The von Neumann Architecture is due for Retirement

Aleksander Budzynowski, Gernot Heiser
NICTA and UNSW
The von Neumann Bottleneck

Fundamental problem:

- Conceptual model that all data goes to/from memory
- Random-access memory leads to uncontrolled communication
  - expert coding needed to avoid this
- Hardware provides shortcuts (cache-cache transfer, message passing)
- Expert coding or expensive re-discovery of parallelism by HW
  - ILP easy to discover, TLP hard
- The model *hides* the parallelism

The von Neumann architecture is a poor *bridging model* for modern hardware
A Better Bridging Model

Desired Properties:
1. Expose parallelism
2. No global addresses/communication
3. Support CS-type abstractions

How about dataflow?
4. ✔ excellent match
   - used a lot inside the hardware
5. ✔ for pure dataflow
6. ✗ traditional dataflow is static
   - doesn’t support CS-like data structures
   - no function calls
   - past dynamic attempts lose (1), (2)
About Dataflow Computing

- Instructions have inputs and outputs
- Instruction “fires” when all inputs available
- Outputs feed into inputs
- High level of (logical) concurrency
  - instructions fire independent of each other
  - natural pipelining
  - self-synchronising (but needs ack cycles)
Dataflow

- Map instructions (DF graph nodes) to compute elements
  - multiple instructions may be on same node
  - only nearest-neighbour communication (with forwarding)
- Tolerates heterogeneous CEs!

Problem: All static
- data structures, algorithms

Solution: graph manipulation instructions

» self-modifying dataflow graph (SMDG)
Self-Modifying Dataflow Graph

“Control structure”

“Data structure” list/stack

©2013 Gernot Heiser NICTA

HotOS'13
Self-Modifying Dataflow Graph

Sends top input to node designated by second

Forwards bottom input

Retargets outgoing edge of top input (ptr) by bottom input (ptr)

Duplicates target node (pointer)

Template

©2013 Gernot Heiser NICTA

HotOS'13
SMDG List Insertion

Template

gate

dup

write edge

send
gate

done

input

d2

list ptr

gate

gate

φ

d1

d0

©2013 Gernot Heiser NICTA
SMDG List Insertion

Template

©2013 Gernot Heiser NICTA
SMDG List Insertion

input

list ptr

gate

dup

send

gate

done

d_{2}

write edge

Template

gate

d_{1}

gate

d_{0}

gate

©2013 Gernot Heiser NICTA
SMDG List Insertion

```
input

list ptr

gate

dup

write edge

send

gate

done

d_2

gate

d_1

gate

d_0

gate

Template

φ

```

©2013 Gernot Heiser NICTA
Challenges

• Mapping graph to CE array dynamically
  – multi-graphing and self-modification requires resource management
  – maybe something like simulated annealing will work?
  – Note: graph nodes can be moved between CEs transparently

• Expressing more conventional algorithms in DF
  – we *think* we can compile Haskell:
    • STG (Haskell intermediate) to simple bytecode translation
    • design for bytecode interpreter written in SMDG assembler
      – with argument and continuation stacks, heap, closures

• Garbage collection for completed computations
  – some vague ideas on how to do this…
Opportunities

• Pointers are capabilities
  – SMDG code is typesafe: clear distinction between pointers and data
  – pointers can be converted to data, possibly resulting in garbage
  – creation of pointers only by duplicate instructions

• Idea: resource management by controlling pointer consumption
Summary

• It’s time to move to a computing paradigm that liberates parallelism
• Self-modifying dataflow graphs have the right properties
  – ... as long as we can solve the challenges

We’re hiring! Systems or formal methods

Boolean, not lawyers’ “or”!