This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2014.2299731, IEEE Embedded Systems Letters

High Communication Throughput and Low Scan Cycle Time with Multi-/Many-Core Programmable Logic Controllers

Arquimedes Canedo, Hartmut Ludwig, and Mohammad Abdullah Al Faruque, *Member, IEEE*

***Abstract*—Programmable logic controllers (PLCs) are hard real-time embedded systems designed to interact with (through sensors and actuators) and control physical processes in pharmaceutical, manufacturing, energy, and automotive industries. PLCs are the fundamental building blocks in modern industrial automation systems; they are designed to operate in extreme, harsh environments for decades without interruption. This paper presents and evaluates a novel, scalable hardware/software architecture for multi-/many-core PLCs capable of pipelining the "*sensing-executing-actuating*" stages of a control system and thus reducing the scan cycle time (SCT) and maintaining a high-throughput communication (HTC). SCT and HTC are two of the most important key performance indicators (KPIs) in industrial control systems because they determine the accuracy of the control algorithms.**

***Index Terms*—Cyber-Physical Systems, Embedded Control Systems, Programming Languages for Embedded Real-Time, Real-time Systems, Compilation Methods**

I. Introduction

Programmable Logic Controllers (PLCs) are hard real-time, cyclic, embedded control systems designed to interact with and control physical processes through sensors and actuators in pharmaceutical, manufacturing, energy, transportation, and other industries (See Figure 1(left)). PLCs typically operate with *millisecond* resolutions at the control level of the automation system. PLCs use a real-time operating system (RTOS) and industrial-strength communication networks – industrial Ethernet [10], Profinet [8], and CAN [4] – to interact with the physical system under control through multiple data streams. Switching from single-core PLCs to multi-/many-core PLCs represents a major investment for PLC manufacturers; however, some of the key benefits that motivate our R&D efforts are (See Figure 1(right)):

- Multiple devices (e.g. ASICs, FPGAs, DSPs in sensors/actuators at the field-level) can be consolidated in a single device (multi-/many-core PLC) making it more economical, power efficient, and reliable.
- The functionality of field level devices can be front-loaded into the control level devices (multi-/many-core PLCs) thanks to the availability of computational resources and high-speed on-chip interconnection networks in multi-/many-core processors.
- The capabilities of control algorithms are enhanced by having sensor and actuator functionality being executed in software at the control level, rather than in hardware at the field level. Sophisticated data processing algorithms can now be applied to sensor/actuator data to add more *intelligence* to the PLCs.

This paper focuses on the challenge of finding suitable software approaches to parallelize PLC applications to take advantage of multi-/many-core processors. PLCs are programmed with domain-specific languages – ladder diagram (LD), instruction list (IL), structured text (ST), function block diagrams (FBD), sequential function chart (SFC) – and standardized by the IEC 61131-3. With the objective of reducing scan cycle time (SCT) performance, previous work has mainly focused on application-level parallelization where different applications are mapped to different cores [1], and in functional parallelization where a compiler identifies data-independent fragments of the application that can be executed at the same time in different cores [2], [5]. Unfortunately, the typical workload distribution in industrial applications (See Figure 1(center)) results in an unbalanced parallel execution that yields modest SCT improvements.

The key contribution of this paper is the adaptation of software pipelining to PLCs. Specifically, our pipelining technique for PLCs inserts data dependency *delays*[1] and optimizes the partitioning of any IEC 61131-3 application into pipeline stages that can be overlapped in time to reduce the SCT of industrial applications. Our technique also prioritizes the communication with sensors and actuators using a time slack pipeline balancing approach to achieve a high-throughput communication (HTC) with sensors, actuators, and other automation devices. The paper is organized as follows. Section II presents our pipelining technique for SCT and HTC. Section III evaluates our technique with an industrial benchmark application. Section IV differentiates our contributions from the existing work on pipelining for PLCs. Section V summarizes our contributions.

II. Time-balanced Pipelining Algorithm for Scan Cycle Time and High-Throughput Communication

IEC 61131-3 applications are logically divided into three sequential, atomic steps as shown in Figure 2(a): "*sensing*" from sensors/network in "R" circles, "*processing*" the application, and "*actuating*" to sensors/network in "W" circles. Based on the observation that the *state* (e.g. speed, temperature, pressure) of a CPS is often the physical system itself read directly from the sensors, some industrial applications do not need to maintain the state within the application (e.g. "Mem") and the data dependencies can be relaxed to solve the shared memory problem. Consider the strongly connected component (SCC) created by {X→A→Y→Mem} in Figure 2(a). The X reads the memory location Mem at the beginning of the iteration, and the computation executes stages A, B, and C. Notice that the same memory location Mem is written by Y during the execution of stage B. Figure 2(b) illustrates the problem with pipelining. Since the iterations overlap in time, X0.3 has a dependency on Mem0 but this is not available until Y0 completes. Therefore, X0.3 reads Mem committed by the previous iteration or Mem_{init}. Since intermediate iterations (e.g. X0.3 and X0.6) are created to *improve the sampling time of the controller to the environment* and not to improve the execution time of a single iteration, the application memory behaves as a zero-order hold (ZOH). This mechanism maintains consistent memory values for the intermediate iterations that are aligned to the original SCT boundaries (e.g. X0, X1, X2, ...). Similarly, writes from intermediate iterations (e.g. Y0.3 and Y0.6) are not stored to the main memory but into privatized copies of the application memory space.

The focus of our partitioning algorithm is to find the optimal number of *processing* stages before providing it to the pipelining algorithm (see top part of Figure 3 for the various steps of this algorithm). In this work, we use an optimization algorithm to find the optimal number of processing stages based on a linear programming

A. Canedo and H. Ludwig are with Siemens Corporation, Corporate Technology, Princeton, USA e-mail: arquimedes.canedo@siemens.com, hartmut.ludwig@siemens.com

M. Al Faruque is with the Department of Electrical Engineering and Compute Science, University of California, Irvine, USA email: mohammad.alfaruque@uci.edu

[1]Delays relax the data dependencies in an industrial control system based on the observation that the state is maintained by the physical system and not by the application. Faster SCT provides more up-to-date snapshots of the physical system.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2014.2299731, IEEE Embedded Systems Letters

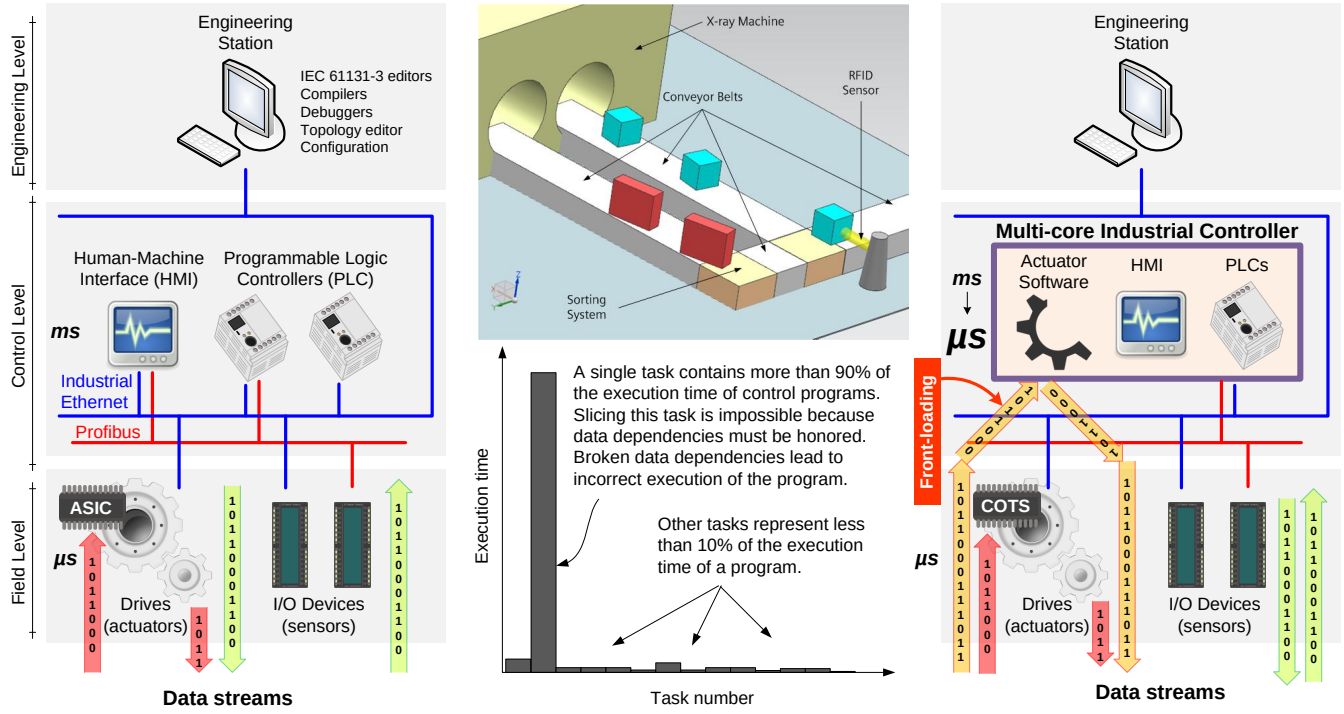
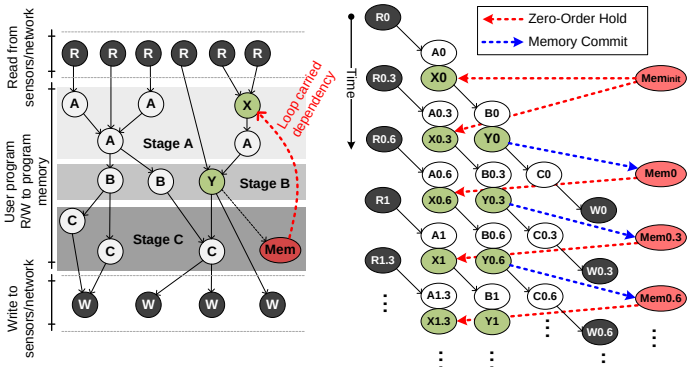


Fig. 1. Typical workload distribution of industrial applications (center-top). A single task concentrates more than 90% of the workload of the application (center-down). The current industry automation practice (left) uses single-core controllers and expensive ASICs and FPGAs to achieve high-performance. Multi-/Many-Core PLCs reduce the cost and increase the efficiency of automation systems because data processing is front-loaded to software and additional cores rather than specialized hardware (right).

method and WCET analysis of each IEC 61131-3 task of the application. The WCET timing is obtained by first compiling the IEC 61131-3 application into an intermediate representation based on C language. The application consists of a set of tasks $T = \{T_1, T_2, T_3, ..., T_N\}$ with execution times $T_e = \{T_{e1}, T_{e2}, T_{e3}, ..., T_{eN}\}$ and scan cycle times $T_s = \{T_{s1}, T_{s2}, T_{s3}, ..., T_{sN}\}$ executed in a multi-/many-core processor with communication overhead $Comm$. During the linear programming formulation, we have considered $\forall_i T_{ei} < T_{si}$, and communication overhead $Comm = Comm_1 + Comm_2$, where $Comm_1$ is the communication cost of data transfer and $Comm_2$ is the communication cost of synchronization between different cores. Communication cost is architecture-dependent and this information is characterized in the architecture information step in Figure 3. The real-time constraints on the execution time in relation to the SCT is guaranteed by the PLC runtime system[2]. Therefore, each $T_i \in T$ can be partitioned into *stages* $T_i = \{t_1, t_2, t_3, ..., t_M\}$ given that $\sum_{i=1}^{M} T_{ei} + Comm <= T_{si}$ and $ST_i >= \chi\%$, where $\chi = ((T_{si} - T_{ei}) * 100)/T_{si})$. ST_i is the *sleep time* given by the PLC to each task to allow thread synchronization, context switches, and execution of system-services (e.g. communications). In our evaluation, we have used $ST_i >= 10\%$ (see Figure 6).

Figure 3 shows a PLC application whose $T_{s1} = 100$ms. `X0` represents the output vector of the first iteration and `X1` the output of the second iteration. The implicit assumption made by the control engineers is that for every sample the system takes *at most* 100ms (deadline enforced by the PLC). Using our pipelining approach, the IEC 61131-3 compiler partitions the original application T_1 into three stages $t_1 = A$, $t_2 = B$, and $t_3 = C$. The stars in Figure 3 represent the data dependency delays introduced to create pipeline stages. During execution, the PLC runtime system schedules the stages into different cores (A to core 0, B to core 1, and C to core 2). Notice that in the multi-core execution, `X0` is produced at 100ms and `X1` at 200ms. These results correspond to the ones produced in the uniprocessor version. However, notice that once the pipeline is filled (after the 100ms mark), intermediate results `X0.3` ($t = 133.33$ms) and `X0.6` ($t = 166.66$ms) are produced. These intermediate iterations improve the quality of the control algorithms because they provide a more recent snapshot of the physical system under control. This occurs because the pipelining approach allows reading the sensors again before the result of the current iteration is produced, i.e. A0.3 and A0.6 are executed before `X0` is produced. In general, the performance improvement of pipelining is proportional to the workload of the longest stage, or the critical path. If an application is partitioned into balanced stages, the SCT improvement is proportional to the number of stages.

[2]PLCs, including single-core implementations, raise hardware exceptions whenever the execution time of a task exceeds the SCT.



a). Industry automation program with a loop-carried dependency illustrated by the SCC {X→A→Y→Mem}.

b). Reads from memory behave as a zero-order hold where the last known value is provided. Writes to memory are versioned.

Fig. 2. Pipelining the *sensing-processing-actuating* steps in a multi-/many-core PLC improves the SCT. Shared memory problem is relaxed by using zero-order-holds.

III. Evaluation

The implementation of our hardware/software architecture is based on an in-house proprietary software PLC[3]. The evaluation is based

[3]A *software* PLC or soft-PLC is a PLC running on a general purpose computer and has identical functionality to a *hardware* PLC. running on a commercial off-the-shelf quad-core x86 processor and Windows XP with real-time extensions [7]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2014.2299731, IEEE Embedded Systems Letters

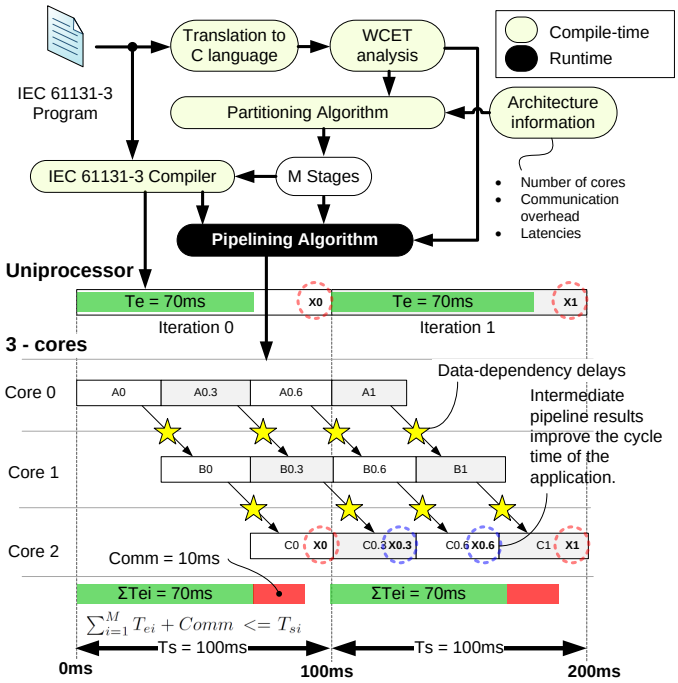


Fig. 3. Our pipelining approach uses a time-based optimization to partition the IEC 61131-3 application into M number of stages. The partitioned application is scheduled to different threads running in different cores and by streaming data over the pipelined application, additional intermediate results are produced and this effectively reduces the cycle-time of the application.

on a Structured Text (ST) application benchmark that tests the integer and floating point performance, control flow, memory bandwidth, and I/O capabilities of commercial PLCs. We evaluate the pipelining technique with respect to SCT and HTC, two of the most important KPIs in PLCs.

Figure 4 compares the SCT of four programs (X-axis). Sequential represents uniprocessor execution and it serves as our baseline because most PLCs have single-core processors; Functional(3) represents the state-of-the-art parallelization approaches based on data dependencies; Pipeline(3) and Pipeline(4) represent the programs generated using our pipelining approach. The "(3)" and "(4)" specify the number of cores used to execute the application. Each application contains two columns: the cumulative execution breakdown per stage, and the measured SCT. The baseline is the Sequential with SCT=120ms executed in a single stage corresponding to the critical path. The Functional(3) application has a SCT=70ms, this is 1.71× improvement over Sequential. Notice that because the functional parallelization relies on data dependency analysis it is heavily dependent on the topology of the application and partitioning typically results in unbalanced stages. The data dependency analysis discovered three stages of 54ms, 30ms, and 30ms in the test application. The SCT difference between 70ms and 54ms of the critical path is the *sleep time*, and run-time checks to guarantee hard real-time execution. In comparison, our pipelining algorithm is insensitive to the application's data dependence topology but sensitive to the accuracy of the timing analysis. For the same number of cores, 3, it creates three stages of 49ms, 32ms, and 40ms. Although not perfectly balanced, the critical path is shorter than the functional parallelism analysis and a SCT=52ms is achieved, this is 2.30× improvement over Sequential and 1.34× improvement over Functional(3) using the same number of cores. Pipeline(4) shows that SCT can be reduced further when more pipeline stages are created. In this case, four balanced stages of 33ms (critical path), 29ms, 34ms, and 30ms are created. The measured SCT is 40ms and equivalent to 3.00× improvement over Sequential, 1.75× improvement over Functional(3), and a 1.30× improvement over Pipeline(3). Notice that the difference between the SCT and the critical path for Pipeline(3) is significantly less than for Pipeline(4). This is due to the increase in communication cost associated with adding more pipeline stages. In summary, our pipelining approach effectively reduces the SCT of PLCs by 2.30× using 3 cores, and by 3.00× using 4 cores, compared to existing state-of-the-art commercial implementations of PLCs that use single-core.

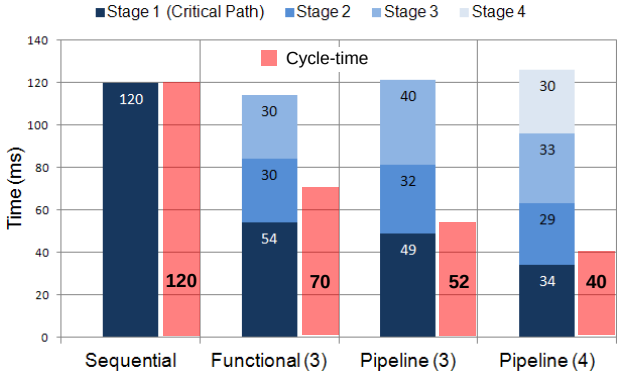


Fig. 4. Breakdown of execution time per stage including the critical path and the measured SCT.

To evaluate the performance of the TCP/IP stack to maintain a HTC with the sensors and actuators, we analyzed the window size field included in each TCP header. This field reflects the available buffer size: 0 bytes indicate that the buffer is full and no storage is available, and 65535 bytes indicate an empty buffer and maximum availability. When the system is busy, incoming packets are stored in the buffer for later processing and therefore the buffer size is reduced. Figure 5 shows our measurements for the Functional(3), Pipeline(3), Pipeline(4), and Pipeline(4) with balanced stages. For the Functional(3) and Pipeline(3) cases, we assigned the 3 user application stages to 3 different cores, and assigned the TCP thread to the fourth core. As expected, the window size for the two applications shows that the buffer is kept empty for the whole execution of the application because the TCP thread is running on a dedicated core. Pipeline(4) highlights the case when the TCP thread cannot have its own dedicated core. In this case, one core is time-shared between the user application and the TCP thread and this has direct negative impact in the system-level communication performance. During Pipeline(4) execution the buffer is, on average, filled at 23% of its capacity throughout the execution of the application and frequently exceeding 50%. A closer look to the implementation and profile data explains why this occurs. The CPU executing a pipeline stage is maintained occupied trying to dequeue data from the previous stage or I/O with a busy-waiting loop. Although other implementation alternatives exist, we rely on lock-free queues optimized for the inter-processor communication mechanisms available in existing multi-/many-core processors to achieve maximum SCT reduction in our pipelining approach.

We take advantage of the imbalance in the pipeline stages to create additional opportunities for the system-level threads to utilize the shared CPUs. We perform a *time slack pipeline balancing* [9] on the short stages and *normalize them to the length of the SCT* by idling the CPUs instead of busy-waiting. These idle periods are utilized to improve the performance of the system-level threads. Pipeline(4) with balanced stages in Figure 5 shows that this technique significantly improves the performance of the TCP thread that maintains, on average, the buffer filled at 7% and contained within 30% of its capacity throughout the execution of the application. It is important to note that the SCT remains identical to the original non-balanced version (40ms) because the balancing affects the small stages and not the critical path.

Table I shows the effects of time slack pipeline balancing on the TCP performance. Notice that the balancing of short pipeline stages to the length of the SCT improves the average TCP buffer availability by 16% and the variability of the buffer utilization is reduced from 14Kb to only 727b. In our experiment, a total of 22ms (5ms + 9ms

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2014.2299731, IEEE Embedded Systems Letters

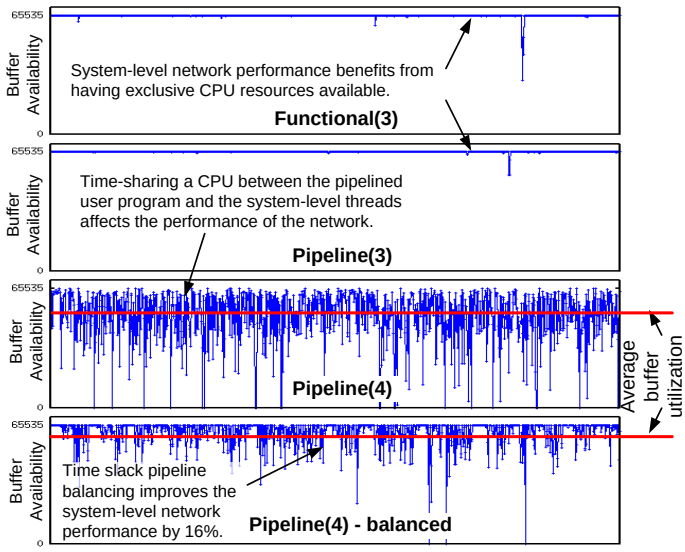


Fig. 5. Dedicated cores for system-level services such as TCP/IP are beneficial. However, in a pipelined execution the CPUs are fully utilized and have to be time-shared with system-level threads. Our communication-sensitive approach uses a time slack pipeline balancing to improve the performance of system-level threads (e.g. TCP/IP).

+ 8ms) of slack time per iteration is leveraged to improve the TCP thread without affecting the SCT.

TABLE I
EFFECTS OF TIME SLACK PIPELINE BALANCING ON SYSTEM-LEVEL COMMUNICATION PERFORMANCE.

	Pipeline(4)	Pipeline(4) balanced
TCP buffer (avg)	50Kb	61Kb
Std. deviation	14Kb	727b
Stage 1 (CP)	34ms	34ms + **0ms** = 34ms
Stage 2	33ms	33ms + **5ms** = 38ms
Stage 3	29ms	29ms + **9ms** = 38ms
Stage 4	30ms	30ms + **8ms** = 38ms
SCT	40ms	40ms

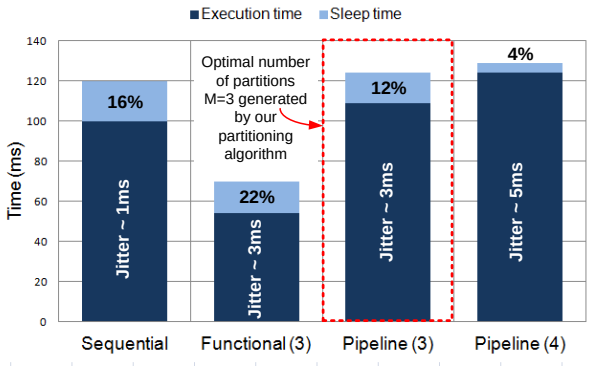


Fig. 6. User application performance per iteration (average of 1e7 iterations). The overhead of pipelining due to inter-processor communication is 4ms and 9ms when 3 and 4 cores are utilized and although jitter increases, it only affects the slow stages and therefore it is hidden behind the critical path. Although the execution time of **one iteration** slightly increases, the SCT is not affected (40ms).

Finally, we analyze the overhead of our approach. Figure 6 shows the *performance per iteration* (in milliseconds) achieved for the four versions of the application. Notice that Functional(3) parallelization effectively improves the execution time per iteration when compared to the Sequential execution from 120ms to 70ms using 3 cores. Functional parallelization relies on the Bulk Synchronous Parallel (BSP) model [6] exploit intra-iteration parallelism and thus the execution time per iteration is reduced. The Pipeline(3) and Pipeline(4), on the other hand, increase the cost per iteration to 124ms and 129ms. This is expected as pipelining exploits inter-iteration parallelism and its benefits must be quantified using other metrics (e.g. SCT); however, these results show that the overhead of synchronization and communication in a pipeline with 3 and 4 stages is about 4ms and 9ms, respectively. Additionally, the measured jitter is 1ms for the Sequential, 3ms for the Functional(3) and Pipeline(3) versions, and 5ms in the Pipeline(4) version. Interestingly, the jitter affects the short stages in all the parallel versions and therefore it is hidden behind the critical path. It is also important to note that Pipeline(3) is the optimal partitioning generated by our algorithm given the constraint $ST >= 10\%$ and it achieves a $ST = 12\%$, whereas Pipeline(4) achieves a $ST = 4\%$.

IV. Related Work

Previous work in parallelization for PLCs [2], [3], [5] focuses exclusively on SCT reduction and does not take into consideration the higher demand for communication due to the faster sampling rate. Current industry practice shows that it may lead to CPU starvation for the communication processes in a PLC, and this defeats the purpose of having a reduced SCT because the PLC is unable to receive/send data from/to the sensors/actuators fast enough. The major benefit of our approach over the existing work is that our pipelining technique is aware of both SCT and HTC KPIs to achieve a better balanced execution and CPU utilization.

V. Summary

This paper presents a novel pipelining technique that focuses on reducing the cycle-time of the PLC application by creating intermediate iterations and overlapping their execution in multiple cores over time while optimizing the high-throughput communication. This effectively exploits inter-iteration parallelism and although the total execution time of one iteration is greater due to the cost of synchronization and communication, the pipelined execution effectively reduces the SCT of the application. Faster SCTs are beneficial for many industrial applications as their precision can be greatly improved. Our implementation shows a 2.30× speedup when using 3 cores, and 3.00× speedup when using 4 cores with respect to the benchmark applications executed in a single processor. In addition, we have presented an approach to utilize the slack time of pipeline stages to improve the system-level communication performance without hurting the SCT in industrial applications.

References

[1] Beckhoff. TwinCAT 3. http://www.beckhoff.com.
[2] A. Canedo and M. A. Al Faruque. Towards parallel execution of IEC 61131 industrial cyber-physical systems applications. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 554–557, 2012.
[3] A. Canedo, L. Dalloro, and H. Ludwig. Pipelining for cyclic control systems. In *Proceedings of the 16th International conference on Hybrid systems: computation and control*, HSCC '13, pages 223–232, 2013.
[4] X. feng Wan, Y.-S. Xing, and L. xiang Cai. Application and implementation of CAN bus technology in industry real-time data communication. In *Industrial Mechatronics and Automation, ICIMA*, pages 278–281, 2009.
[5] J. R. Guo, F. Ran, Z. Bi, and M. H. Xu. A compiler for Ladder Diagram to Multi-Core Dataflow Architecture. *Advanced Materials Research*, 462:368–374, 2012.
[6] Q. Hou, K. Zhou, and B. Guo. BSGP: bulk-synchronous GPU programming. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 19:1–19:12, 2008.
[7] IntervalZero. RTX hard real-time platform. http://www.intervalzero.com.
[8] R. Pigan and M. Metter. *Automating with Profinet: Industrial Communication based on Industrial Ethernet*. Publicis Publishing, 2nd edition, 2008.
[9] A. Tiwari, S. R. Sarangi, and J. Torrellas. Recycle: pipeline adaptation to tolerate process variation. In *Proceedings of the 34th International Symposium on Computer Architecture*, ISCA '07, pages 323–334, 2007.
[10] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto, and C. Zunino. Real-time ethernet networks for motion control. *Comput. Stand. Interfaces*, 33(5):465–476, Sept. 2011.