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 SC08 Research Exhibit (shown above). **Each of the colored balls has a different path to take (MIMD), yet it is perfectly feasible to 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 ATI). 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 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 our **public domain MIMD On GPU (MOG)** technologies overviewed here.

MOG Instruction Set Architecture (ISA). To be efficient, the MOG ISA must make the performance-critical GPU features accessible while minimizing the number of different types of instructions in common use. For example, our MOG ISA for NVIDIA CUDA targets is a stack machine using shared memory to implement a variable-depth stack cache.

MOG Compiler. Our goal is to compile unmodified C or Fortran programs with MPI and/or OpenMP communications. Our compiler currently accepts and optimizes only a subset of C supporting both integer and floating point data, the usual C operators and statements, recursive functions, etc.

MOG Simulator. Perhaps the most obvious way to execute MIMD code on SIMD hardware is to write a SIMD program that fetches and simulates the code compiled for a MIMD machine. Our implementation is called `mogsim`, and has been operational on NVIDIA CUDA systems, and generic C hosts, since October 2008 – usably fast since Summer 2009. The simulator correctly handles recursion, system calls, breaking execution into fragments fitting within the allowable GPU execution timeout, etc.

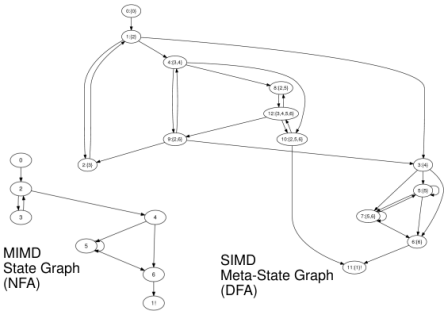
MOG Assembler. The simulator literally fetches instruction bit patterns from a GPU texture memory. Thus, the assembly code output by the compiler must be assembled into binary data. Our assembler, `mogasm`, does not require all simulated PEs to execute from the same code image; multiple node programs can be compiled separately and integrated by the assembler for true MIMD (not just SPMD with MIMD semantics). Currently, the output is a set of data structures compiled with the base `mogsim` code using the target system's compiler (e.g., `nvcc`) to produce a host-executable program.

MOG Meta-State Converter. Although the simulator uses a variety of technologies that make it surprisingly efficient in simulating MIMD execution, it still suffers a great deal of overhead from use of GPU resources to fetch and decode instructions. Using `mogmsc` instead of `mogasm` results in code that has *no* such overheads. How? **Meta-State Conversion (MSC)** literally transforms MIMD code into pure SIMD code by converting code into an NFA representing MIMD control flow, and then a DFA is constructed by enumerating sets of NFA states that could be simultaneously occupied by different processors in MIMD execution of the NFA. For example, a simple factorial program and its NFA and DFA follow:

```
/* Factorial in C... */
int a;

int fact(int i)
{
    if (i < 2) return(1);
    return(i*fact(i-1));
}

int main()
{
    a = fact(IPROC%10);
}
```



For more information, talk to us in our SC09 exhibit or see `http://aggregate.org/MOG/`

This document should be cited as:

```
@techreport{sc09mog,
author={Henry Dietz, Dalton Young, Diego Rivera},
title={A Maze Of Twisty Little Passages},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc09mog.pdf},
month={Nov}, year={2009}}
```

Aggregate.Org UNBRIDLED COMPUTING

SC Portland, OR 2009