a holistic view of the design dimensions and scalability challenges of ACC-based CMPs. This paper systematically identifies and quantifies the design bottlenecks. Following that, this paper also proposes a novel solution, Transparent Self-Synchronizing (TSS) accelerators which streamlines the realization of CMPs and removes bottlenecks hurting efficient realization of ACC-based CMPs.

3. SCALABILITY LIMITATIONS OF ACC-BASED CMPS

This section studies and explores the scalability of ACC-based CMPs (highlighted in Fig. 1). It focuses on streaming applications (e.g. vision computing; multimedia and radar processing) as they are well-suited for ACC-based realization.

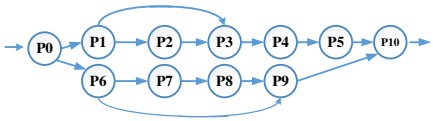


Figure 2: Streaming application captured in SDF

Streaming applications are often captured in process models, such as Synchronous Data Flow (SDF) [19] or Kahn Process Network (KPN). Fig. 2 illustrates an example of SDF model consisting of 11 computation nodes. Nodes represent individual computation kernels performing their predefined tasks upon the availability of data. Edges connect nodes to construct the complete vision flow. Edges indicate communication relations and computation dependencies across the computation nodes. Ideally, all nodes operate in a pipeline fashion over the streaming data in a consumer and producer fashion [20]. Such a model particularly maps well to an ACC-based CMP, where nodes with compute-intense kernel are mapped on individual ACCs and nodes with control-oriented tasks are mapped to a processor. Nodes pass data to each other using shared memory through DMA. The processor has the responsibility for orchestrating the entire system, including synchronization of ACCs and DMAs for coordinated data access and data processing.

With the increase in processing, the processor needs to spend more time for this orchestration. Due to on-chip memory limitations (area and power consumption), ACCs cannot operate on the whole data set (i.e. frame) at once. Instead, the input data is split and operated on in smaller chunks called *jobs*. The SPM size of each accelerator determines maximum job size. Smaller SPMs lead to a smaller job size, thus increase synchronization load on the processor.

To illustrate the work necessary for orchestrating ACC interaction, Fig. 3 plots a typical event sequence for processing a single job. It takes *9 phases* across processor and ACC. The processor *(1)* initializes a DMA to copy the job's input data from shared memory to the ACC's SPM. The DMA *(2)* transfers the data through the streaming communication fabric and indicates by interrupt *(3)* finishing the

transfer. Then, *(4)* the processor initializes the ACC to process the data *(5)*. When done, the ACC notifies *(6)* the processor, which then configures the DMA to write back the processed data from ACC's SPM to the shared memory *(7)*. The DMA writes the data to shared memory (*8*) and signals being done *(9)*.

To achieve the highest performance, all ACCs need to be fully utilized. To overlap processing and communication, ACCs can use double buffering for read and write channels. Double buffering can improve the overall throughput at the cost of larger SPMs (almost double). Ideally, ACCs are fully utilized as illustrated in Fig. 4 with a (perfect) pipeline execution for 3 ACCs. By overlapping data transfer and computation in each ACC, processing data by ACC (P) can be done in parallel with receiving data from DMA (R) and sending data to DMA (S).

However, keeping ACCs fully utilized demands immense communication bandwidth, very large memory, and many processor cycles as the number of ACC increases. Conversely, ACC-based CMPs (Fig. 1) offer a limited set of resources, which often creates a mismatch between application demands and the architecture's available resources. The gap increases as the number of ACCs increase. To analyze these bottlenecks, the following section introduces a first-order analytical model to capture the demand for shared resources and the effect on ACCs utilization.

3.1 Analytical Model and Evaluation

The ideal pipeline execution as illustrated in Fig. 4 is often not achieved due to resource limitations. E.g., with limited concurrency in the communication infrastructure, the ACC's bus requests are serialized and the pipe duration expands. In order to estimate the serialization effects, we have derived a first-order performance model. For the sake of simplicity, we assume an equal job size and processing duration for each ACC. A single processor core, operating at 1GHz ($Freq_{Proc}$) with a light-weight OS with 20000 cycles ISR latency ($Load_{Int}$), coordinates the system. ACCs compute 1 byte per cycle at 200MHz ($Freq_{ACC}$) and have double buffer SPMs. The communication fabric supports four or eight parallel channels of 32bit width. Each communication channel has a dedicated DMA. The total data to process is 1GB. SPM size and job size vary with number of accelerators. Given these definitions, the latency of a single stage $Latency_{Pipe}$ can be calculated by Equation (1).

$$Latency_{Pipe} = Max(P, S, R) + Latency_{Synch} + Latency_{Arb} \quad (1)$$

$$Latency_{Synch} = Num_{ACC} * 3 * Freq_{Proc} * Latency_{ISR} \quad (2)$$

$Latency_{Pipe}$ is a summation of the maximum between processing (P), sending (S), receiving (R) – as all happen in parallel –, the latency of bus arbitration ($Latency_{Arb}$), and the synchronization latency in the processor core ($Latency_{Synch}$). For simplicity, we assume a constant $Latency_{Arb}$ (note: we consider contention for S and R). $Latency_{Synch}$ represents synchronization overhead on the processor(s), estimated by Equation (2). It depends on the frequency of the processor; ISR latency and number of simultaneous interrupt requests. We assume a constant $Latency_{ISR}$ and 3 interrupts per each job (as illustrated in Fig. 3). Following the pipeline fashion, the execution time of this system is as Equation (3).

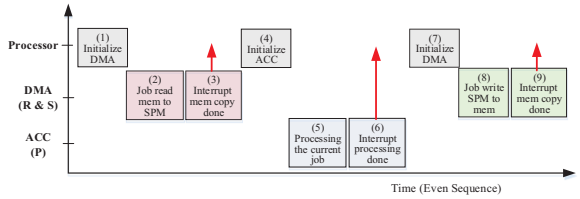


Figure 3: Event sequence of one transaction toward one ACC

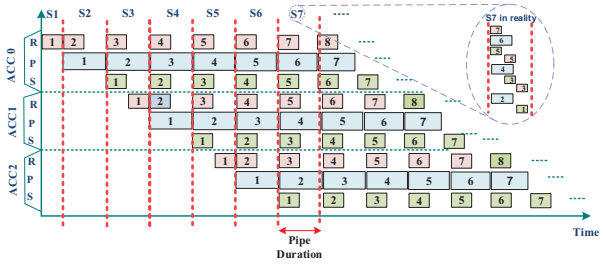


Figure 4: Timing diagram of accelerator integrated MPSoC

$$Time_{Exec} = (Num_{Jobs} + Num_{ACC} - 1) * Latency_{Pipe} \quad (3)$$

$$Latency_{R/S} = Job_{size} * (\frac{Bus_{layers}}{Bus_{freq}} + \frac{Mem_{ports}}{Mem_{freq}}) \quad (4)$$

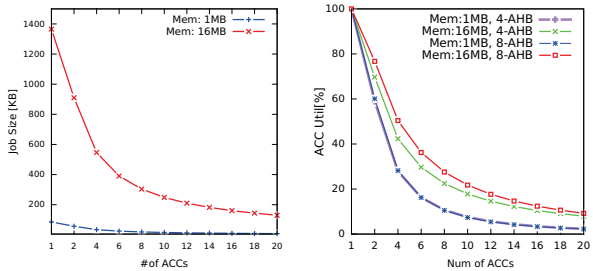
$$Latency_{P} = (\frac{Job_{size}}{ACC_{freq}}) \quad (5)$$

The latency of R and S depends on the number of memory ports (Mem_{ports}) and bus layers (Bus_{layers}) as calculated in Equation (4) (symmetric for both R and S.) In contrast to communication, computation latency is fairly constant and only depends on the Job_{size} for a fixed frequency of ACC (highlighted in Equation (5)).

We assume the same Job_{size} for all ACCs, only depending on the available SPM. The total on-chip memory is equally distributed across all ACCs. Since the maximum on-chip memory size is limited, and each ACC needs an own SPM, the SPM size shrinks with increasing number of ACCs. We evaluate the impact of job size and number of ACCs, in two configurations: (1) 1MB of total on-chip memory (observed in current state-of-the-art embedded platforms), and (2) 16MB of total on-chip memory(assumed for future platforms). Fig. 5a shows the correlation between maximum job size and number of ACCs. The job size exponentially drops as the number of ACCs increases.

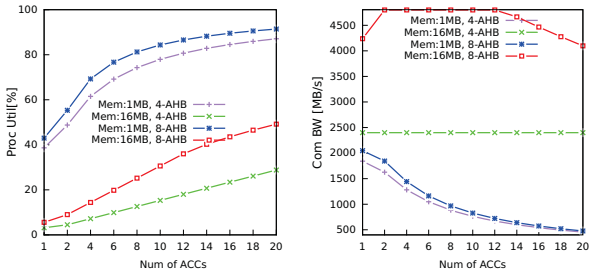
Fig. 5b plots ACC utilization over increasing number of ACCs for the two total memory sizes (1M and 16M) and two interconnect configurations: Multi-Layer AHB with 4 and 8 layers. ACC utilization drops for all configuration significantly as the number of ACCs increases beyond a few. The drop is more pronounced for the 1MB configuration and 4-AHB communication channels. ACCs do not receive their data in time for processing most of the time, and stay idle. Utilization is bounded by system resource limitations: (a) only 4 or 8 simultaneous transfers through the communication fabric, and (b) interrupt request serialization in the host processor. With larger SPMs in the 16MB setting, ACCs synchronize less often, thus the effect of (b) is less pronounced.

To analyze reasons for the low ACC utilization, Fig. 6a and Fig. 6b plot the core utilization and communication bandwidth. The core utilization significantly increases with more ACCs. The utilization is higher in the 1MB configuration, as it results in a smaller job size



(a) Job size (b) ACC utilization

Figure 5: Job size and ACC utilization with increasing # of ACCs.



(a) Processor utilization (b) Communication BW

Figure 6: Processor utilization and communication BW with increasing number of accelerators.

and more transactions. The utilization levels off with more than 10 ACCs as the processor core becomes the bottleneck. The ISR latency increases as the requests are being serialized. The *Mem:16MB, 4-AHB* is more linear and does not saturate the processor indicating that a different resource is the bottleneck.

Fig. 6b plots the communication bandwidth (BW). The configuration *Mem:16MB, 4-AHB* saturates the whole fabric having a flat BW of 2400MB/s with any number of ACCs. Even 8 layers saturate at 4800MB/s. However, with more than 12 ACCs, processor serialization becomes more dominant and bandwidth drops. In the 1MB configurations (both 4 and 8 layer), the communication fabric is less utilized as the processor core is drowned in synchronization overhead due to the smaller jobs.

Overall, our analytical explorations reveal the scalability challenges of ACC-based CMPs. As the number of ACCs increases, the benefits are overshadowed by the significant overhead required for orchestrating the entire system. Three main bottlenecks appear: (1) a significant synchronization load to the host processor for synchronizing / scheduling ACCs; (2) a large volume of redundant communication traffic for exchanging the streaming data across the ACCs; (3) a large on-chip memory dedicated for ACC's SPM and shared memory space to hold the streaming data under processing. Novel solutions are needed to eliminate the bottlenecks and streamline realization of heterogeneous CMPs with many ACCs.

4. TRANSPARENT SELF-SYNCHRONIZED (TSS) ACCELERATORS

This section introduces our architecture template Transparent Self-Synchronized (TSS) Accelerators to avoid the system bottlenecks when integrating many ACC on a chip. Fig. 7 highlights the basic components of our TSS architecture. TSS primarily is a composition of two gateways (Starter and Terminator) and set of ACCs interconected with multiplexers (MUXes). The gateways offer autonomous control and scheduling to minimize or even eliminate

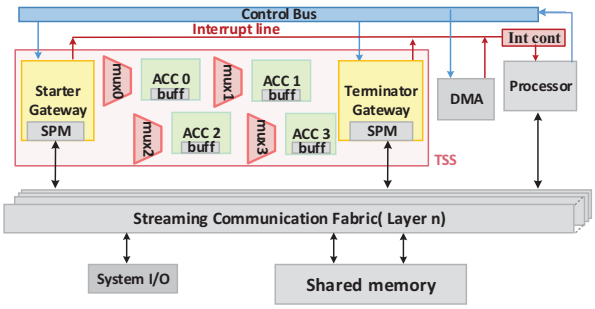


Figure 7: Transparent Self-Synchronized (TSS) Accelerators.

the unnecessary rescheduling interaction with the host processor. Also, MUX-based interconnect hides ACC-to-ACC streaming traffic from the system communication fabric. The combination of gateway and MUX-based interconnection helps to efficiently manage the SPM required per ACC. The following sections provide details.

4.1 Autonomous ACCs Gateways

The gateways offer autonomous control for synchronizing and scheduling of the ACCs. Self-synchronization in gateway allows autonomous management of the ACC data interaction, avoiding costly scheduling overhead on the host processor. The gateways can independently fetch new configurations data from the memory directly. The configuration data is placed in predefined memory addresses and fetched autonomously through the gateway controller. The transitions between configuration sets are triggered by events (from host processor or internal). The gateways are also responsible for governing spatial parallelism across the ACCs. Multiple concurrent application flows can be mapped to TSS at the same time as long as they do not need same ACC at the same time.

The gateways manage the ACCs, make their integration autonomous and eliminate the need for micro management of each data transaction by the processor. The starter gateway splits an incoming external job into more internal jobs based on SPM size and feeds connected ACCs. On the other side, terminator gateway collects the internal jobs from the last ACC for the coarser outside communication. Within the TSS, as shown in Fig. 8, the multiplexers can be configured to realize different parallel chains of accelerators, where each chain works in producer/consumer fashion.

4.2 MUX-Based Interconnection

The MUX-based interconnection offers a customizable direct ACC-to-ACC communication. Direct connection between accelerators removes the need to pass data through the bus and memory. With sufficient number of ACCs and multiplexers, parallel flows (we implemented 3) are possible. Based on the application's parallelism degree, more parallel applications could be configured to improve performance. For this purpose, gateways need to have enough size of SPM based on the granularity of data input and data output per application flow. Within each flow, the multiplexers are configured for direct ACC-to-ACC communication without accessing memory for each transaction. This also avoids the need for synchronization by the processor which in turn makes it efficient to operate on smaller internal jobs.

Fig. 8 highlights the MUX-based interconnection across the ACCs. At run-time, ACCs are interconnected by MUX configuration. After establishing the connection; no further control is required, ACCs directly communicate independently and synchronized by the gateways. ACCs in different stages can be interconnected together in scatter, gather, and self-feedback modes. Additional MUXs allow for- and back-warding data between ACCs increasing the flexibility in macro-pipeline construction. The distributed MUXed interconnect system empowers concurrent execution. A fully connected n:n communication is not required. Instead, a sparser connectivity is

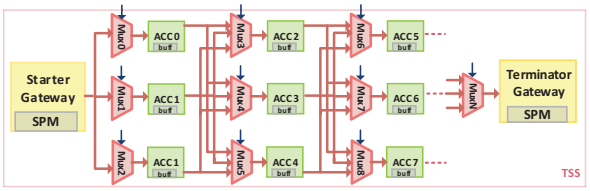


Figure 8: TSS MUX-based ACC-to-ACC connection.

Table 1: TSS with 2 different sizes of memory

	Mem: 1MB	Mem:16MB
Interrupt Rate [K/s]	20.15	1.67
Processor Utilization [%]	0.01	0.0012
ACC Utilization [%]	63.87	85.1
Communication Vol. [MB/s]	1673.2	2231.78

desirable that only offer feasible ACC connection with respect to the targeted applications.

TSS architecture also aids in reducing SPM size across ACCs. The direct communication eliminates the need for DMAs and their configuration. The self-synchronization eliminates the need for interrupts to the processor for each internal data transaction. Both together eliminate the negative effects (higher processor utilization, serialization) of smaller job sizes and thus allow to operate on a smaller internal job size. This dramatically reduces the total memory requirements.

4.3 TSS Analytical Evaluation

A similar timing diagram as shown in Fig. 4 can be imagined for TSS. However, P is smaller with the smaller internal job size. Moreover both R and S become negligible due to the direct ACC-to-ACC communication without needing copy data from/to memory. Both starter and terminator gateway operate on a lager external job size and then split/join into the smaller internal job size for ACC communication. Only two interrupts are exposed to the processor per each data input to the starter and data output from the terminator. This a low, constant volume of interrupts and a low, constant bandwidth. Table 1 shows the results using the same assumptions outlined in Section 3.1. As all internal traffic and synchronization is hidden, the rates/utilization remain constant regardless of the number of ACCs.

5. EVALUATION

This section evaluates the benefits of TSS in comparison to ACC-based CMPs using automatically generated virtual platforms. It utilizes different benchmarks in order to verify the correctness of proposed analytic models.

To facilitate the exploration of TSS architecture and ACC-based CMPs, we have captured a high-level description of each application model in a System-Level Design Language (SLDL)[21], and used an system-level refinement tool-chain[22] to automatically generate the virtual platforms (VPs) for both architectures. Each VP uses an ARM9 processor core (simulated by an OVP ISS with a cycle approximate model) running uCOS/II operating system. The communication fabric is a multi-layer AMBA AHB (32 bit-width, 100MHz) with four concurrent communication channels. The multi-layer AHB is captured at the cycle-level to correctly represent arbitration and transfers. ACCs in both platforms operate at 200MHz. Each platform contains behavioral DMA models (one dedicated DMA per channel). The TSS architecture, in addition, contains the gateways (timing annotated behavior level) and the MUX infrastructure.

5.1 VP-based Scalability Analysis

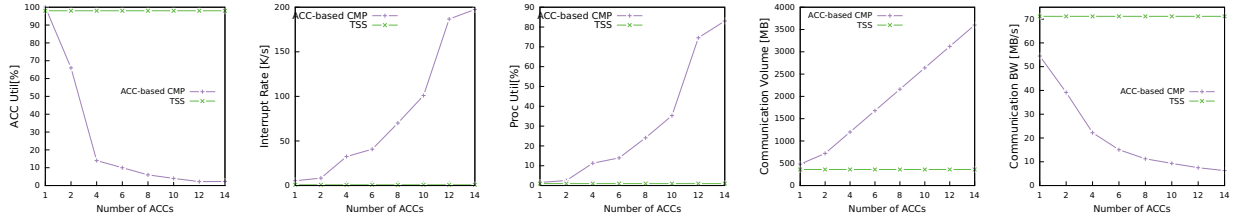
Section 3.1 identified the system bottlenecks of ACC-based CMPs through an analytical evaluation, and Section 4 quantified how the proposed TSS architecture avoids the bottlenecks. This section supplements the analytical study with an actual simulation of virtual platforms for both TSS and ACC-based architectures. The generated virtual platforms are configured based on the assumptions in Section 3.1. Due to the long simulation time at cycle-approximate granularity, we explore both platforms only for 1MB total on-chip memory configuration and reduce the streaming data set to 60MB.

Overall, the simulation results validate our analytical results and demonstrate the benefits of TSS compared to ACC-based CMPs. Fig. 9a indicates a very high ACC utilization in TSS (above 96%) independent of the number of ACCs. In contrast, ACC utilization significantly drops in ACC-based CMP as number of ACCs increases. The major limitation in ACC-based CMPs is the limited on-chip memory, which then results in smaller job size, a very high interrupt rate leading to a high processor utilization. Fig. 9b illustrates this trend. As the number of ACCs increases, the interrupt rate significantly grows in ACC-based CMPs. This is a compound effect: (a) more ACCs need to be orchestrated, and (b) the job size decreases in the configuration with constant total memory. This results in a very high processor utilization and imposes significant serialization effect to the platform (highlighted in Fig. 9c). Conversely, TSS has fairly stable behavior. Very low interrupt rate results in very low processor utilization (less than 2%).

Fig. 9d compares the communication volume. Compared to TSS, ACC-based CMPs would demand much higher communication volume when all ACCs are 100% utilized. ACC-based CMPs would generate 7x more traffic for 14 ACCs than TSS. However, Fig. 9e shows a different picture. The observed bandwidth in ACC-based CMP is much lower than in TSS. Due to serialization effect of the processor to schedule the ACCs, ACCs are underutilized and produce a much lower bandwidth. Consequently, ACC-based CMP also shows a much higher execution time than TSS.

5.2 Heterogeneous HW SW Mapping

The evaluations up to now assumed that all kernels are executing on ACCs. This section evaluates the efficiency of TSS when some nodes execute in SW and others in HW to simulate a combined HW/SW mapping. This section also shows the most general case with a variable job size across nodes (and consequently ACCs) to better reflect the real applications. Fig. 10 illustrates an SDF model with heterogeneous mapping of nodes between ACCs (HW execution) and processor (SW execution). Kernels with blue color (P4, P6 and P10) are mapped to the processor and the others in red to the ACCs. Each kernel is customized to generate 1 byte data per cycle, and 60MB uniform traffic is processed in total. Both architectures map the same 3 kernels (*P4*, *P6*, *P10*) to SW. Processing the kernels in SW has lower priority and interrupts for synchronization are preferred. Both platforms operate on 1MB total on-chip memory for all SPMs.



(a) ACC Utilization[%] (b) Interrupt Rate (c) Processor Utilization (d) Communication Volume (e) Communication BW

Figure 9: Simulation comparison between TSS and ACC-based CMPs over increasing number of ACC with 1MB on-chip memory.

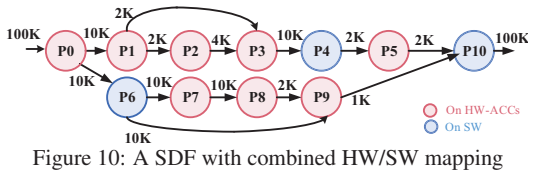


Figure 10: A SDF with combined HW/SW mapping

Comparing TSS versus ACC-based CMP in Table 2 shows that overall, TSS is about 6x faster than ACC-based CMP. TSS also reduces the synchronization demand by 5x reduction in the number of interrupts compared to ACC-based CMP. However, the rate of interrupt behaves in reverse: TSS has 1.2 times more interrupts per second than the ACC-based CMP. This is due to the vastly differing execution times, the 5 times higher number of interrupts spread in the ACC-based CMP over a 6 times longer time span. With the TSS internal communication, the fabric is less loaded with the TSS. It injects 3x less traffic than the ACC-based CMP.

Table 2: Heterogeneous HW/SW mapping

	TSS	ACC-based CMP
Execution Time [s]	15.91	100.03
Interrupt Volume	192	1063
Interrupt Rate [1/s]	12.06	10.62
Communication Vol [MB]	960	2880

5.3 Object Tracking Vision Flow

To demonstrate TSS efficiency on a real application, we have chosen an application from the embedded vision domain: object detection. Fig. 11 highlights the SDF model of the object detection application. It contains 6 streaming nodes: RGB2Gray, pixel smoothing, MoG background subtraction, dilation, erosion, and component labeling. The granularity of data passed across the nodes also differs depending on the kernel running per node.

Figure 11: SDF for object detection vision flow.

We have mapped and simulated the object detection application on both TSS and ACC-based CMP. To speed up simulation on the virtual platforms, we used a low resolution stream as an input with 120*160 pixels in RGB format and 60 Frames/s. Table 3 illustrates the result and compares the two platforms. Overall, TSS achieves much higher efficiency by faster execution time, much lower communication demand and synchronization load to the host processor. Based on Table 3, ACC-based CMP is about 7x slower than TSS with 6x more volume of interrupts. Similar to before, the rate of interrupts in ACC-based is a bit lower because of the longer execution time in ACC-based CMP. The sources of this long execution time is high coordination requests to the processor as well as serialization effect. Table 3 also indicates that the processor utilization in the ACC-based CMP is about 3x higher than in TSS. The serialization effect and high processor utilization both cause under-utilized ACCs which in fact results in lower bandwidth.

6. CONCLUSIONS

This paper has discussed the inherent scalability issues in ACC-based CMPs with increasing number of accelerators. Many resource bottlenecks appear on processor, memory and communication fabric. The paper introduced an analytical model to evaluate the impact on the key resources individually and extract trade-offs helpful for designing ACC-based CMPs. We proposed Transparent Self-Synchronizing (TSS) accelerators to relieve the scalability issues.

Table 3: Object tracking vision flow

	TSS	ACC-based CMP
Execution Time [s]	3.32	24.99
Interrupt Volume [K]	96805	585560
Interrupt Rate [K/s]	29093	25490
Processor Utilization [%]	35	86
Communication Vol. [MB]	322	765

Overall, TSS architecture opens a new path toward efficient realization of high-performance streaming applications by removing the system level bottlenecks of ACC-based CMPs. We demonstrated the benefits of TSS using the analytical model, as well as a set of automatically generated virtual platforms executing synthetic and real (object detection) streaming applications. Our results demonstrate that TSS improves scalability with keeping accelerator utilization above 85% with fewer limitations on resources.

7. REFERENCES

[1] J. Cong, , and et al., "Customizable domain-specific computing," *IEEE Design Test of Computers*, vol. 28, no. 2, pp. 6–15, March 2011.

[2] J. Cong and et al., "Accelerator-rich architectures: Opportunities and progresses," in *Design Automation Conference (DAC)*, 2014, pp. 180:1–180:6.

[3] Sematech. (2012) Itrs technology reaod map for semiconductor. [Online]. Available: http://www.itrs.net/Links/2011ITRS/Home2011.htm

[4] J. Cong and et al., "Axr-cmp: Architecture support in accelerator-rich cmps."

[5] Sotiriou-Xanthopoulos and et al., "Co-design of many-accelerator heterogeneous systems exploiting virtual platforms," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, July 2014, pp. 1–8.

[6] M. Lyons and et al., "The accelerator store framework for high-performance, low-power accelerator-based systems," *Computer Architecture Letters*, vol. 9, no. 2, pp. 53–56, Feb 2010.

[7] E. Cota and et al., "Accelerator memory reuse in the dark silicon era," *Computer Architecture Letters*, vol. 13, no. 1, pp. 9–12, Jan 2014.

[8] J. Cong and et al., "Bin: A buffer-in-nuca scheme for accelerator-rich cmps," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2012, pp. 225–230.

[9] P. Stillwell et al., "Hippai: High performance portable accelerator interface for socs," in *International Conference on High Performance Computing (HiPC)*, Dec 2009, pp. 109–118.

[10] D. Melpignano et al., "Platform 2012, a many-core computing accelerator for embedded socs: performance evaluation of visual analytics applications," in *Design Automation Conference (DAC)*, 2012, pp. 1137–1142.

[11] H. Javaid, D. Witono, and S. Parameswaran, "Multi-mode pipelined mpsocs for streaming applications," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2013, pp. 231–236.

[12] A. Ramirez et al., "The sarc architecture," *IEEE Micro*, vol. 30, no. 5, pp. 16–29, Sept 2010.

[13] G. Agarwal, "Get smart with TI's embedded analytics technology," *Texas Instruments (TI) white paper*, 2012. Available: www.ti.com/dsp-c6-analytics-b-mc.

[14] A. Krishna et al., "Hardware acceleration in the ibm poweren processor: architecture and performance," in *Parallel architectures and compilation techniques (PACT)*, 2012, pp. 389–400.

[15] A. D. Inc.(ADI), "ADSP-BF60x Blackfin Processor Hardware Reference Manual," *Reference Guide, Part Number 82-100113-01*, 2012.

[16] S. Nilakantan, S. Battle, and M. Hempstead, "Metrics for early-stage modeling of many-accelerator architectures," *Computer Architecture Letters*, vol. 12, no. 1, pp. 25–28, January 2013.

[17] A. Morad and et al., "Generalized multiamdahl: Optimization of heterogeneous multi-accelerator soc," *Computer Architecture Letters*, vol. 13, no. 1, pp. 37–40, Jan 2014.

[18] J. Cong and et al., "Architecture support for accelerator-rich cmps," in *DAC*, June 2012, pp. 843–849.

[19] M. Damavandpeyma and et al., "Modeling static-order schedules in synchronous dataflow graphs," in *Design, Automation Test in Europe (DATE)*, March 2012, pp. 775–780.

[20] W. Thies, V. Chandrasekhar, and S. Amarasinghe, "A practical approach to exploiting coarse-grained pipeline parallelism in c programs," in *International Symposium on Microarchitecture (IEEE Micro)*, 2007, pp. 356–369.

[21] "Specc technology open consortium." [Online]. Available: http://www.specc.gr.jp/eng/index.htm

[22] O. Rainer and et al., "System-on-chip environment: A specc-based framework for heterogeneous mpsoc design," *EURASIP J. Embedded Syst.*, vol. 2008, pp. 5:1–5:13, Jan. 2008.