Universal Processors

David May

Background

Turing: a Universal Machine can emulate any specialised machine

For Random Access Machines, the emulation overhead is constant

Is there an equivalent Universal Parallel Machine?

A key component is a Universal Network

Idea: A Universal Processor is an infinite network of finite processors

Another Idea: Use a non-blocking network

Universal Parallel Processors

Universal networks emulate specialised networks

Universal processors emulate specialised processors

Networks must have scalable throughput (bisection bandwidth)

Networks must have low latency ($\leq log(p)$) under continuous load

Use network pipelining for continuous (stream) processing: optimal

Use *latency hiding* otherwise: optimal with $log(p)$ excess parallelism

Program Structures

Parallel Random Access Machines (PRAMs)

Data Parallelism

Systolic Arrays

Directed Dataflow Graphs

Task Farms and Server Farms

Recursive Embedding

Communication Patterns

Communication and data access patterns are often known, especially in embedded processing

Processing often involves a series of permutation routing operations between known endpoints

Compilers can allocate processors and network routes

For unknown patterns, use randomisation

For many-to-one, use hashing and combining (or replication)

Emulating Sequential Processors

Distribute data structures across processors

Distribute procedures, functions and objects across processors

Accesses to data are less than 10% of instructions; calls are less than 5%

Network is under-loaded

Optimisations: concurrent accesses and concurrent calls

Composition

Patterns can be composed and embedded within each other

Sometimes the entire program evolution is visible to a compiler

Sometimes the evolution is data-sensitive

The issues in allocating processors and network routes mirror those of allocating memory in sequential processing

... global, stack, heap

How fast can a computation spread?

Non-Blocking Networks

Clos networks implement *permutations* on their inputs

A *strict-sense* network can always allocate a new route

A *re-arrangeable* network needs fewer routers but may require re-arrangement of existing routes

Known permutation + Re-arrangeable = Compile-time (or on-the-fly)

Unknown pattern + Re-arrangeable = Run-time using randomisation

Redundant processors, routers and links can be provided

Benes Networks

For parallel computers, use a *folded* network with two-way links

A network is built from switches with two *edge*-facing links and two *core*-facing links

At each switch, a single bit is needed to route each packet

A network with l layers has 2^l edge facing links and 2^l core facing links

It connects 2^l processors together and provides 2^l *external* links

Route allocation is well understood; there are parallel algorithms

Addressing and Partitions

Processor addresses are in the range 0 ... $2^n - 1$

A partition is of size 2^p and starts at base b: $(b \bmod 2^p) = 0$

Partitions are the unit of processor allocation, like pages for memory

Partitions are used for (logical) synchronisation between processors

Within each partition, *synchronised messages* do not overtake those from a previous permutation

Partitions and Synchronisation

Within each switch, a *synchronised messages* do not overtake

Synchronised messages must be received and forwarded from both inputs before a following synchronised message is forwarded

This ensures an entire partition performs a series of permutations

The entire partition is *in-order pipelined* for synchronised messages

Synchronised messages do not need to carry identifiers

Asynchronous messages can be sent at any time

In-Order Pipelining

In-Order Pipelining = Time-Division Multiplexing (TDM)

For message passing, this allows multiple channels per processor

Communication scheduling + Serial re-use of network

Communication scheduling is a simple extension of network path allocation (similar to adding extra layers)

2-phase TDM + re-arrangeable = strict-sense

Routing

A message route starts with a *depth* that determines how far it progresses towards the core

The next part of the route determines the route towards the core

The final part routes the message to its edge destination

Each switch compares the depth part of each message from the edge with the depth of the switch

When they match, the switch routes the message back towards the edge

Reply addresses

Each switch modifies the route as it passes through

The address of the exit link is deleted; that of the entry link is added

This builds a reply address that can be used to reply to the message

non-blocking outgoing permutation = non-blocking reply permutation

Interconnect Interface

Message structure

SYN	length	route	data
ASYN	length	route	data

where route is (depth, tocore, toedge)

Each processor should have several ports able to buffer one incoming message

This allows concurrent communications to be implemented at each processor by first sending, then receiving

Compiling communications

One-one communication: single permutation

Many-many communication: series of permutations

One-many (broadcast): series of permutations with forwarding via tree from root at source

Many-one (reducing): series of permutations with forwarding via tree to root at destination

All compilable communication patterns transformed to a series of permutations; no network contention

An Example

Switch has 2 edge-facing links and 2 core-facing links.

There are d switch layers connecting 2^d processors

The interconnect contains $d \times 2^{d-1}$ switches

For $d = 16$, there will be 65536 processors and 524288 switches

A route will have a 4-bit depth, 15-bits for core routing and 16-bits for edge-routing

An Alternative Example

Switch has 2^s edge-facing links and 2^s core-facing links.

There are d switch layers connecting 2^{sd} processors

The interconnect contains $d \times 2^{s(d-1)}$ switches

For $s = 4$ and $d = 4$, the routers have 16 edge-facing and 16 core-facing links

There will be 65536 processors and 16384 switches

A route will have a 4-bit depth, 15-bits for core routing and 16-bits for edge-routing

Technology and Packaging

Network on processing chip has 256 links to processors, 256 to network core

Routing chips have 256 links to network edge; 256 to network core

Silicon Photonics would (massively) improve inter-device connections

Synchronisation is only needed between successive router layers

Processors handle input and output on the edge of the system

Network Throughput and Structure

Parallel or serial

Time division multiplexing

Wavelength division multiplexing

... or any combination

Narrow and fast - or Wide and slow?

Processor Node Architecture

Almost any processor architecture can be used

Must be able to source and sink messages concurrently

Conventional interrupt mechanisms can do this

Low latency processor-network interface is useful

Multi-threading is useful

Deterministic execution is useful

Simple architecture enabling on-the-fly compilation is useful

Summary

Scalable, real-time, parallel architecture

Supports all software patterns with low overhead

Software topology independent of physical topology

Homogeneous or Heterogeneous nodes

Redundancy

Problems with ‘Manycores’

Effect of caches in barrier synchronised programs

Example: for 1000 processors, almost all barriers are delayed

For $p = (1-p)^{1000}$, p is about 0.005 (1 in 200)

Contention in packet-routed grids causes similar problems

To overcome this, have to increase granularity and reduce parallelism

Or build specialised processors: problems of heterogeneity