

# Revisiting APL in the Modern Era

Aaron W. Hsu                      William J. Bowman  
awhsu@indiana.edu    wilbowma@indiana.edu  
Indiana University

**APL deserves another look.** Most people think of APL only peripherally as a language with terse syntax and strange symbols in its code. While programmers used to other programming languages may encounter an initial barrier due to these symbols, we believe that APL is worth the learning curve and that one quickly crests this particular hill. APL has a simple, consistent syntax and semantics coupled with an very rich vocabulary of built in primitives for manipulating arrays. The modern APL implementation encourages functional and points-free programming styles. APL's minimal syntax and rich vocabulary suggest its suitability for representing domain specific computations as well as serving as an common language for other, less concise notations. Furthermore, the features of APL encourage thinking about problems in implicitly parallel and aggregate ways, making it a powerful tool for talking about and writing parallel programs.<sup>1</sup>

**Parallel programming and APL.** Ubiquitous multi-core and parallel hardware should be accompanied by the software tools that enable us to think about problems in ways conducive to efficient execution on these platforms. A new wave of research on parallel programming has hit the programming languages field with the advent of modern hardware, but we believe that there is still much to be gleaned and learned from past research on parallel machines; APL is a real world, industrial language that is built on implicit parallelism, especially data parallelism.

We believe that APL is still ripe as a research target. APL's rich vocabulary and concise notation encourages a points-free and functional style where primitive operations are naturally and implicitly parallel. Important parallel techniques such as map and reduce see first-class, primitive support in APL's notation. Users are encouraged to use bulk, aggregate operations, and with the incorporation of features from other languages into APL, such as dynamic functions, APL becomes that much more friendly to the parallel programmer.<sup>2</sup>

**APL to the modern PL researcher.** New research into programming languages should be informed by existing research. APL is one of the few active, production-class languages that has seen as much research into its parallel potential. Leveraging this past research is important, but being able to build directly upon it with a common language seems to us an obvious step. Why then does APL not see more attention in the parallel programming languages fields?

**APL as an IR and DSL.** APL is useful as a language in its own right, but with a proliferation of parallel domain specific languages, including things like ArBB<sup>3</sup> and Copperhead,<sup>4</sup> APL presents a number of advantages to domain specific languages. Its simple syntax makes it easy to map problems in one DSL to APL and back. APL's vocabulary can provide a core calculus by which researchers may easily ground and relate their systems to others in the field. Moreover, APL can itself serve as an effective parallel programming DSL.

**Conclusion.** Researchers may or may not find APL suitable for use directly, but the long history of research on the language, its continued development and progress, and the different approach that the language takes from other languages makes it worthwhile to learn and study. Companies use APL to do real work in industry, and it makes sense to understand and evaluate APL before embarking on yet another parallel language.

<sup>1</sup> Willhoft, R. G. Parallel expression in the APL2 language. *IBM Systems Journal* 30, no. 4 (1991): 498--512.

<sup>2</sup> Scholes, John. Introduction to D-Functions. *Dyalog '09*.

<sup>3</sup> Newburn, C. J. et al. Intel's Array Building Blocks: A retargetable, dynamic compiler and embedded language. *IEEE/ACM Symposium on Code Generation and Optimization* (April 2011): 224--235.

<sup>4</sup> Catanzaro, Bryan, Michael Garland, and Kurt Keutzer. Copperhead: Compiling an Embedded Data Parallel Language. *PPoPP* (February 2011).