A look ahead: Echelon

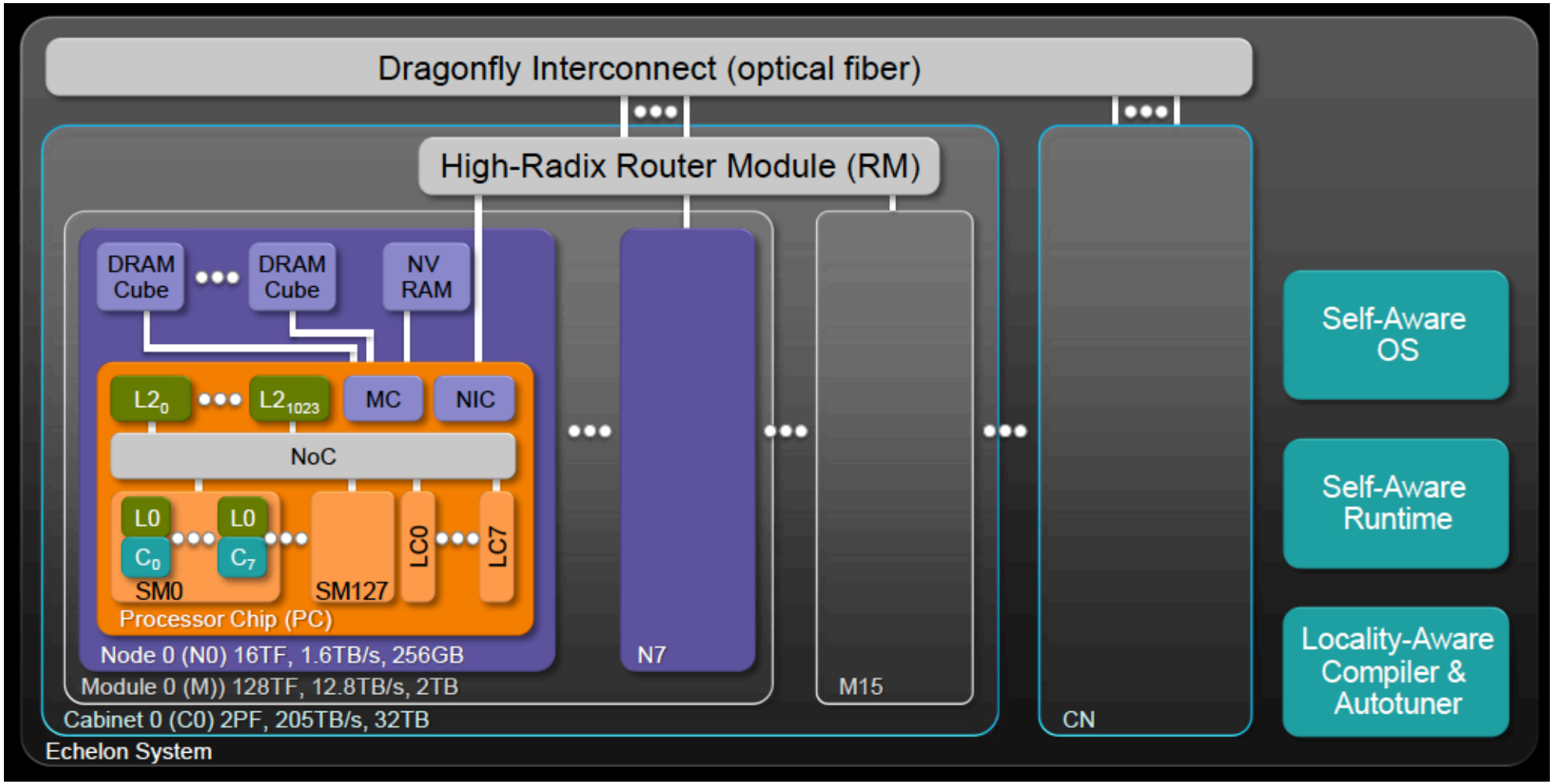
Talk contents [13 slides]

1. The Echelon system [4].
2. The challenge of power consumption in Echelon [9].

I. Introduction

Companies involved in the project

System sketch



Thread count estimation

	2010: 4640 GPUs (32*145 Fermi GPUs)	**2018: 90K GPUs** (based on a Echelon system)
Threads/SM	1 536	~ 1 000
Threads/GPU	(16x) 24 576	~ 100 000
Threads/Cabinet	(32x) 786 432	~ 10 000 000
Threads/Machine	(145x) ~ 100 000 000	~ 10 000 000 000

How to get to billions of threads

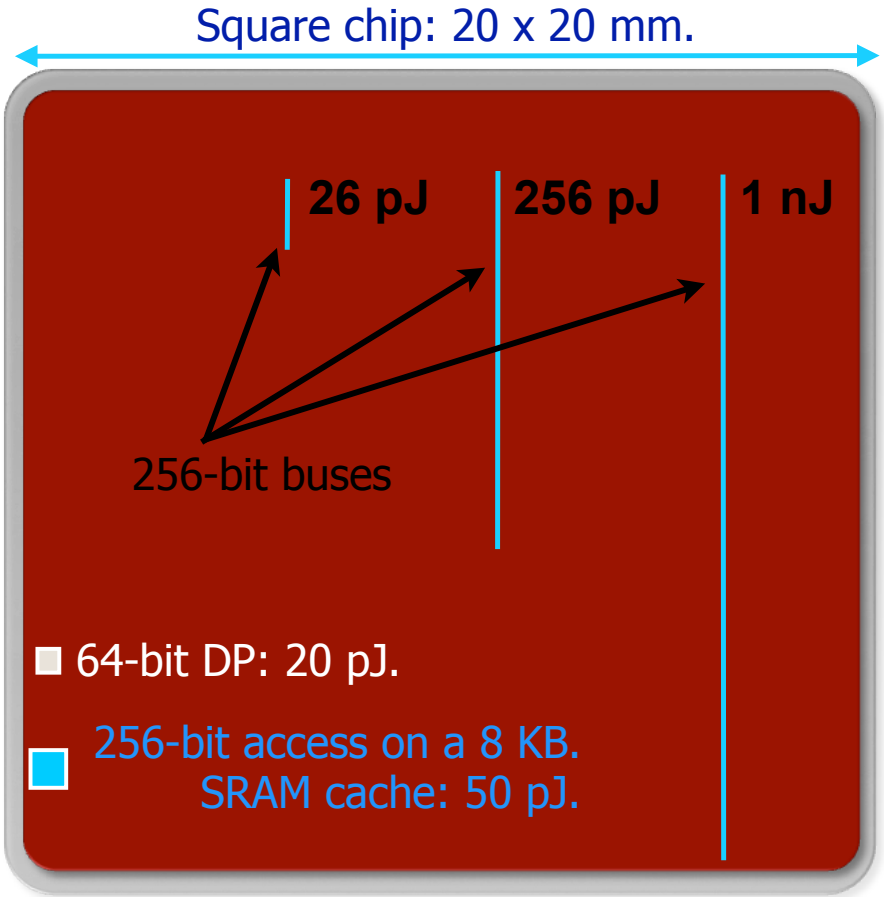
- Programming System:
  - Programmer expresses **all** of the concurrency.
  - Programming system decides how much to deploy in space and how much to iterate in time.
- Architecture:
  - Fast, low overhead thread array creation and management.
  - Fast, low overhead communication and synchronization.
  - Message-driven computing (active messages).

II. The challenge of power consumption in Echelon

Power consumption in typical computers

The high cost of data movement: Fetching operands costs more than computing on them

- As of 2012 chips in silicon, we have:



Manufacturing process: 28 nm.

Efficient off-chip link:

500 pJ

DRAM read/write:

16 nJ (16.000 pJ)

Addressing the power challenge

- Locality and its role on power consumption:
  - Bulk of data must be accessed from register file (2 pJ.), not across the chip (integrated cache, 150 pJ.), off-chip (external cache, 300 pJ.), or across the system (DRAM memory, 1000 pJ.).
  - Application, programming system and architecture must work together to exploit locality.
- Overhead:
  - Bulk of execution energy must go to carrying out the operation, not scheduling instructions (where 100x is consumed today).
- Optimizations:
  - At all levels of the memory hierarchy to operate efficiently.

Power consumption within a GPU

Manufacturing process (and year):	40 nm. ('10)	10 nm. (estim. 2017)	
User platform:	**Desktop**	**Desktop**	**Laptop**
Vdd (nominal)	0.9 V.	0.75 V.	0.65 V.
Target frequency	1.6 GHz.	2.5 GHz.	2 GHz.
Energy for a madd in double-precision	50 pJ.	8.7 pJ.	6.5 pJ.
Energy for a add with integer data	0.5 pJ.	0.07 pJ.	0.05 pJ.
64-bit read from 8 KB. SRAM	14 pJ.	2.4 pJ.	1.8 pJ.
Wire energy (per transition)	240 fJ/bit/mm	150 fJ/bit/mm	115 fJ/bit/mm
Wire energy (256 bits, distance of 10 mm.)	310 pJ.	200 pJ.	150 pJ.

- Communications take the bulk of power consumption.
- And instruction scheduling in an out-of-order CPU is even worse, spending 2000 pJ. for each instruction (either integer o floating-point).

Scaling makes locality even more important: Power consumption within VRAM

Manufacturing process (and year):	45 nm. (2010)	16 nm. (estimated for 2017)
DRAM interface pin bandwidth	4 Gbps.	50 Gbps.
DRAM interface energy	20-30 pJ/bit	2 pJ/bit
DRAM access energy	8-15 pJ/bit	2.5 pJ/bit

Projections for power consumption in CPUs and GPUs (in picoJules)

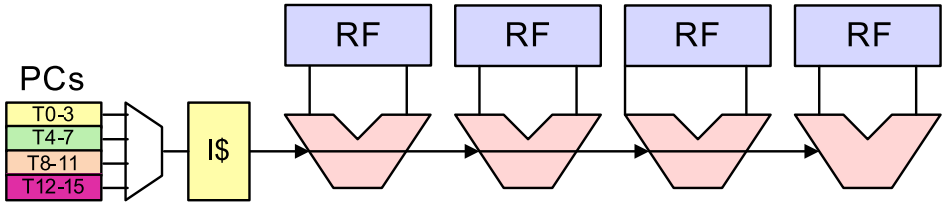
	CPU in 2010	CPU in 2015	GPU in 2015	Echelon's goal by maximizing locality
Instruction scheduling	2000	560	3	3
Access to on-chip cache	75	37,5	37,5	10,5
Access to off-chip cache	100	15	15	9
Arithmetic operation (average cost)	25	3	3	3
Local access to register file	14	2,1	2,1	2,7
TOTAL	2214	617,6	60,6	28,2

Basic power guidelines at different levels

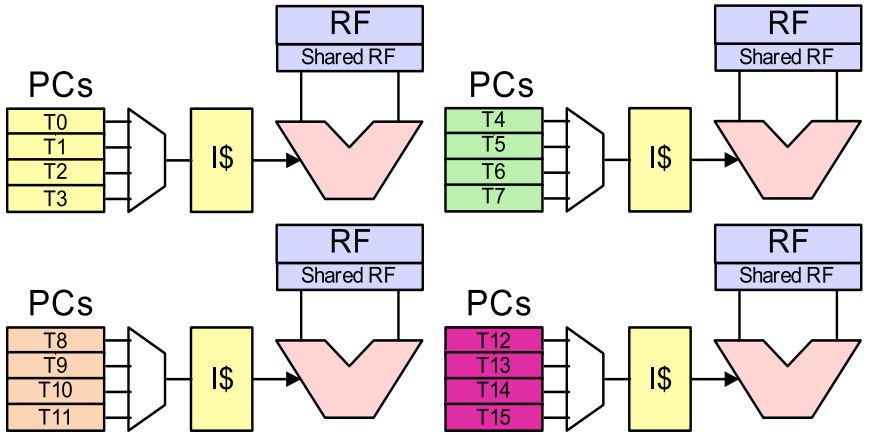
- The bulk of the power is consumed by data movement rather than operations. Therefore, algorithms should be designed to perform more work per unit data movement:
  - Performing more operations as long as they save transfers.
  - Recomputing values instead of fetching them.
- Programming systems should further optimize this data movement:
  - Using techniques such as blocking and tiling.
  - Being aware of the energy cost for each instruction.
- Architectures should provide:
  - A memory hierarchy exposed to the programmer.
  - Efficient mechanisms for communication.

A basic idea to optimize power consumption in GPUs: Temporal SIMT

- Existing SIMT (Single Instruction Multiple Thread) amortizes instruction fetch across multiple threads, but:
  - Perform poorly (and energy inefficiently) when threads diverge.
  - Execute redundant instructions that are common across threads.



- Solution: Temporal SIMT.
  - Execute threads in thread block in sequence on a single lane, which amortizes fetch.
  - Shared registers for common values, which amortizes execution.



Power consumption on Nvidia's roadmap