# Comparing Synthesizable HDL Design and Stream Programming

Jesse G. Beu
Georgia Institute of Technology
Jesse.Beu@gmail.com

Thomas M. Conte
Georgia Institute of Technology
Tom@conte.us

In recent years there has been growing concern over whether we as a community will be able to exploit future parallel architectures, specifically with regard to the programmer's ability to adapt. It is often argued that programmers are not well equipped to program in parallel and thus new paradigms are developed to ease the transition from conventional sequential semantics to more parallel friendly languages [1, 2, 6, 7]. One area that is often ignored in this quest for new tools, however, are synthesizable hardware descriptive languages (HDLs), such as Verilog HDL [5] and VHDL [4], where hardware engineers have been successfully implementing parallel algorithms for years. This brings about an interesting consideration: how does HDL design compare to stream programming?

One of the fundamental properties of stream programming is the decoupling of memory and computation through a cycle of data organization phases followed by kernel computation phases [3]. This results in a gather-operate-scatter type methodology for execution. A similarity to hardware design exists because one of the driving principals of good hardware design is efficiency, often obtained by keeping hardware busy at all times. This means being prepared to deliver data any time a resource is available, often resulting in extensive use of buffers and queues. As an interesting thought experiment, consider the design of a familiar piece of hardware, a CPU core. A considerable amount of resources are dedicated to the organization and pre-processing of data in order to stream it through small processing units (ALUs), then redistribute this data back to memory. This is reflected in the HDL model that represents a CPU core, which obviously mirrors the stream programming methodology of gather-operate-scatter.

Another important similarity between HDL and streams are the similarities between Synchronous Data Flow (SDF) graphs, which can represent stream programs, and pipelined block diagrams, a common tool used during the design of ASICs. Both break the problem into discrete pieces of work with specific inputs and outputs to arrive at a solution. Both are also a convenient way to represent parallel opportunity and pipeline-potential, important factors for parallelization in both hardware design and stream programming. Further, the hierarchical nature of block diagrams provides a natural representation of locality at multiple levels of granularity, another property often exploited by stream program compilers [1, 7].

A fundamental difference between these realms, however, is the fact that dynamic memory allocation is not an option in HDL since hardware cannot simply grow more silicon. This restriction means that HDL has very finite bounds on what it can and cannot do, which translates into restrictions on the kinds and sizes of data structures that can be employed and the amount of flexibility available; HDL algorithms often err on the safe side of simplicity and worst-case scenarios. Stream programs, being software rather than hardware, means not only are they free of these concerns but also have the additional benefit of flexibility in that they can be quickly modified and tuned as an application space evolves, something that is often not a consideration during hardware design. Additionally, since stream programmers are not restricted to a box, they are afforded the luxury of more complex and sophisticated techniques a hardware designer would never dream of implementing.

Hardware Descriptive Languages show us that designers are capable of algorithmically parallel thinking and have been successful at it for many years. Due to similarities with stream programming, there is hope for the widespread acceptance of stream programming in the future. Additionally, considering HDLs as a form of stream programming may enable the next generation of stream languages to learn from the success of synthesizable HDLs to create more user friendly, successful parallel languages for the future.

## 1. REFERENCES

[1] Buck, Foley, Horn, Sugerman, Fatahalian, Houston, and Hanrahan 2004. Brook for GPUs: stream computing on graphics hardware. In *ACM SIGGRAPH 2004 Papers* (Los Angeles, California, August 08 - 12, 2004). J. Marks, Ed. SIGGRAPH '04. ACM, New York, NY, 777-786.

[2] Gholoum, Sprangle, Fang, Wu and Zhou, "Ct: A Flexible Parallel Programming Model for Tera-scale Architectures", Intel Whitepaper, October 25, 2007

[3] Gummaraju and Rosenblum 2005. Stream Programming on General-Purpose Processors. In *Proceedings of the 38th Annual IEEE/ACM international Symposium on Microarchitecture* (Barcelona, Spain, November 12 - 16, 2005). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, 343-354.

[4] IEEE. IEEE standard VHDL language reference manual, IEEE Std. 1076-1993, 1994.

[5] IEEE std. 1364-2001, Verilog HDL Reference Manual, IEEE press, 2001.

[6] NVIDIA. 2007. CUDA Technology; http://www.nvidia.com/CUDA.

[7] Thies, Karczmarek and Amarasinghe, StreamIt: A Language for Streaming Applications, Proceedings of the 11th International Conference on Compiler Construction, p.179-196, April 08-12, 2002