1 Publishable Summary

1.1 VELOX Motivation and Goals

The current trend in designing processors with multiple cores, where cores operate in parallel and each of them supports multiple threads, makes the case for a broader use of parallel programming. The success of these multi-core architectures crucially depends on the ability of software programmers to harness parallelism within their software. So far, locks have been used as the common coordination mechanism for concurrent programming. Unfortunately, lock-based programming is not a perfect solution. Coarse-grained locking is easy to program but scales poorly with the number of cores because of limited parallelism. Programs using fine-grained locks can perform exceptionally well, but designing them has long been recognized as a difficult task – better left to experts. Finally, even when programmed optimally for a given architecture, lock-based code might not scale when moved to machines with a different memory layout or more cores.

Therefore, there is a need for a new approach to multi-core programming that retains scalability and, at the same time, preserves the ease of programming with coarse-grained locks.

1.1.1 TM: A New Paradigm for Concurrent Programming

Transactional Memory (TM) is a strong contender to be the paradigm of choice for replacing, or complementing, locks in multi-core programming. It greatly simplifies the parallelization of existing single-threaded code by eliminating the need to explicitly write fine-grained-lock-based code. Programmers simply write code in which methods or blocks of code accessing shared data are declared as high-level transactions, typically using *transaction block* language constructs. The synchronization and coordination details are left to the underlying TM mechanisms.

Transactions allow developers to indicate that code sections must execute atomically, that is, as if they were run alone in isolation from the rest of the code. TM is a speculative execution mechanism where accesses to shared objects are allowed to run simultaneously. In case of conflicting accesses, detected at runtime, one or several transactions may have to roll back and restart their execution. In other words, transactions execute optimistically in parallel and, as long as conflicts are rare, the performance gains resulting from the higher level of concurrency dominate the overheads introduced by the support for transactional execution.

TM executes in hardware, in software, or as a combination of both. As transaction block constructs are typically provided at the programming-language level, TM support spans across the **complete computer stack** from applications down to hardware, encompassing language extensions, compilers, libraries, TM runtimes, and operating systems.

The high level objectives of the VELOX Project can be summarized as follows:

- ***Objective 1:*** To show proof of concept for TM via an *Integrated TM Stack* that demonstrates ease of programming and performance scalability.
- ***Objective 2:*** To conduct frontier research that proposes solutions to TM Challenges; produce prototype Applications, System SW and HW that showcase these solutions.
- ***Objective 3:*** To provide input into TM Standards definition (at the level of language and library extensions) for industry based on the successes and failures of Objectives 1 and 2.

The main outcomes of this project will be:

- A TM-based reference software stack that will help European companies (i) develop new software for multi- and many-core architectures and (ii) convert their existing sequential or lock-based code into TM-based software that is more scalable and easier to maintain.
- A set of long-term design guidelines for hardware and software systems to improve the efficiency of TM-based applications running on many-core systems.

1.1.2 The VELOX Stack at a Glance

The VELOX Stack (see Figure 1) supports two families of programming languages, C/C++ and Java, and their associated runtimes, libraries, OS modules and Hardware simulators. Several cross-cutting research challenges have been studied in the context of the project to drive the design and development of the stack components. The VELOX Team includes internationally recognized TM experts in each of those components. These fully integrated TM systems will not only improve the understanding of TM designs, but will greatly help in the adoption of the TM paradigm by the European software industry, making it a tool-of-choice for concurrent programming on multi-core platforms.

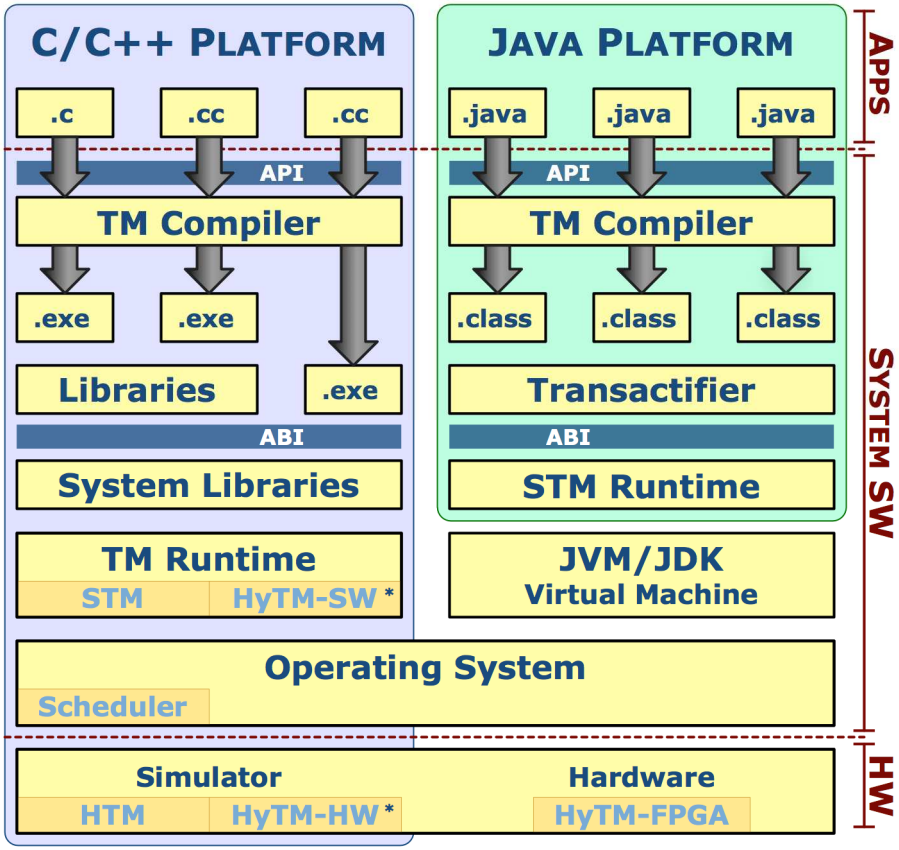

Figure 1: VELOX Stack

The VELOX Stack consists of the following basic components:

- The existence of **TM applications** is critical to making a case for the up-take of Transactional Memory by the programming community as well as system integrators. Although a few TM applications were available prior to the VELOX Project, these applications were developed by converting previously existing versions of lock-based applications instead of being written from scratch. Moreover, these applications were and

still are limited in size and complexity. The VELOX Team aims to develop real, complex TM applications using the TM programming model wherever possible.

- **TM language extensions** and programming language constructs also referred to as **APIs** are the most visible aspects of TM for the programmers. While it is possible to add TM support to applications by only using explicit library calls or declarative mechanisms (e.g., annotations), such an approach is not satisfactory for large systems: it relies on coding conventions, may lead to intricate code and is typically error prone. On the other end of the spectrum, automated source or binary code instrumentation work on simple examples but are difficult to extend to realistic codes. The **ABI** specifies the interface between the compiler and the TM runtime, i.e., the set of function calls that are generated by the compiler, which the TM runtime library must implement. The addition of new language constructs with well-defined semantics is the soundest approach to import TM support into existing languages. The VELOX Project focuses on two families of languages: C/C++ and Java.
- **Compiler support for TM** vis-à-vis **TM-enabled compilers** is necessary for implementing an efficient TM stack in a way that is transparent to the programmer. For instance, the TM-enabled compiler can identify transactional load and store operations on shared data and map these operations to the underlying TM, without requiring the programmer to explicitly mark these operations. Moreover, the compiler will automatically employ TM-specific optimizations that improve the performance of transactional code and reduce the overheads of transactional memory accesses.
- To better exploit TM, **system libraries** need to be adapted so that they can execute speculatively inside transactions despite performing potentially unsafe operations (e.g., I/O). Where applicable, one can also replace locks by transactions within libraries for better performance.
- The **TM runtime (also referred to as runtime library)** is the central component of the TM integrated stack. It implements the synchronization logic of the TM. The VELOX Project proposes several such libraries. A first class uses software transactional memory (STM) only, e.g., TinySTM. A second class uses a mix of software-based support of transactions with specific hardware extensions for scalability and efficiency. The VELOX Stack uses AMD's Advanced Synchronization Facility (ASF) mechanism to build such a Hybrid TM (HyTM). Finally, the TM stack can rely on a pure hardware approach where the TM mechanisms are completely implemented in hardware (HTM).
- **Operating system** (OS) **extensions** can help improve the performance of TM for some tasks that relate to the system as a whole (e.g., scheduling), in particular because transactional workloads co-exist with traditional ones. A typical example in the context of the VELOX Stack is the support of TM-aware scheduling in the Linux kernel to avoid pre-empting threads inside transactions and serialize conflicting transactions on the same core.
- Finally, the complete stack builds upon a **simulated TM hardware platform** that provides the necessary support for designing efficient TM implementations. Synchronization facilities such as AMD's ASF are used for supporting a large part of the application's transactional execution in hardware while relying on software for non-supported transactions. At the same time, the stack is partially integrated with the VELOX multi-FPGA multiprocessor prototype which supports a Hybrid TM proposal modelled after the public ASF specification (AMD).

1.2 Work Performed and Main Results

In the first year of the VELOX Project (P1), the VELOX Team focused on implementing experimental ideas in the initial versions of the building blocks for the VELOX Stack, composed of the TM applications, the TM-enabled compilers, the TM runtimes, the operating system extensions and the simulated TM hardware platform. At the same time, we began to examine the higher level language constructs (APIs) as well as the compiler level interfaces (ABIs) to determine where and how we might generalize TM concepts. In the second year (P2), we focused on releasing stable versions of these building blocks as well as beginning the integration of the VELOX Stack. In the third and final year of the project (P3), we concentrated on integrating the stable versions of these building blocks into a complete stack.

For example, we modified the **TM runtime**, the **TM-enabled compilers** (WP3 and WP4) and the **simulated HW platform** (WP2) in order for our novel HyTM solution (developed in P2) to support our **TM C applications** (WP6) in Release 2, the first Integrated Release of the VELOX Stack. In a follow-on release, Release 3, we added support for STL and exception handling to the **TM runtimes** and **TM-enabled compilers** to further integrate the stack with our **TM C++ applications.** We continued to modify the **TM runtime** to support the **ABI Specification** developed by WP3 and at the same time modified the **TM applications** to support the **APIs** developed by WP5**.** This tight coordination between Work Packages resulted in two successful releases of the integrated stack in the third year of the project.

At the VELOX Stack component level, our achievements can be broken down as follows[2]:

- (WP6) In P2, we worked to address the urgent need for new **TM applications** by releasing several new applications while in P3, we turned our focus toward integrating these applications with the VELOX Stack. First, we integrated C Applications (*RMS-TM C subset, QuakeTM, TMunit)* with the VELOX Stack in Release 2. *QuakeTM* is a game application that uses TM for synchronization, and *TMunit* is a TM stress benchmark application. Next, we integrated our C++ Applications (*STMBench7*, *RMS-TM C/C++ Full*) and our game application *Globulation2* in Release 3 (WP6). *Globulation2*, like *QuakeTM* is also a game application, but it is a real-time strategy game application that parallelizes the strategy engine using TM for synchronization. Both *STMBench7 C++* and *RMS-TM C/C++ Full* can be run on the VELOX HyTM (Hybrid) and VELOX HTM platforms. Moreover, one of the benchmarks of the *RMS-TM C Subset*, a new scalable TM benchmark suite, can run on the Release 3 FPGA prototype.
- (WP5, WP3) In P3, the VELOX made further progress in defining **TM language extensions** in both the *VELOX API and ABI Specifications*. Moreover, we continued to play a significant role in influencing *TM ABI and API Standards* that are currently being developed by key industrial players. Perhaps the most important breakthrough of the period, though, is that RHAT has put a plan into place to ensure that the next official "stable" version (v4.7) of the GCC Compiler (arguably the most widely used open compiler framework in the world) supports the TM extensions defined in the latest versions of these standards and is fully compatible with the VELOX Stack. The code to be included in GCC v4.7 will be based on the gcc-tm Compiler, the transactional memory branch of GCC, which is the direct result of the VELOX Project. Finally, the Java TM language extensions in VELOX have been defined to map as closely as possible to the C/C++ TM Standards.

[2] The names of products (or software generated) for the VELOX Project research have been *italicized*.

- (WP3, WP4) In P2, we released new versions of the **TM runtimes** and the **TM-enabled Compilers** and integrated them with the rest of the VELOX Stack. We implemented support for irrevocability in the Java **TM runtime,** *Deuce* in R2 and integrated an enhanced version of Deuce in R3 with our Java **TM-enabled compiler**, *TMjava*. For P2, the major goal for the C / C++ Runtime *TinySTM*, was to add appropriate support for C Applications and HyTM extensions in Release 2 while in Release 3, we focused on adding support for HTM extensions, soft real-time scheduling and C++ applications (including STL and exception handling). This final addition completed the integration of the *TinySTM* with the rest of the VELOX Stack. At the same time, we added C and C++ language support to both of our C/C++ **TM-enabled compilers**, *DTMC* and *gcc-tm*, which are now fully compatible with the *VELOX ABI* and *API Specifications*.
- (WP3) We continued to adapt the **system library**, *dietlibc C library*, which is built on top of the **TM-enabled compiler**, *DTMC* and **TM runtime,** *TinySTM* and reviewed its performance with various test cases; this included a successful test with one of the VELOX Applications. At the same time, Gibraltar, the binary transactification tool, was further extended in P3 to support the VELOX ABI. Both the library and tool can be run with Release 3 of the VELOX Stack.
- (WP2) We released a version of each of the simulated TM hardware platforms for HTM and HyTM; both of these are integrated with VELOX Applications RMS-TM (C/C++) and STMBench7 (C/C++). These platforms support detailed TM application analyses and the identification of bottlenecks; they were expressly used to verify new ideas for hardware support to increase TM performance and power efficiency. Moreover, we released a version of our VELOX FPGA multiprocessor prototype (BeeFarm) that supports TinySTM and runs a VELOX Application, featuring a HyTM Hybrid TM proposal modeled after the public ASF specification (AMD).

In addition to our successful integration of nearly all components into the VELOX Stack, we celebrated a successful year in research. We published in highly selective workshops, conferences and journals including: **TRANSACT, PPoPP, MICRO, OPODIS, ICPE, ARC, LCPC, SPAA, SSS, DISC, PODC, PACT, EuroPar, SAMOS, PPL, IEEE MICRO magazine and Communications of the ACM**. The VELOX Project was also prominently featured during the HiPEAC Systems Computing Week which included a VELOX Tutorial, various talks and invited participation in a forum on Recent FP7 Successes in Multi-core Computing. Finally, the VELOX Team submitted and was accepted to prepare a final dissemination of the VELOX Stack in a tutorial on the VELOX Stack at PLDI 2011.

More information about the VELOX Project and access to our publicly available deliverables can be found by visiting our website: www.velox-project.eu or by contacting the coordinator at velox-coordinator@bsc.es.

2 Project Objectives for P3

The following table summarizes the top VELOX Research Challenges as described in the Description of Work. The table also includes our proposed “VELOX Solutions” which represent the highest level objectives of the project.