Executing a Program on the MIT Tagged-Token Dataflow Architecture

By: Arvind et al.

Presented By: Blake Hechtman
Let the Data Flow

• Id: high level description of data Flow (Looks Like LISP)
• Compiles to a representation of a Data Flow graph instead of a sequential set of Instructions
• Ideally false sequencing is avoided or minimized
• Programs are Data driven instead of PC driven like in Von Neuman Architectures
Motivation

- Long delays involved in accessing memory in a parallel system
- Loss of determinacy when sequential code is converted to parallel code
- It is difficult for programmers to adequately split tasks
- New language with implicit parallelism
- Functional Language
- Deterministic
- LISP-like Syntax
- Task switching is very explicit
- Programmer agnostic to actual hardware being run
- Operations done on array-like data structures called I-structures

```cpp
Def vsum A B =
  { C = array (1,n) ;
    For j From 1 To n Do
    In
  C ;
```
I-structures

• Initially empty and can only be written once.
• Data is therefore ready when the memory location is no longer empty.
• Reading is automatically deferred until a block is full.
• Non-strict
  – Data structure can be read while other parts are being written.
  – Removes false dependencies and allows for greater concurrency.
Data Flow Graph Compilation

- Operators connected by input and output arcs that correspond to program variables
- Data values are carried on tokens that are said to flow along the arcs with are pointers to the I-structures that contain data
- An operation fires when all tokens are available and then produces tokens on the output arcs
- Operations with constants can be considered one input operators with one output
Functions

• Arcs for each parameter and result
• Invokes trigger the function
• Need to differentiate invocations of a function with tags that allow tokens to be passed correctly (tagging)
• Callee must return to correct context
• Caller must extract tags to send to callee and changes tags to call multiple instances caller also must get_context allocate a function
I-structure Implementation

- Memory module with a controller
  - Handles reads, writes, and storage initialization
  - Present (full), Absent (empty), Waiting (in use)
- When read token arrives
  - Present: Data is sent
  - Absent, Waiting: Request is deferred and queued
- When a write token arrives
  - Absent: Data is written \(\rightarrow\) Present
  - Waiting: Data is written \(\rightarrow\) Present
  - Present \(\rightarrow\) ERROR
Graph Rules for Sanity

• Initially there are no tokens in the graph
• Given exactly one token on input, ultimately exactly one token is produced on every output
• When all outputs have been produced there are no tokens left in the graph
• In addition Loops, conditional statements, high order function add additional layer of complication to Graph creation.
Managers

• Essential a hardware level OS to manage hardware resources at runtime.
• Fills Processing Elements with necessary information to execution of its portion of the dataflow.
• Despite being so low level, it is programmable on an application level
• Allows for mapping of one code to many structures
• Like dynamic P-threads
Connection Architecture

- Symmetric Parallel Processing Elements connected with an n-cube packet network with a global address space
- Processing elements are each dataflow processors
- Dataflow proc+I-struct=dataflow computer
Pipeline

- Wait-Match Unit
- CBRs/DBRs
- Program Memory
- Control
- ALU/Compute Tag
- Form Token
Wait-Match Unit

• Very similar to Wake Up Select in Dynamically scheduled Core
• Memory containing Pool of Ready Tokens
• One input instructions go straight to Ifetch
• Two input operators associatively wait for partner token
• Main simplification is lack of a “Program Order” or control flow
Instruction Fetch

- Instructions are gathered from CBR
- Literals and destination instruction offsets come from the DBR
- This information is gathered in the Program memory and prepares all inputs and outputs for the ALU and Compute Tag Unit
ALU/Compute Tag

- ALU is a normal two operand ALU that uses the opcode to perform an operation.
- The compute Tag is a parallel ALU that calculates instructions off-sets for the results and therefore act like memory and control flow in a Von-Neuman Model.
- Note Control is implemented by CBR offset of instructions.
- These Results are Passed to Form Tokens.
Form Tokens

• Takes data values from the ALU and the tags from the compute tags unit to form tokens
• Essentially acts like commit/writeback
• Tokens are then routed based on destination either to the top of PE, Network, or I-structure.
• Global addresses allow for easy routing
Control

- Control receives Special signals to manipulate PE state
- Used by Manager to initialize/update the CBRs and DBRs as well as store to constant and instruction memory.
- This unit also deals with external world communication
Multiprocessor Mode

- A group of PEs acts very similar to a single PE with very large Address space and a wide pipeline
- Mapped with a “sophisticated” mechanism that spreads work across PEs
- Sharing work is easier because there is no broadcasting only the passing of tokens
- Thankfully no recompiling by configuration
Sounds Simple Right?

• Only need an interconnection network with high bandwidth
• No speculation, Cache Coherence, Consistency, Branch Prediction, Prefetching, register renaming, ROB, MOB or program order
• Provides ideal ILP and a scalable system
• Probable Glueless interconnection of many PEs
Here’s the Catch

• Software is expensive Hardware is cheap, this idea can not go anywhere with out support for not Id languages.
• How would this work for streaming media and commercial Workloads (or any workload)?
• Id seems difficult to use for non-Mathematical Code
• Wait-Match seems highly non-scalable and a probable bottleneck
  – Maybe smaller pools and more nodes would be the best plan (but harder mapping)
Is there hope

- Could different technologies (not CMOS) simplify the implementation of this idea?
- Could an interpreter be written to convert any code into a data flow graph?
- Could this be an add-on to a more standard chip?
- Could hashing actually simplify the associative search of tokens?