Experiences in Parallel System Development

POETS

Alex Rast
University of Southampton
UKDF 2019
Manchester, UK

Andrew Brown *adb@ecs.soton.ac.uk*
Graeme Bragg *gmb@ecs.soton.ac.uk*
Alex Rast *A.D.Rast@soton.ac.uk*
Mark Vousden *m.vousden@soton.ac.uk*

What is a Parallel System?

Multiprocess

Multiple independent processes (units of execution)

Multi-data

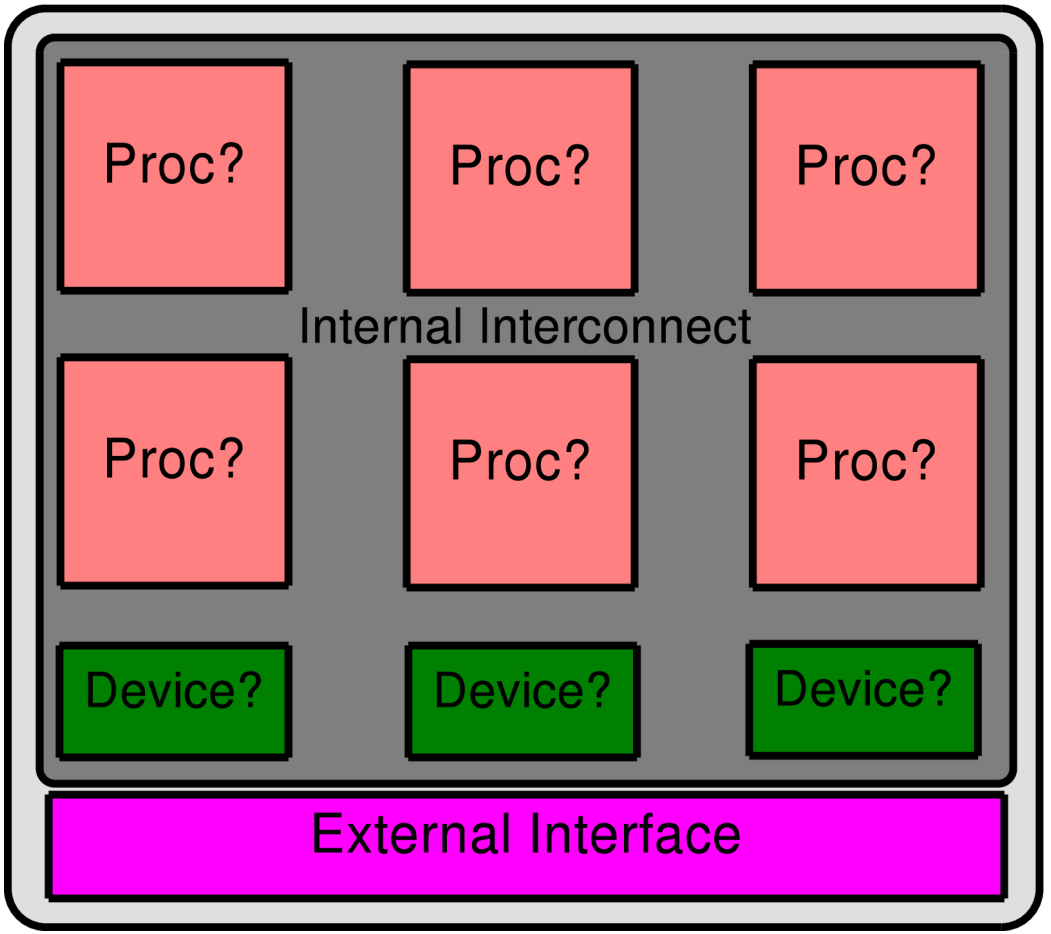
Multiple data streams concurrently in flight

Multicore

Multiple execution engines (hardware units)

Multipath

Multiple independent interconnect links capable of carrying the same data



Parallel Hardware I: Synchronous Multicore

Standard CPUs

Virtually all modern desktop PCs are multicore.

GPUs

Massive but specialised and heterogeneous parallelism.

Specialist Multicore

e.g. Intel Xeon PHI, Adapteva Epiphany. Specs vary.

Synchronous Interconnect

Internal links, whether busses or NOCs, use clocked communication.

Usually Memory Coherent

Typically, chips use shared memory and cache hierarchies.

NVIDIA Volta GV100 Tile – From 'NVIDIA Tesla V100 Architecture'

Parallel Hardware II: (Partially) Asynchronous Multicore

Small Cores

Fairly basic (RISC-V, ARM) RISC cores. Internally synchronous.

Asynchronous NoC

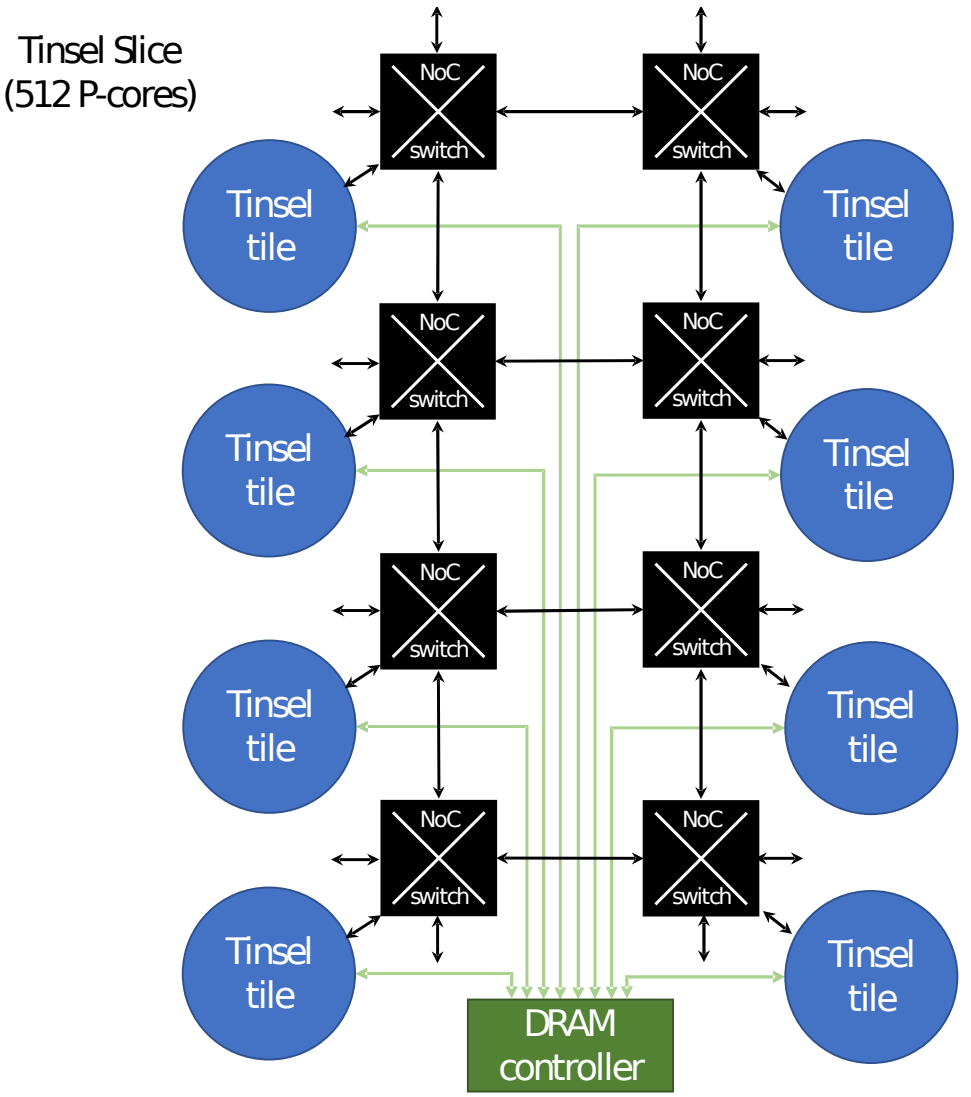
Messages are routed from device to device over an arbitrary path

Event-Driven Control

Processors respond to events, not program sequences

No Memory Coherency

Memory is effectively private. No attempt is made to guard against simultaneous accesses to anything.



Horses for Courses

Conventional Multicore

+: Easy to program, comprehensive tool support, zero learning curve

-: Low parallelism, hard to optimise

GPU

+: Mature tools, high performance in the right application, NVIDIA is taking over the world

-: The problem needs to look like matrix-bashing, memory accesses must be carefully synchronised, power-hungry

Specialist Multicore

+: Tunable for optimal performance, relatively problem-agnostic

-: Steep learning curve, partial support, hardware obsolescence

Asynchronous Multicore

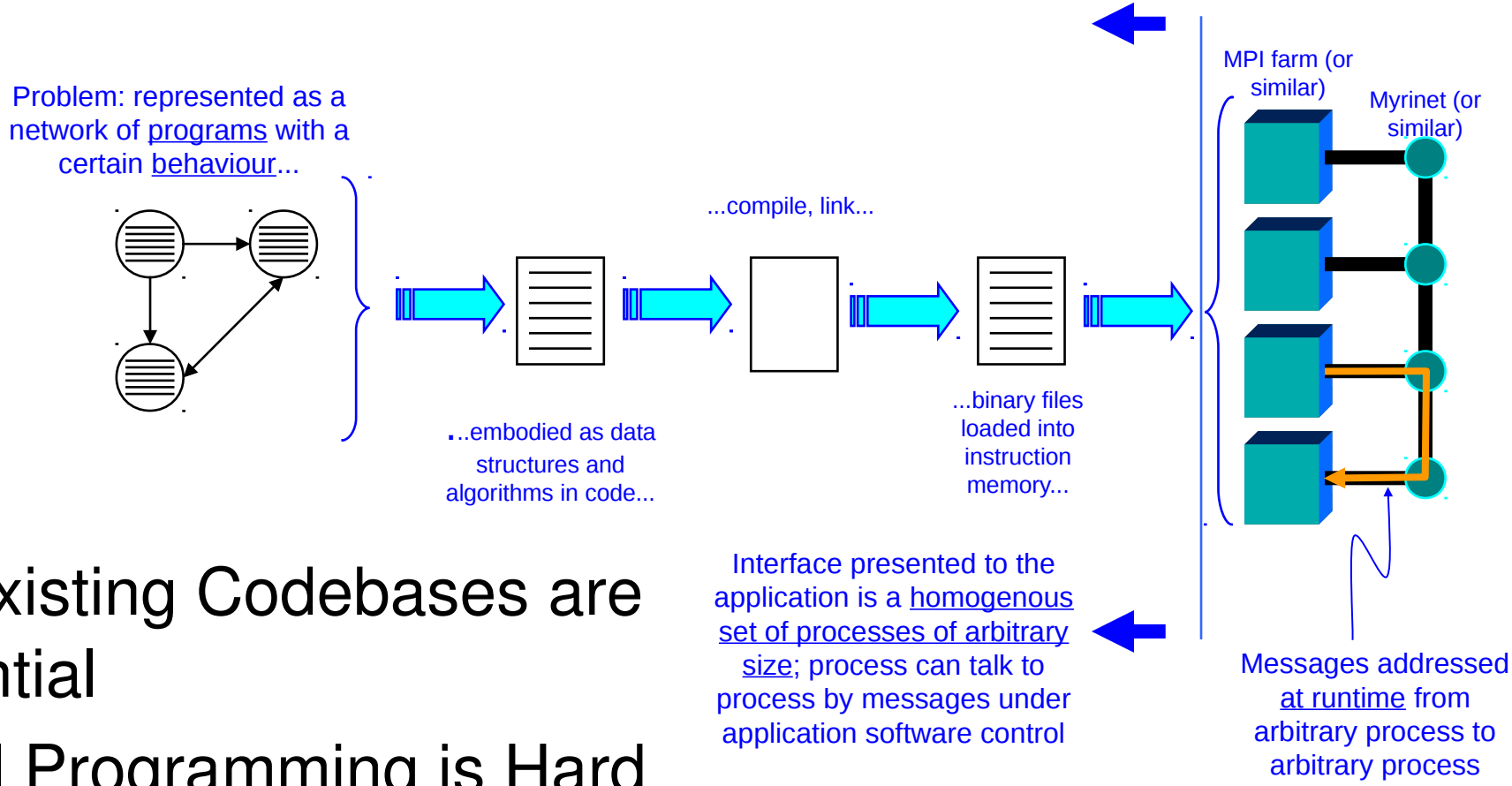
+: Low power, intuitive application model, no need to worry about synchronisation or coherency

-: Immature tools, debugging is hard, research technology, implementation pitfalls

...`*shape'* of the application matters

But How to Program These Things?

A conventional multi-processor program:



- Most Existing Codebases are Sequential
- Parallel Programming is Hard
- Is Language the Solution?
- You Still Have to Map it to the Hardware

Parallel Software I: Of Processes and Threads

MPI

A heavy-duty standard for message-passing

Processes and Comms

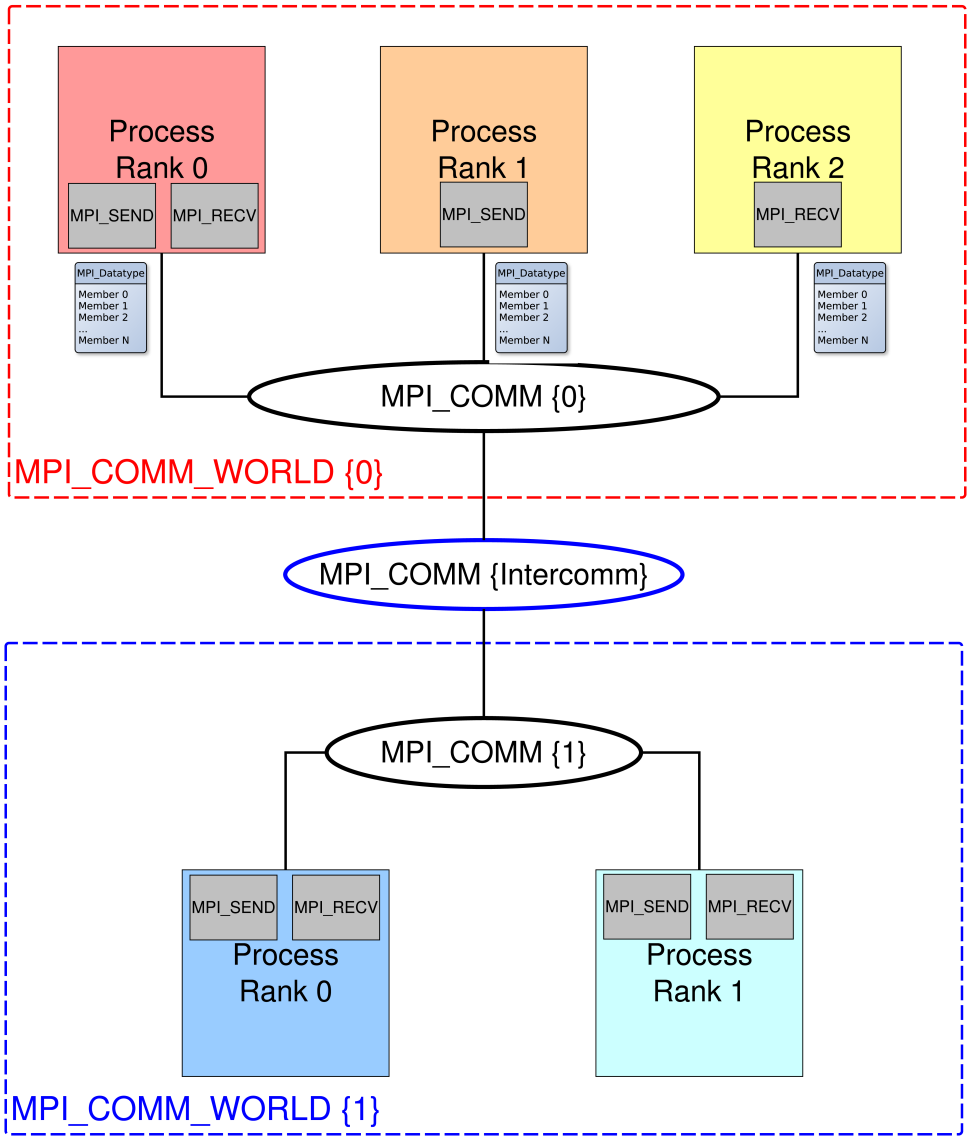
Physical ↔ Logical resource mapping.
Thread support available.
Multicomm support is spotty.

Message Send/Receive

Relies on highly choreographed communications.
May be blocking or nonblocking.

Diverging Implementations

MPICH, OpenMPI, MS-MPI, and others, each with different support and little cross-platform interoperability



Parallel Software II: Of Graphs and Messages

Processing Element = Node

Performs some *simple* function

Communications Link = Edge

Represents an abstract message path between nodes

Event = Message

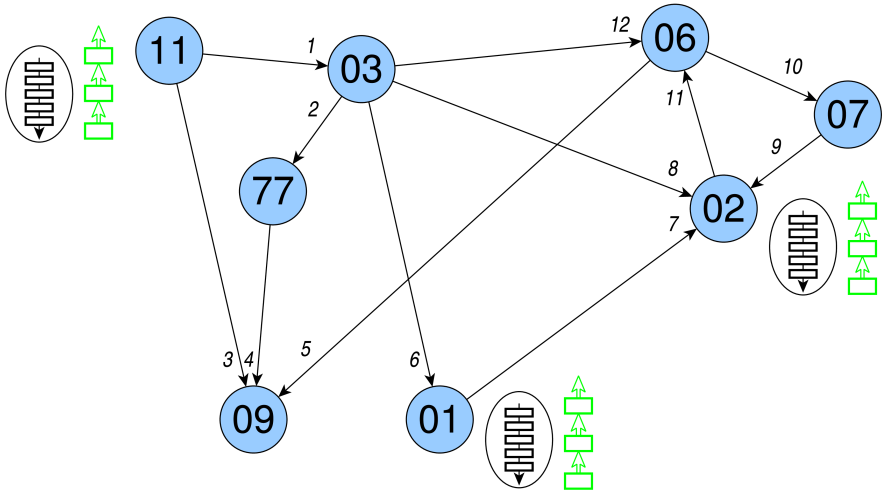
Sends a small data item and a notification

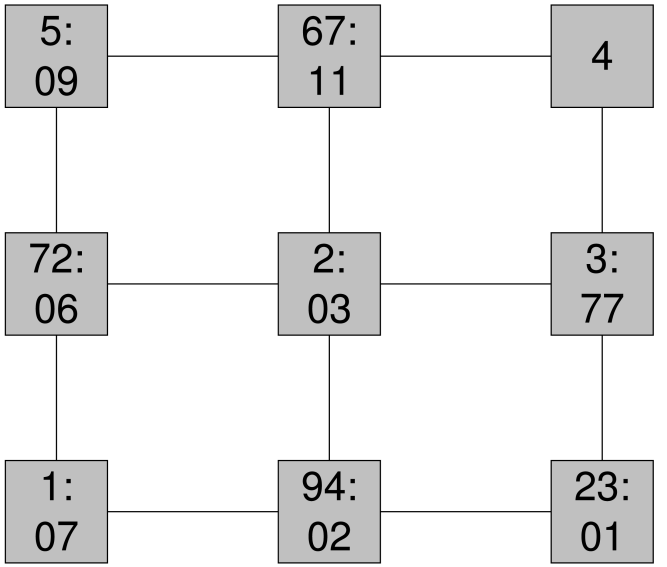
User Specifies the Application Graph

Describes the *program* topology

System Software Maps to a Hardware Graph

Application ←→ Hardware ID mappings





An Old Problem: Clock Trees

Parallel

Models fanout of clock buffers; in large trees there can be ~10^7 devices

Extremely Simple

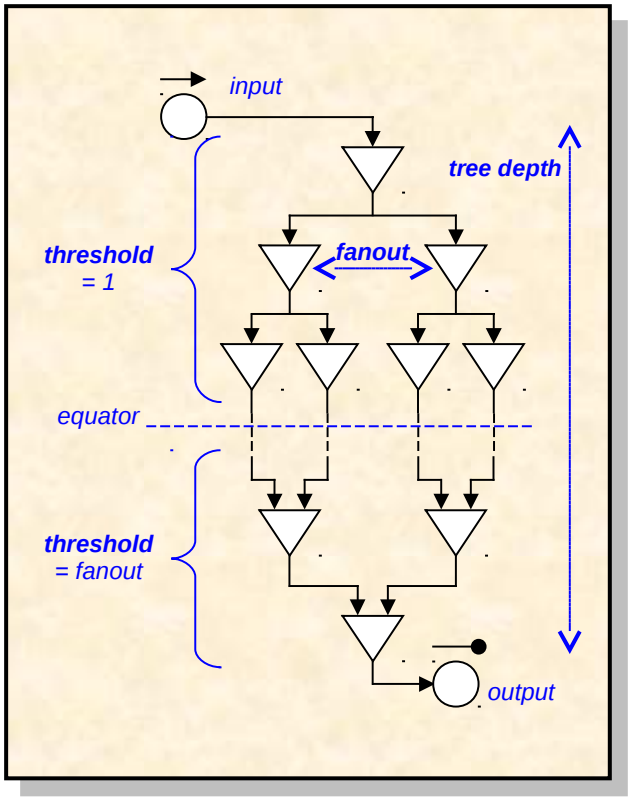
Each device simply records and forwards messages; each message is nothing more than a point event

Synchronous or Asynchronous

Data flow does not depend on explicit synchronisation and is free from side effects

Worst-case Modelling

The layer at the equator generates intense packet storms



Clock Trees on SpiNNaker

(1, 1)	1	(2, 1)	1	(3, 1)	1	(4, 1)	1	(5, 1)	1	(6, 1)	1	(7, 1)	1	(8, 1)	1	(9, 1)	1	(10, 1)	1	(12, 1)	1	(14, 1)	1	(16, 1)	1
3																									
(1, 2)	1	(2, 2)	1	(3, 2)	1	(4, 2)	1	(5, 2)	1	(6, 2)	1	(7, 2)	1	(8, 2)	1	(9, 2)	1	(10, 2)	1	(12, 2)	1	(14, 2)	1	(16, 2)	1
5	0.1	10	0.1	17	0.1	26	0.1	37	0.1	50	0.1	65	0.1	82	0.1	101	0.1	122	0.1	170	0.1	226	0.1	290	0.1
(1, 3)	1	(2, 3)	1	(3, 3)	1	(4, 3)	1	(5, 3)	1	(6, 3)	1	(7, 3)	1	(8, 3)	2	(9, 3)	2	(10, 3)	3	(12, 3)	4	(14, 3)	7	(16, 3)	10
7	0.1	22	0.1	53	0.1	106	0.1	187	(n/a)	302	0.1	457	0.1	658	0.1	911	0.1	1222	0.2	2042	1	3166	2	4642	5
(1, 4)	1	(2, 4)	1	(3, 4)	1	(4, 4)	1	(5, 4)	2	(6, 4)	4	(7, 4)	7	(8, 4)	11	(9, 4)	17	(10, 4)	24	(12, 4)	48	(14, 4)	87	(16, 4)	146
9	0.1	46	0.1	161	0.1	426	0.1	937	0.1	1814	0.5	3201	1	5266	2.5	8201	10	12222	(n/a)	24506	(n/a)	44326	(n/a)	74274	(n/a)
(1, 5)	1	(2, 5)	1	(3, 5)	1	(4, 5)	4	(5, 5)	10	(6, 5)	22	(7, 5)	44	(8, 5)	83	(9, 5)	145	(10, 5)	239	(12, 5)	575	(14, 5)	1213	(16, 5)	2322
11	0.1	94	0.1	485	0.1	1706	0.2	4687	1.5	10886	10	22409	(n/a)	42130	(n/a)	73811	(n/a)	122222	(n/a)	294074	(n/a)	620566	(n/a)	1188386	(n/a)
(1, 6)	1	(2, 6)	1	(3, 6)	3	(4, 6)	14	(5, 6)	46	(6, 6)	128	(7, 6)	307	(8, 6)	659	(9, 6)	1298	(10, 6)	2388	(12, 6)	6893				
13	0.1	190	0.1	1457	0.1	6826	2	23437	(n/a)	65318	(n/a)	156865	(n/a)	337042	(n/a)	664301	(n/a)	1222222	(n/a)	3528890	(n/a)				
(1, 7)	1	(2, 7)	1	(3, 7)	9	(4, 7)	54	(5, 7)	229	(6, 7)	766	(7, 7)	2145	(8, 7)	5267										
15	0.1	382	0.1	4373	1	27306	(n/a)	117187	(n/a)	391910	(n/a)	1098057	(n/a)	2696338	(n/a)										
(1, 8)	1	(2, 8)	2	(3, 8)	26	(4, 8)	214	(5, 8)	1145	(6, 8)	4593														
17	0.1	766	0.1	13121	10	109226	(n/a)	585937	(n/a)	2351462	(n/a)														
(1, 9)	1	(2, 9)	3	(3, 9)	77	(4, 9)	854																		
19	0.1	1534	0.1	39365	(n/a)	436906	(n/a)																		
(1, 10)	1	(2, 10)	6	(3, 10)	231	(4, 10)	3414																		
21	0.1	3070	0.2	118097	(n/a)	1747626	(n/a)																		
(1, 12)	1	(2, 12)	24	(3, 12)	2076																				
25	0.1	12286	(n/a)	1062881	(n/a)																				
(1, 13)	1	(2, 13)	48																						
27	0.1	24574	15																						
(1, 14)	1	(2, 14)	96																						
29	0.1	49150	(n/a)																						
(1, 16)	1	(2, 16)	384																						
33	0.1	196606	(n/a)																						
(1, 18)	1	(2, 18)	1536																						
37	0.1	786430	(n/a)																						
(1, 20)	1																								
41	0.1																								
(1, 22)	1																								
45	0.2																								
(1, 24)	1																								
49	0.1																								
(1, 26)	1																								
53	0.1																								
(1, 28)	1																								
57	0.1																								
(1, 30)	1																								
61	0.1																								
(1, 32)	1																								
65	0.1																								

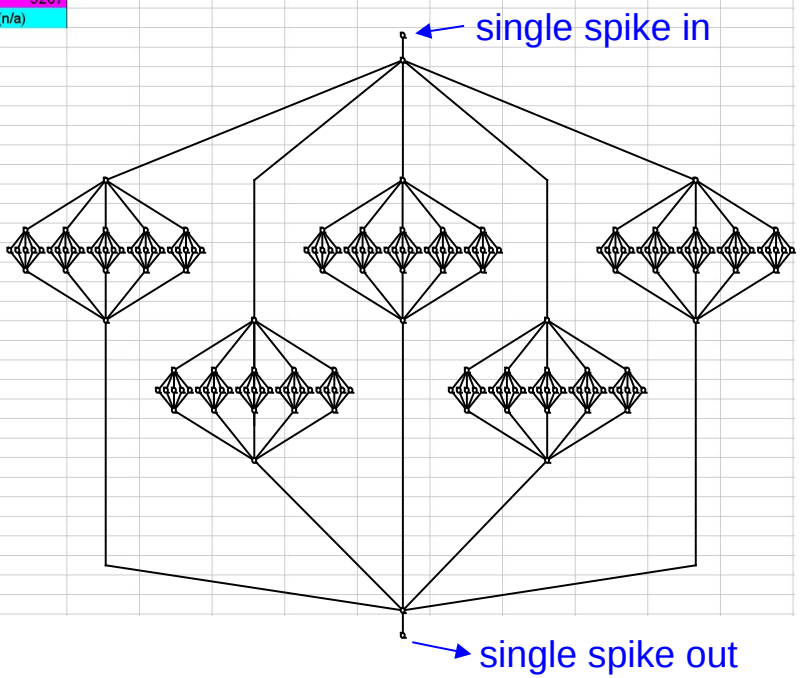
(width,depth)	dev/core
∑devices	min timer

data lost

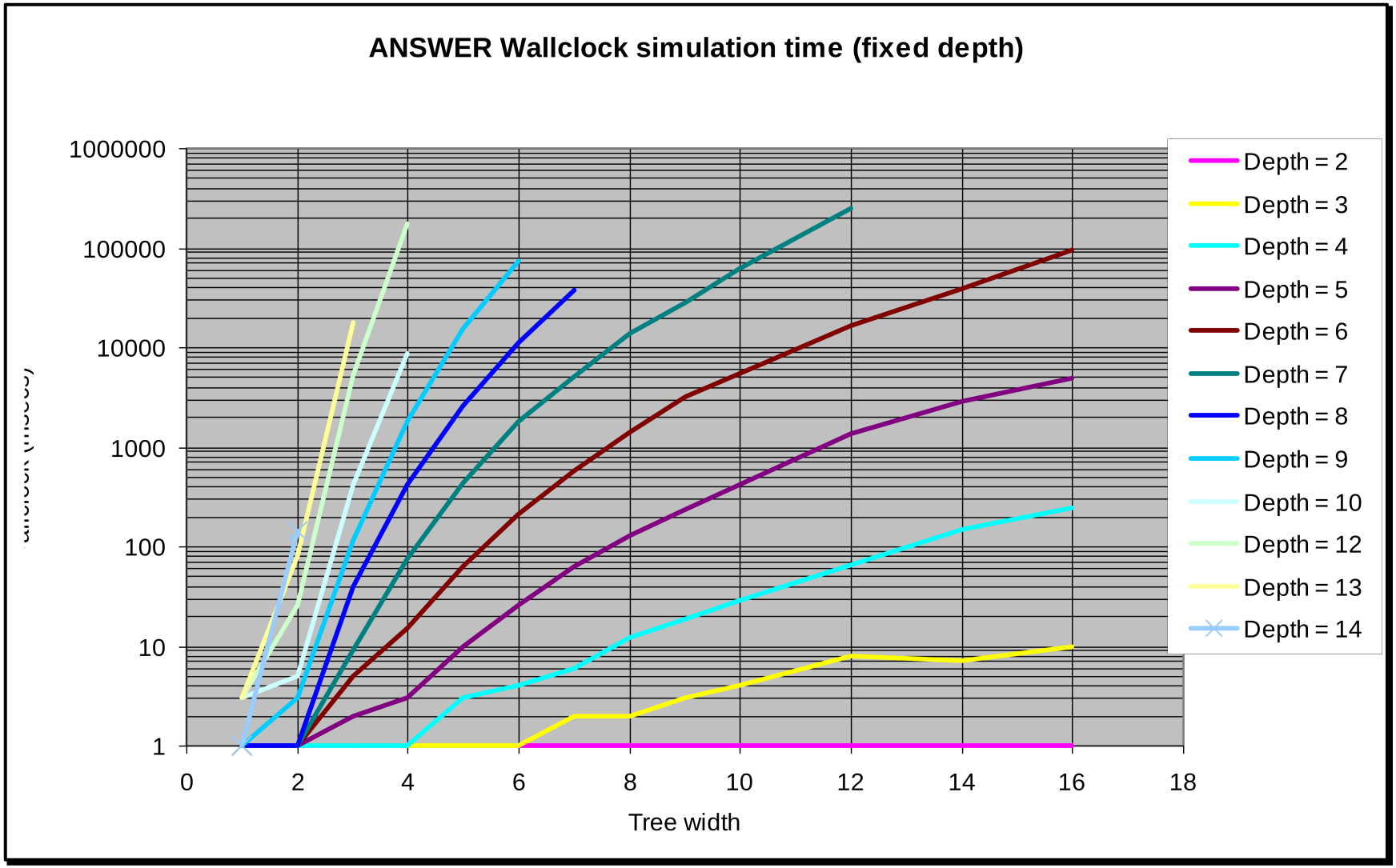
MC table overflow

>1000 dev/core

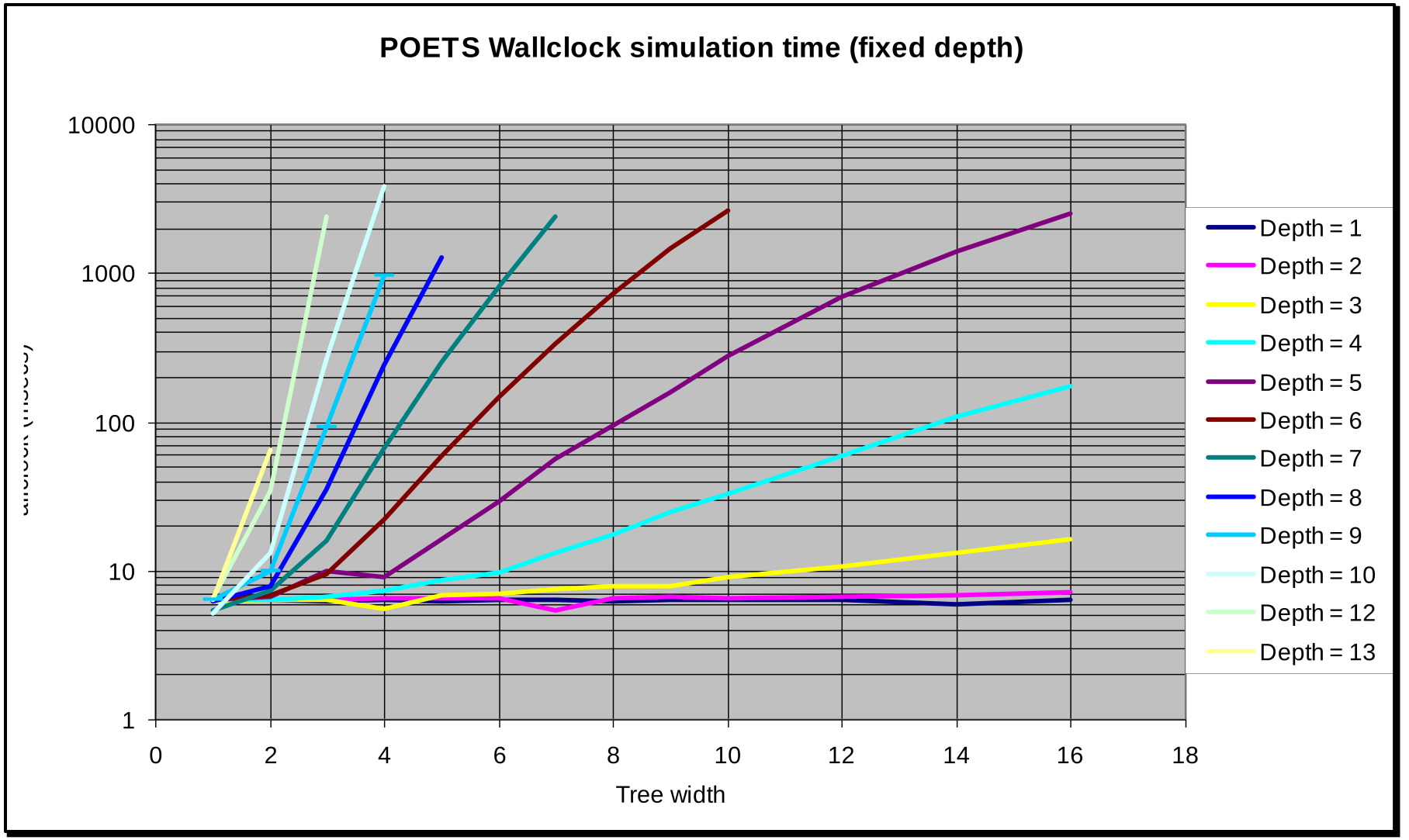
no biological fidelity



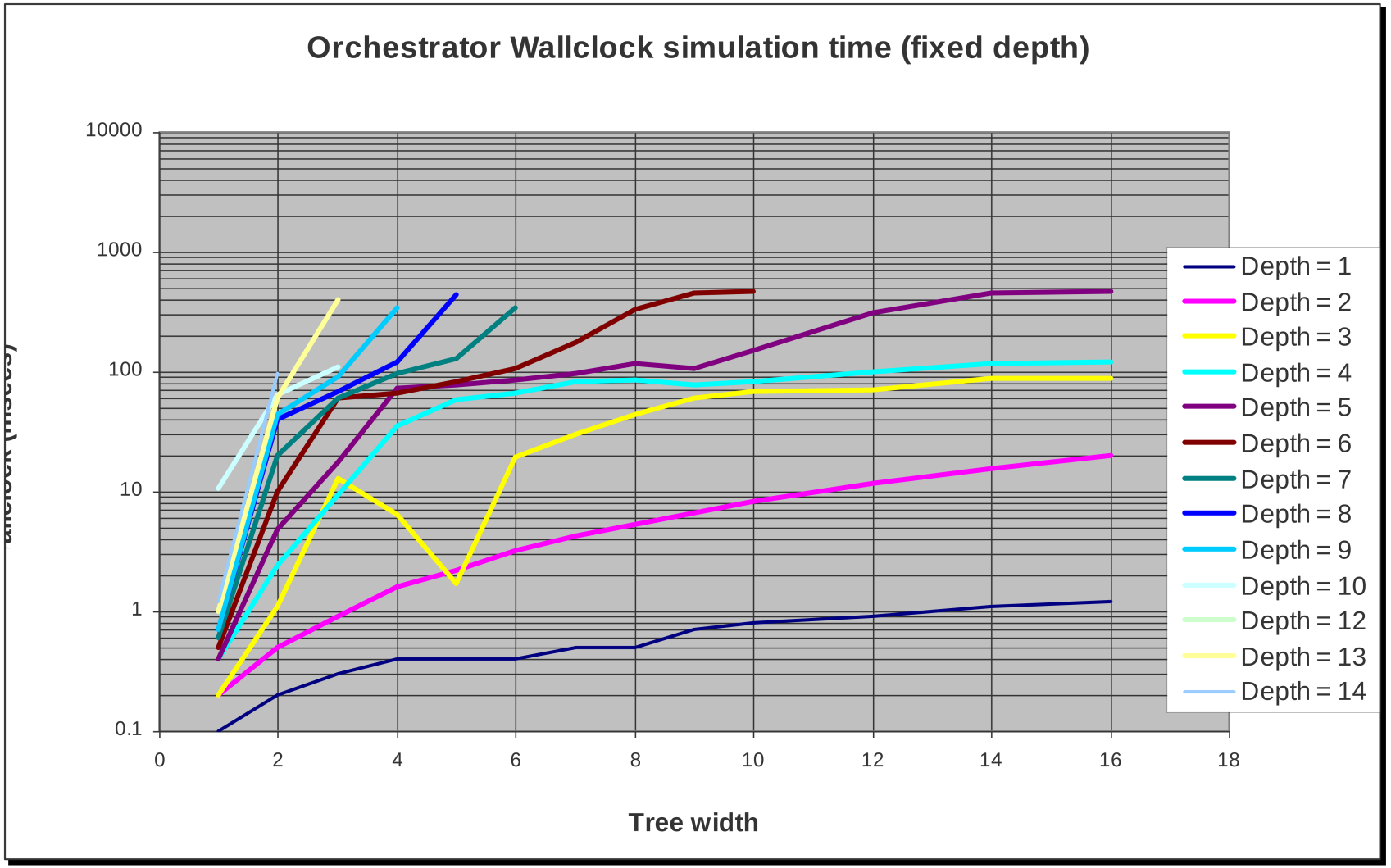
Clock Trees on a Conventional PC



Clock Trees on POETS

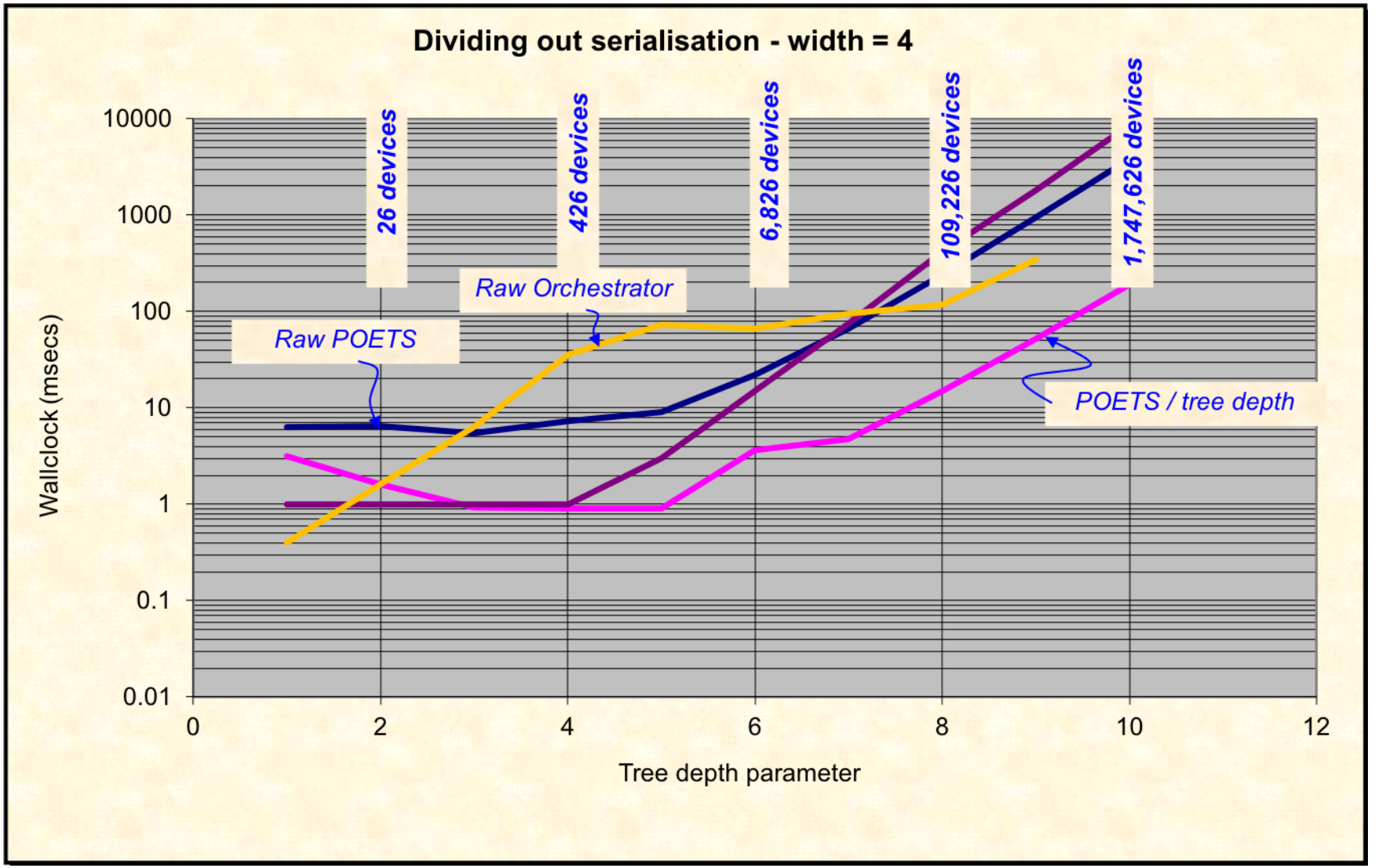


Clock Trees on POETS (Again!)



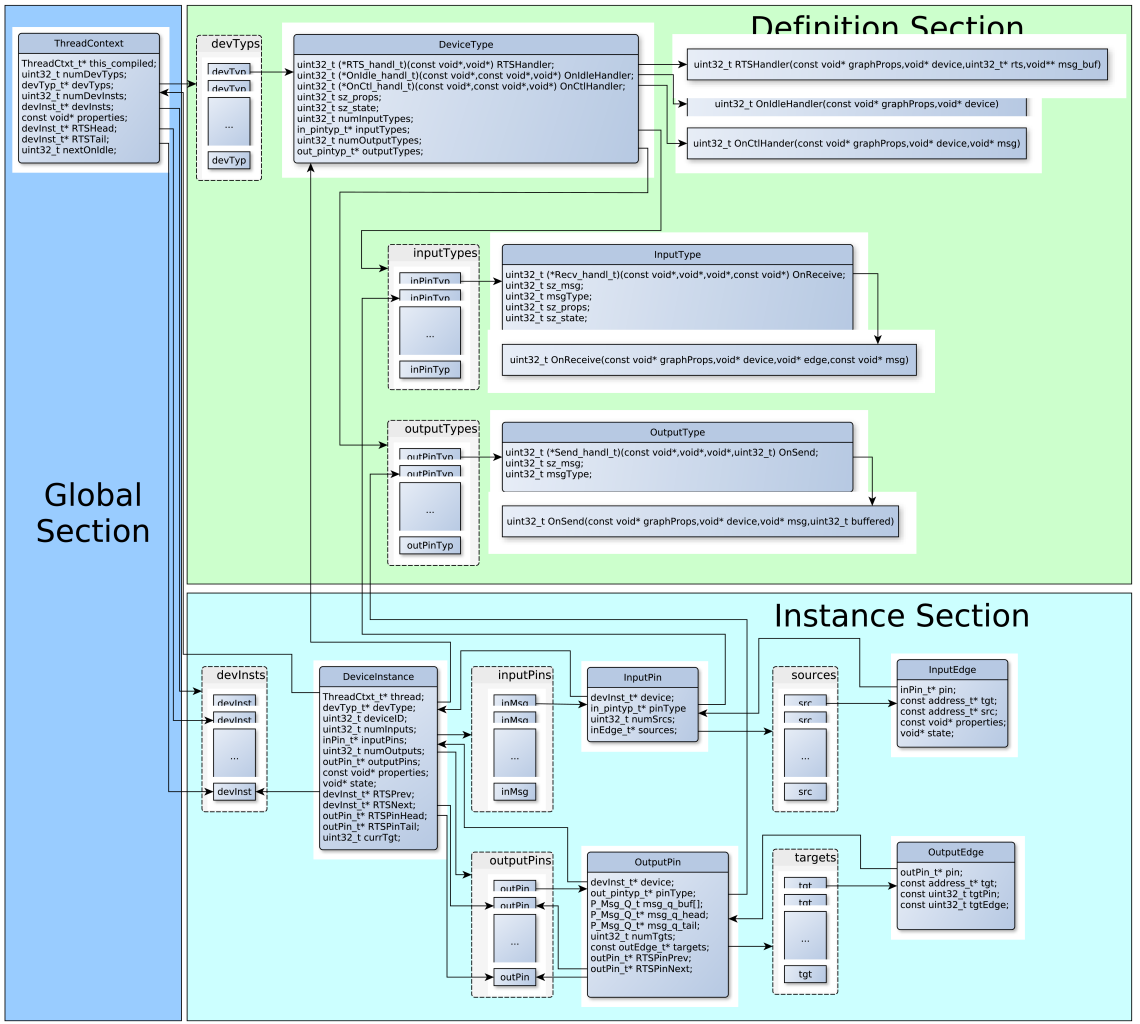
So What's Easy (and Works)?

- Application Specification
- Scaling (up to a point!)
- Code Generation
- Basic Mapping



And What's Hard?

- Debugging
- Recasting the Problem
- Time to Generate and Load
- Data Type Support
- Nontrivial Mapping
- Sensible I/O



What's Next?

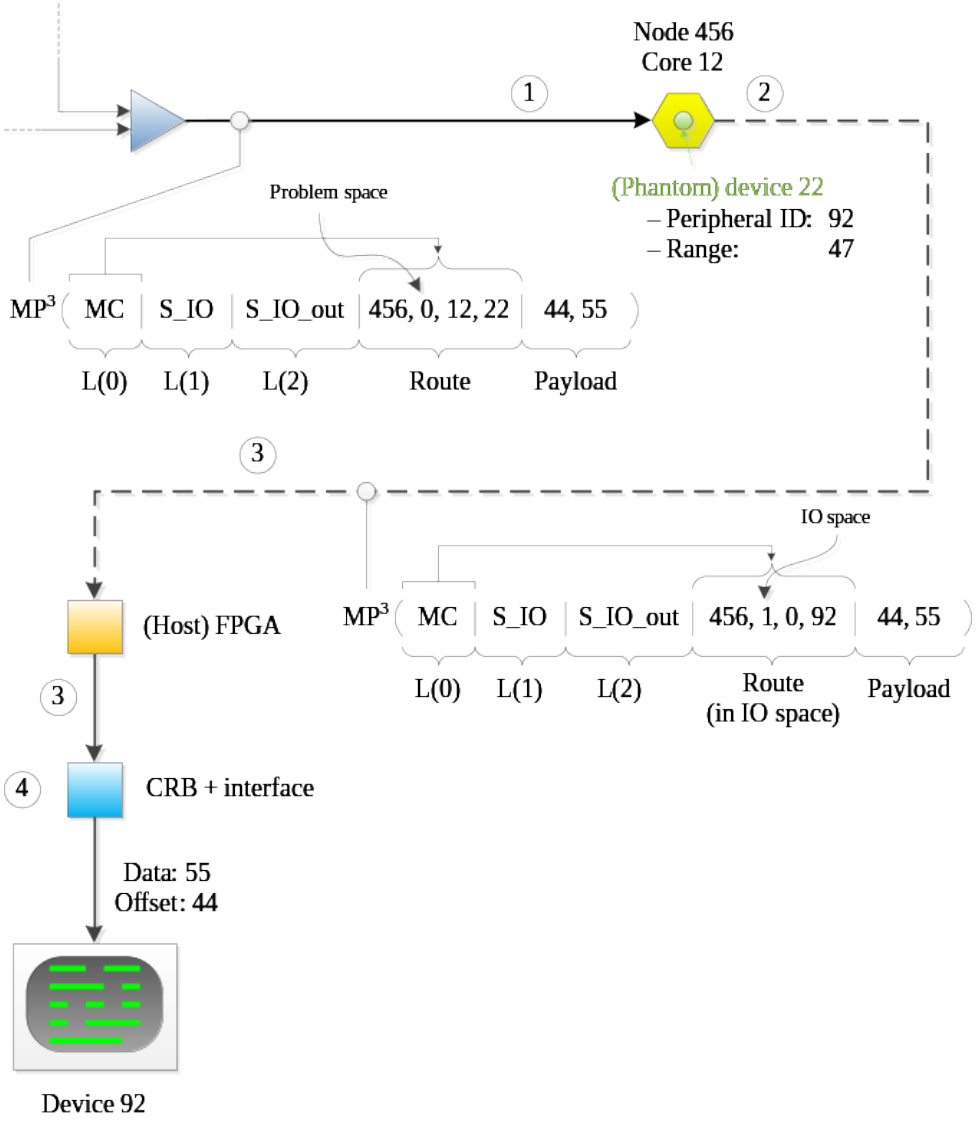
Mapping Algorithms (Simulated Annealing?)

Standards for Asynchronous Message I/O

Work on Formal Verification?

Diversifying the Applications

Better Hardware



And What's The Future?

NVIDIA Isn't Going Away

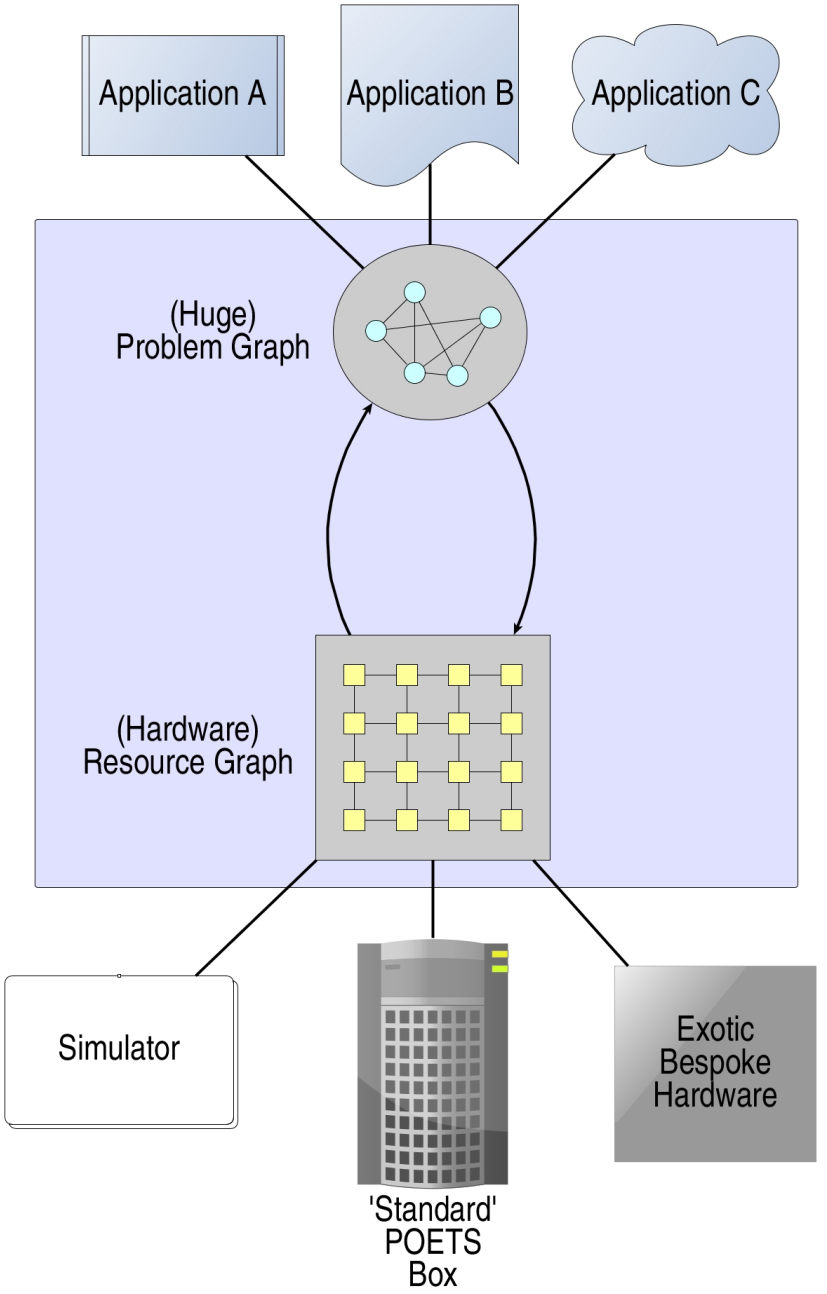
But Some Other Models
Will Find Niches

Graph Descriptions and
Graphical Programming Software

Automation of the
Mapping Process

Asynchronous = True Parallel?

Embedded Parallel Systems



Many Thanks To...