# Auto-Vectorization through Code Generation for Stream Processing Applications

Huayong Wang
IBM China Research Lab
huayongw@cn.ibm.com

Henrique Andrade, Buğra Gedik,
Kun-Lung Wu
IBM T. J. Watson Research Center
hcma@us.ibm.com,
bgedik@us.ibm.com, klwu@us.ibm.com

## ABSTRACT

We describe language- and code generation-based approaches to providing access to architecture-specific vectorization support for high-performance data stream processing applications. We provide an experimental performance evaluation of several stream operators, contrasting our code generation approach with the native auto-vectorization support available in the GNU `gcc` and Intel `icc` compilers.

## Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features—*Frameworks; Modules, packages*

## General Terms

Languages, Performance, Experimentation

## 1. INTRODUCTION

Streaming applications can be extremely challenging to design and implement. The design process has to cope with difficult issues associated with building distributed applications, coupled with the need for careful optimization of the performance-critical sequential portions of these applications. We have found that high-speed stream processing applications present an interesting combination of challenges from a performance optimization standpoint. Two important issues stand out: (1) handling the stringent requirements to cope with high data ingest rates and (2) handling analytics that rely heavily on vectorized processing.

In this work, we demonstrate an effective approach to addressing the second challenge by code generation and auto-vectorization support. We describe language and library support for developing streaming applications in SPADE, a high-level programming language that provides, among other features, auto-vectorization to application developers. In particular, we discuss a two-tier approach for making better use of SIMD instructions in the context of stream processing applications. We argue that this approach is superior to directly using *intrinsics*, which requires application recoding. We provide case studies with empirical evaluations of real-world stream processing operators, demonstrating the actual improvements that can be derived by making use of transparent vectorization when adding new operators to extend SPADE.


*ICS'09,* June 8–12, 2009, Yorktown Heights, New York, USA.
ACM 978-1-60558-498-0/09/06.

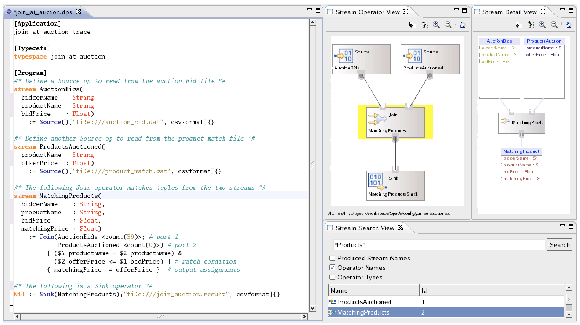


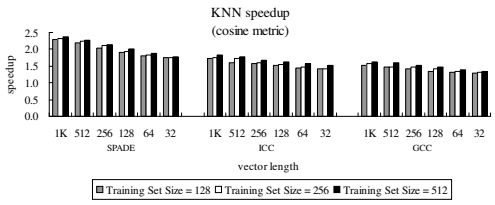
**Figure 1: The rapid application development environment for System S**

## 2. SYSTEM S AND SPADE

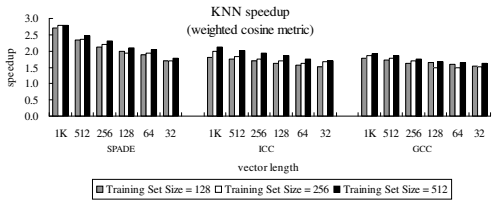
System S [3] is a large-scale, distributed data stream processing middleware under development for the last 5 years. It supports structured as well as unstructured data stream processing and can be scaled to a large number of compute nodes. The System S runtime can execute a large number of long-running applications that take the form of flow graphs.

SPADE [1] (Stream Processing Application Declarative Engine) is the stream processing application development framework for System S. SPADE provides a rapid application development environment including design and debugging tooling as seen in Figure 1. The SPADE compiler makes use of several optimizations. First, code fusion, i.e., the ability to translate the logical description of an application in terms of operators into a set of processing elements (PEs) such that multiple operators may be placed inside a single processing element and the streams between them are converted into direct function calls. Second, transparent multi-core adaptation, i.e., the ability to dynamically tune data parallel operators to exploit additional processing resources in reaction to changes in resource availability or changes in the workload. Third, auto-vectorization, i.e., the ability to express an operator's vector-heavy internal computation in terms of low-level vector operations, translating into code that employs SSE or AltiVec SIMD instructions. The design for this feature is the central contribution of this work and we discuss it next.

The SPADE application support framework utilizes a very minimal runtime system that enables the creation of System S processing elements and their interconnection. The actual application processing is carried out by custom-made instantiations of the operators used in the application. Each




(a) Cosine measurement




(b) Weighted cosine measurement

**Figure 2: KNN speedup**

code generator is tasked with specializing the template code associated with an operator's code generator based on the operator parameter configurations as well as environmental configurations. Most programmers rely solely on this so called SPADE *programming tier*, writing their applications completely in the SPADE language.

The *toolkit programming tier*, which is not normally seen by application developers, allows SPADE to be an extensible language. New operators can be added to the language as needed, seaminglessly extending its syntax. To support the addition of new operators, considerable infrastructure is provided. The SPADE compiler libraries provide services that enable the integration of the master compiler with new operator-specific code generators. Each code generator is defined by an operator model describing, for example, how many streams an operator produces, how many streams it consumes, what are the valid configuration parameters, what are the constraints on these parameters, among others. The operator model is used by the master compiler to perform syntax checking as it compiles an application. Each code generator is also informed about the architecture-specific configurations and can thus specialize the code accordingly. Specifically, since vectors are first class types in the SPADE language, operator writers can make use of a templatized vector manipulation class when implementing operators that manipulate vectors. Two implementations of this class are provided as part of the operator building support library: a scalar and a vectorized one, hand-written using *intrinsics* where vector operations are required.

# 3. EMPIRICAL EVALUATION

We employed 3 empirical configurations to evaluate our auto-vectorization approach. We defined a *scalar* implementation as our baseline and compiled it with `gcc`. The same code used for the *scalar* implementation was also built with the C++ compiler-based auto-vectorization feature turned on, by employing the appropriate command-line switches. Two compilers were used, the Free Software Foundation `gcc` [4] (version 4.3) as well as Intel's `icc` [2] (version 11.0). Finally, we made a single change in terms of the operators' source code. We employed a version of the vector manipulation class that was written using the `gcc` *intrinsics* and, naturally, we built the code using the `gcc` compiler. This approach is labeled with the word "SPADE". We measured the *throughput* of the benchmark applications. The ratio between the observed throughput of one of the auto-vectorization approaches (either the C++ compiler-based or the SPADE-based) to the *scalar* version is the speedup we report. The experiments were run on a node with an Intel Core2 Duo processor 6700 running at 2.66 GHz, with 32 KB L1 data cache per core, shared 4 MB L2 cache, and SSE3 support, running Linux.

While we looked at several applications, due to space constraints, we describe the results obtained for only one of them. It employs a KNN (K-Nearest Neighbor) operator, an important classification algorithm in data mining [5]. The algorithm classifies an object based on knowledge gleaned from a set of training objects. Figures 2(a) and 2(b) show the speedup curve as a function of the feature vector length for KNN, when using the *cosine* and the *weighted cosine* as the similarity metric, respectively. As expected, the more elements in the vector, the larger is the speedup. Also, the weighted cosine measurement has better speedup characteristics since it carries out more vector operations compared to the simpler algorithm employing the cosine measurement. In most cases, more speedup is observed for larger training sets, as, in general, more vector-heavy work must be carried out. The exception is for smaller vectors when using `gcc` and the weighted cosine metric.

# 4. CONCLUDING REMARKS

Many streaming applications require vector manipulation. We have shown how SPADE, a language used for developing complex streaming applications, employs code generation as well as a templatized vector class so that application and operator writers can reap the benefits of native SIMD instructions transparently. We have shown experimentally that our approach, in most cases, outperforms the auto-vectorization support from general purpose C++ compilers.

# 5. REFERENCES

[1] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. SPADE: The System S declarative stream processing engine. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 2008)*, Vancouver, Canada, 2008.

[2] Intel C++ compiler user and reference guides. Intel Document number: 304968-022US, 2008.

[3] N. Jain, L. Amini, H. Andrade, R. King, Y. Park, P. Selo, and C. Venkatramani. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 2006)*, Chicago, IL, 2006.

[4] D. Naishlos. Autovectorization in GCC. In *Proceedings of the GCC Summit*, 2004.

[5] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, January 2008.