Simple, efficient, HW/SW support for unstructured control flow.
Motivation

Execute arbitrary SPMD applications
- exploit SIMD efficiency.

Solved problems:
- structured control flow.
- some unstructured patterns (complex analysis required).

Unresolved issues:
- unstructured control flow.
- separable compilation.
- differences between SIMD and SPMD semantics
  - barriers.
  - fair scheduling.
1 History

2 Priority Reconvergence

3 Possible Hardware Implementation

4 Performance Comparisons

5 Conclusions

6 Extensions
The CRS Stack


The compiler determines re-convergence points (post-dominators).

HW detects divergent branches, pushes entries for branch targets.

Execute the entry on the top of the stack.

Pop the stack on re-converge (sync) instructions.
Straightforward for structured control flow.
Structured Control Flow

Hammock graphs: A subgraph with a single entry, and a single exit.

All hammocks are structured.
Unstructured Control Flow

A subgraph with multiple entry points, or multiple exit points, is not a hammock. It is not structured.

```c
if ( (cond1() || cond2()) &&
     (cond3() || cond4()) )
{
   ....
}
```

Unstructured control flow makes re-convergence **conditional**.

- Problem for stacks: a stack pop is unconditional.
Example

Start with an unstructured branch

Threads are divergent

End with the immediate post-dominator of the branch
Extended CRS Stack

We tried to address this in Fermi/Kepler
   (Lindholm & Coon 2007): re-convergence tokens

Main idea:
   - classify branches into categories
     - (e.g. loop exit points, exceptions, others)
   - ignore some categories when computing re-convergence points

Downsides:
   - analysis required to identify classes of branches
   - proof of generality?
Alternatives?
Invented in 1980 by Lorie and Strong at IBM.

Original formulation:
- assign priorities to basic blocks according to a depth-first traversal. *(sub-optimal)*
- on every cycle, execute all threads with the highest priority.
Constrained Scheduling

The program control flow structure restricts divergence.

Scheduling constraints (priorities) can be used to further restrict these locations:

- e.g. to minimize opportunities for divergence.
Definition: For any basic block BB, the complete set of other basic blocks where divergent threads may be executing.

Consequence: If a warp is executing BB, all divergent threads from that warp will be found in the thread frontier of BB.
Example

thread frontier of BB2

Gregory Diamos
SIMD Re-convergence at Thread Frontiers
Algorithm:
1. start from the CFG exit point.
2. construct the edge-covering tree from the exit point to all nodes.
3. the priority is the max distance from the exit point to each node.

Re-convergence always occurs at the earliest point.
Hardware Support

(priority sets)
Entries consist of [(PC) (Priority) (Predicate Mask) (Valid Bit)].

Support the following operations:
- compute the entry with the maximum priority.
- insert into a free slot.
- update (bitwise or) an entry with matching PC and priority.

Size to 4-8 entries.
- comparators find matching entries
- broadcast then bitwise or
- one read port for the entry with max priority
- overflow traps, or a state machine searches the next set
Branch edges that change priorities perform the following actions:

1. Invalidate the current predicate set entry.
2. Insert into the predicate set
   - If it matches an existing entry, merge them (bitwise or).
   - If it doesn’t match, create a new entry.
   - If there is no space, trap or check the next set.
3. Recompute the maximum priority.
Performance Study
Ocelot PTX Emulator

32-wide warp

simulated PTX instructions, in-order execution
Dynamic Instruction Counts

2.2-633% reductions in dynamic instruction counts
- compared to CRS without tokens
The predicate set typically has 1-2 entries

- we can prove that none of these applications will use more than 6
- the max thread frontier size is 5
The percent of updates that modify an entry other than the one with the highest priority.
Stepping back...
Conclusions

Re-convergence at thread frontiers.

Advantages:
- supports all unstructured control flow.
- incorrect analysis impacts perf, not correctness.

Disadvantages:
- new hardware, replaces CRS stack, (power cost?).
- priority assignment requires whole-program analysis.

Unresolved issues:
- separable compilation.
- differences between SIMD and SPMD semantics.
Prior Art

Patents

- predicate stack, ISA (LucasFilms 1982) (expired)
- priority re-convergence (IBM patent 1980) (expired)
- depth-first priority assignment (IBM patent 1980) (expired)
- re-convergence stack + token HW (NVIDIA patent 2005 & 2007) (valid)
- reverse post-order priority assignment (Intel patent 2010) (valid)
- linked list HW (Intel patent 2010) (valid)
- **edge-covering tree priority assignment (not patented)**
- **priority entry set HW (not patented)**
Questions?

Reference:
http://gdiamos.net/papers/thread-frontiers.pdf

Contact Me:

gdiamos@nvidia.com
Extensions

MIMD semantics on SIMD hardware

Handling dynamically linked libraries

Loosely-coupled SIMD