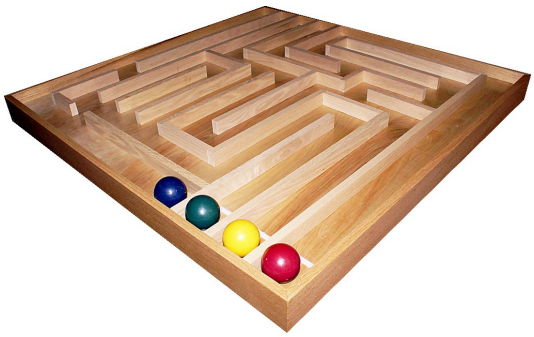
A Maze Of Twisty Little Passages

In Crowther's 1970s COLOSSAL CAVE ADVENTURE, whose layout happened to be partly modeled after Kentucky's MAMMOTH CAVE, you may recall two mazes: the original "all alike" one and an "all different" one that was added later. The same kind of distinction is commonly made in classifying modern parallel computing systems as SIMD or MIMD, and providing different, often mutually incompatible, programming environments for each. Is it really necessary to make such a stark distinction between the two?



Consider the wooden maze in our SC13 Research Exhibit. **Each of the colored balls has a different path to take (MIMD), yet it is perfectly feasible to efficiently get all the balls to their respective destinations by a series of tilts of the table (SIMD).**

GPUs (Graphics Processing Units). Modern GPUs are not exactly SIMD, using a model that avoids most scaling limitations of SIMD by virtualization, massive multithreading, and imposition of a variety of constraints on program behavior (e.g., recursion is not allowed by NVIDIA nor by AMD). This branch off the SIMD family tree has grown quickly, with new programming models and languages appearing at each new bud... but little code base and many portability issues. MIMD C, C++, or FORTRAN using MPI message passing or OPENMP shared memory are now the bulk of the parallel program code base, so we suggest using those – via the **public domain MIMD On GPU (MOG)** technologies we are developing.

SC08 MOG. The first proof-of-concept MOG system was demonstrated in our exhibit at SC08. Actually, there were two systems, one using an interpreter and another using META-STATE CONVERSION (MSC) to generate pure native code. Both shared the same MOG instruction set and C-subset compiler.

The Instruction Set Architecture (ISA) needs to make the performance-critical GPU features accessible while minimizing the number of different types of instructions in common use. Targeting NVIDIA CUDA GPUs led to a stack machine using shared memory to implement an explicitly-managed variable-depth stack cache. The C compiler we built for the system accepted and optimized only a subset of C supporting both integer and floating point data, the usual C operators and statements, recursive functions, etc. It also had a simple parallel-subscript language extension allowing direct access to other processing element's variables.

The simulator, called `mogsim`, targeted both NVIDIA CUDA systems and generic C hosts. It correctly handles recursion, system calls, breaking execution into fragments fitting within the allowable GPU execution timeout, etc. Code was stored in a texture, registers and an evaluation stack in device shared (local) memory, and the regular stack and user data in global memory. The simulator's fixed code was compiled with data structures generated by `mogasm`, our optimizing assembler. Multiple node programs can be compiled separately and integrated by the assembler for true MIMD (not just SPMD with MIMD semantics).

The MSC-based system avoids all overhead from use of GPU resources to fetch and decode instructions, but was somewhat buggy and has not yet been fixed.

SC09 MOG. Much more sophisticated analysis and transformations enabled `mogasm` to create a highly customized `mogsim` for each program – making MOG execution nearly as fast as native CUDA. Slowdown was generally less than 6X and often just a few percent. Actually, there were over a dozen completely different approaches tried for `mogasm` to achieve this performance, including optimizations based on runtime statistics, scheduling using a GENETIC ALGORITHM (GA), and even per-program automatic instruction-set recoding to improve runtime decode overhead.

SC10-12 MOG. The MOG environment using the best of the previous year's interpretation strategies was released at **`http://aggregate.org/MOG/20101122/`** on November 22, 2010 as full "alpha test quality" source code. However, it was actually a complete re-write with one goal in mind: make it possible to trivially port various full-language compilers to target our system. This is accomplished using a new, accumulator plus registers, ISA for which a nasty set of scripts can retarget generic MIPSEL code. The GCC-based version is called `mogcc`, and can process any of the languages that compiler supports. The new ISA enables more optimizations than the old one, and hence typically outperforms it by a small margin. The assembler, `mogas`, generates an optimized CUDA interpreter named `mog.cu`.

SC13 MOG. Work this year centered on fixing "bit rot" in the released code and bringing the host system call interface to a more usable level. General-purpose mechanisms for passing arguments and return values between code running inside the GPU and the host are now being tested, with the goal of making it easy for GPU code to call arbitrary functions on the host.

This document should be cited as:

```
@techreport{sc13mog,
author={Henry Dietz and Sam Morris and Iven Yang},
title={A Maze Of Twisty Little Passages},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc13mog.pdf},
month={Nov}, year={2013}}
```

SC13 Denver CO 2013 Aggregate.Org UNBRIDLED COMPUTING