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?

Take a moment to examine one of the little mazes in our SC14 exhibit. **Each of the colored balls has a different path to take – it's a MIMD program. Yet, it is perfectly feasible to efficiently get all the balls to their respective destinations by a series of tilts of the table – execution on SIMD hardware.**

We have been using MIMD-on-SIMD technologies for over two decades, targeting SIMD hardware from the MasPar MP-1 supercomputer to arrays of millions of tiny nanocontrollers....

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. 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, and a parallel-subscript extension for remote memory access.

The simulator, `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. 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 as full "alpha test quality" source code. Unlike earlier versions, it allows any compiler tool chain targeting MIPSEL to be used to compile your 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`.

SC13-14 MOG. Work has 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 have been implemented and tested using a set of system calls allowing GPU code to do file I/O. The new release is at **`https://github.com/aggregate/MOG/`**

What's Next? Current work involves improving overall usability, in particular the cross compiler and related build mechanisms. We also are building a small library of MPI system calls to make MOG more fully compatible with the MPI environment, and expect to have a GPU cluster running MOG MPI code soon.

This document should be cited as:

```
@techreport{sc14mog,
author={Henry Dietz and Sam Morris},
title={A Maze Of Twisty Little Passages},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc14mog.pdf},
month={Nov}, year={2014}}
```

SC14 New Orleans, LA | hpc matters.

Aggregate.Org UNBRIDLED COMPUTING