Supporting Virtual Memory without Precise Exceptions in GPGPUs

Hyesoon Kim
Motivation

Supporting precise exception on GPGPUs is very expensive.

- Too many in-flight instructions
  - Problem if we use a ROB based structure

- Too many registers
  - Problem if we use a check-point based mechanism
  - E.g. NVIDIA G80: 64KB register file
Why Precise Exception?

Support debugging
- Software support is limited

Support context switch
- Support virtual memory
- Exception handlers

Do we need to handle all cases?
- Supporting virtual memory is critical
Proposal: Guarded Execution

Support only virtual memory using predicated execution

- Debugging support can be implemented using barrier feature
- Some of interrupt/exceptions can be handled in a host processor.

Basic idea

- Marker:
  - Execution barrier
- The code after execution barrier are protected with predication (**predicate bit is set if there is page fault**)
- The penalty of execution barrier is not significant because the processor has multi-threading
Basic Mechanisms

ISA extensions

set start_marker

LD.pfchk, ST.pfchk

- Set pfbit (page fault)

sw_call.start_marker: barrier + function calls

Page miss sets predicate values

All live-out instructions are guarded with predicate values

Stop at sw_call.start_marker and check predicate bits.
Re-execution Based Approach

set start_marker;
inst1
LD.pfchk.br mem[R1];
inst2;
(pfbit)inst3; // Live-out variable producer
(pfbit)inst4; // Live-out variable producer
inst5;
(pfbit)sw_call(start_marker); // call exception handler;

// sw_call()
// re-start from the start_marker
//
Conclusion and Future Work

Precise exception support is not necessary for GPUs. Predicated execution can be a solution to support virtual memory.

In the poster, code examples (array, pointers, and etc.)

Optimization techniques

Future work:

- Develop compiler algorithms
- Combining with iGPU (software marker approach)
Large Arrays

/* original C-code */
for (int ii=0; ii<N; ii++)
a[ii] = b[ii]*2;
}
/* new code */
for (int ii=0; ii<N; ii++)
if (!((ii%kk))) {
// kk = page size%(size of(a[0]))
pfchk(&(a[0])+ii*kk));
pfchk(&(b[0])+ii*kk));
}
a[ii] = b[ii]*2;
}
void pfchk(int addr)
{
/* use intrinsics to insert assembly code */
set start_marker;
LD.pfchk(addr);
(pfb) sw_call(start_marker);
}