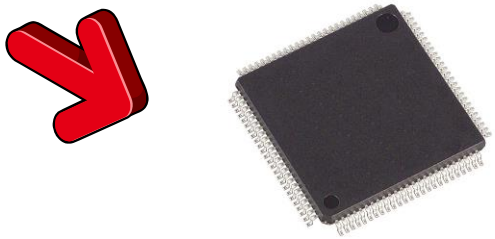


Some examples of many-cores-on-chip architectures

Ran Ginosar

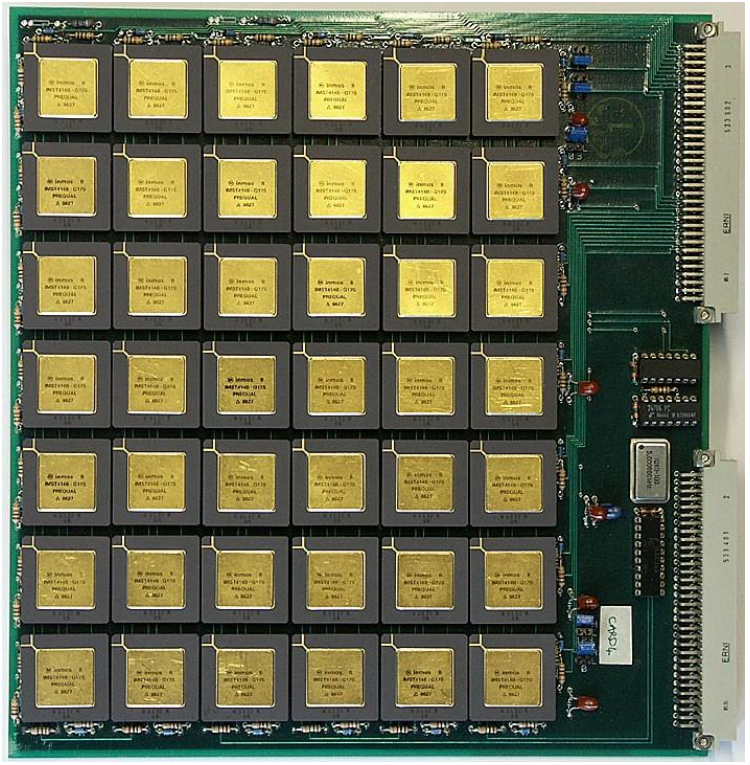
Technion

Many many-core contenders

- Ambric
- Aspex Semiconductor
- ATI GPGPU
- BrightScale
- Calxeda
- ClearSpeed Technologies
- Coherent Logix
- CPU Technology
- Element CXI
- Elixent/Panasonic
- IBM Cell
- IMEC
- Intel Larrabee, SCC, …
- Intellasys
- IP Flex
- KalRay MPPA
- MathStar
- Mellanox BlueField
- Motorola Labs
- NEC
- Nvidia GPGPU
- PACT XPP
- Picochip
- Plurality
- Rapport Inc.
- Recore
- Silicon Hive
- Stream Processors Inc.
- Tabula
- Tilera

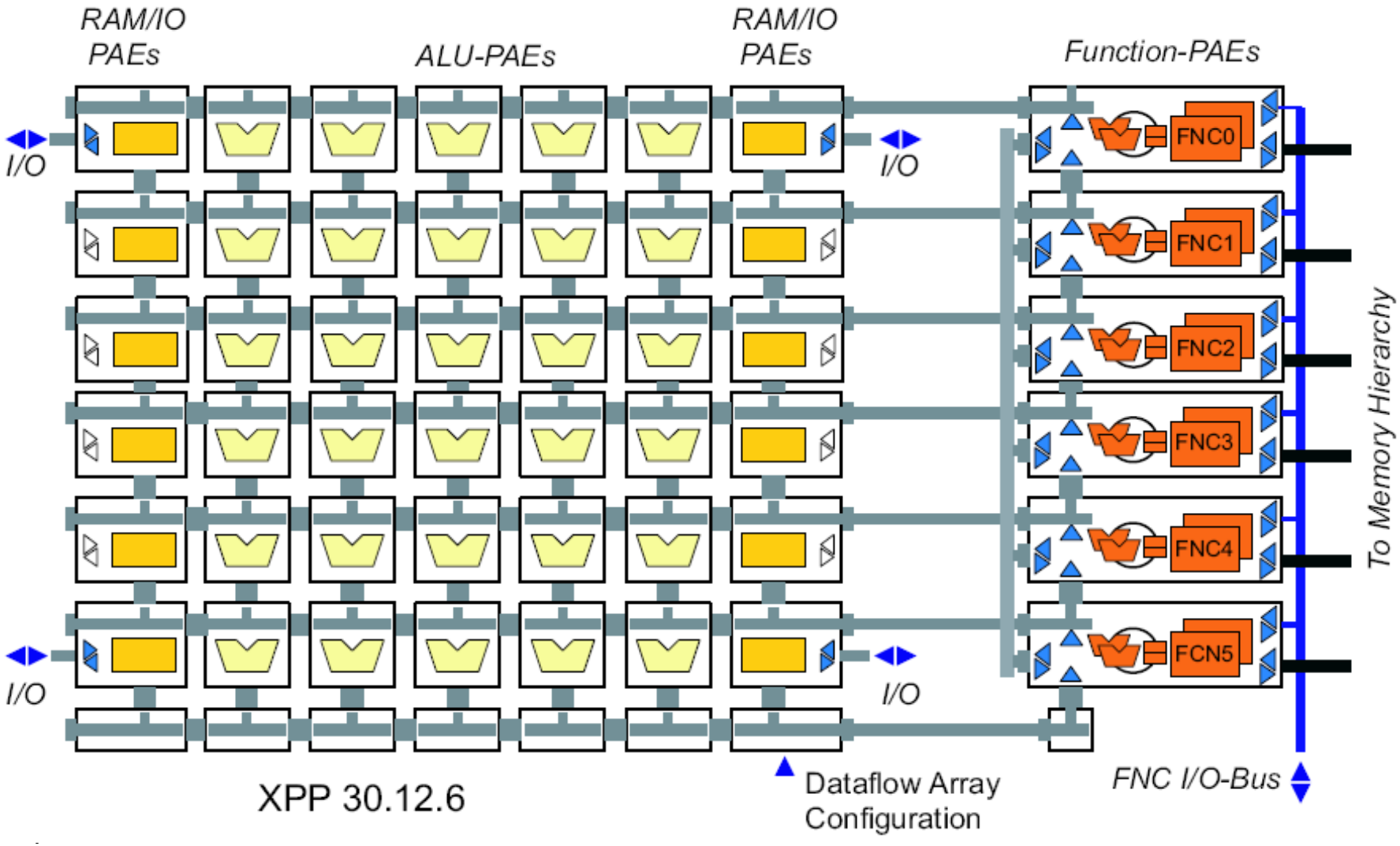
PACT XPP

- German company, since 1999
  - Martin Vorbach,
    an ex-user of Transputers

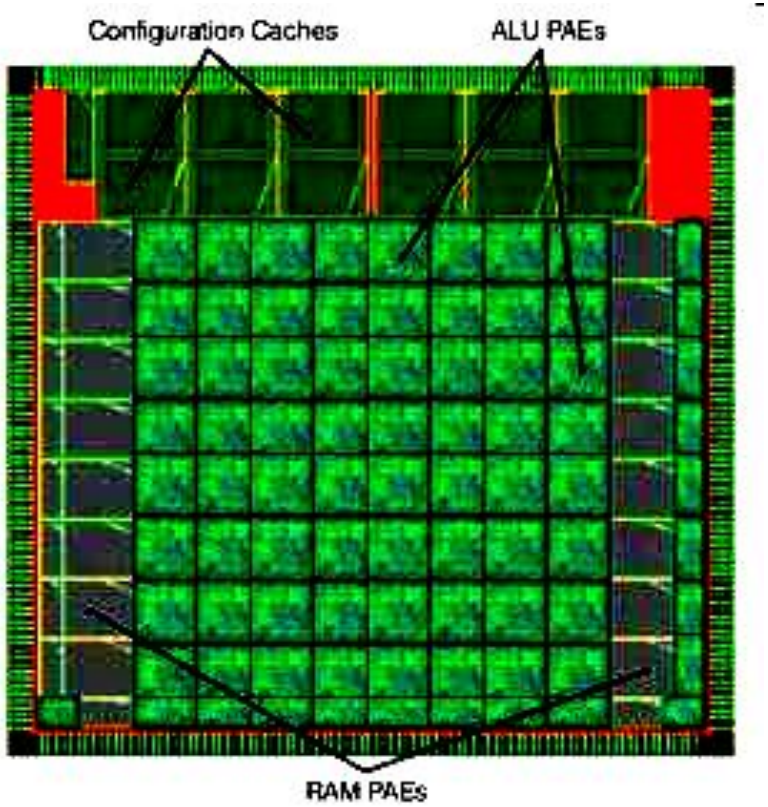


42x Transputers mesh 1980s

PACT XPP (96 elements)



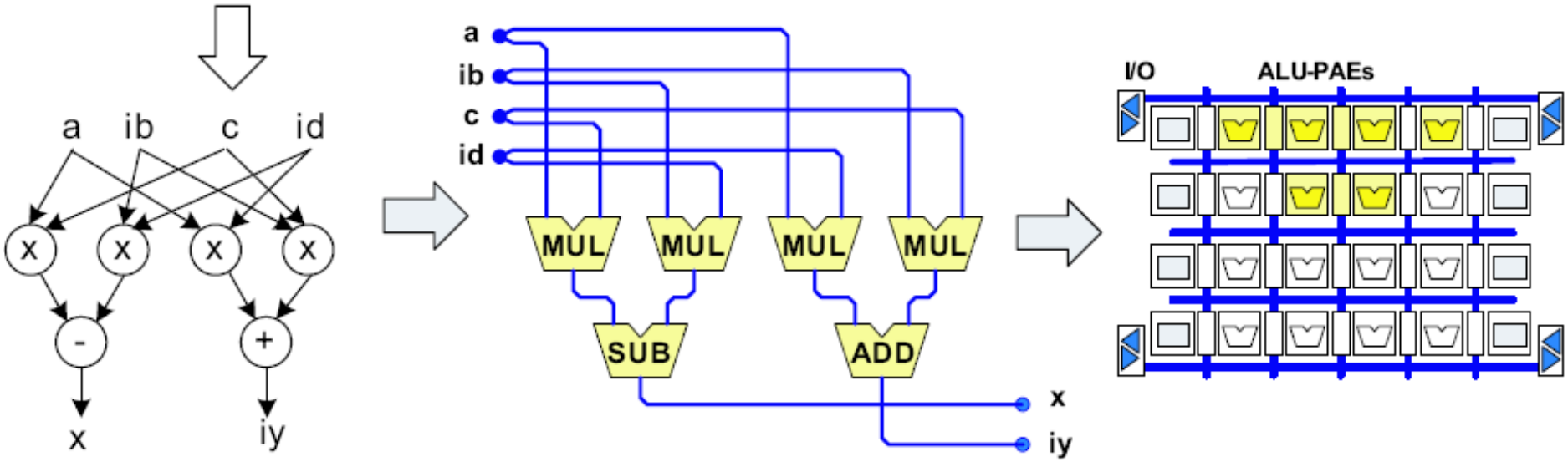
PACT XPP die photo



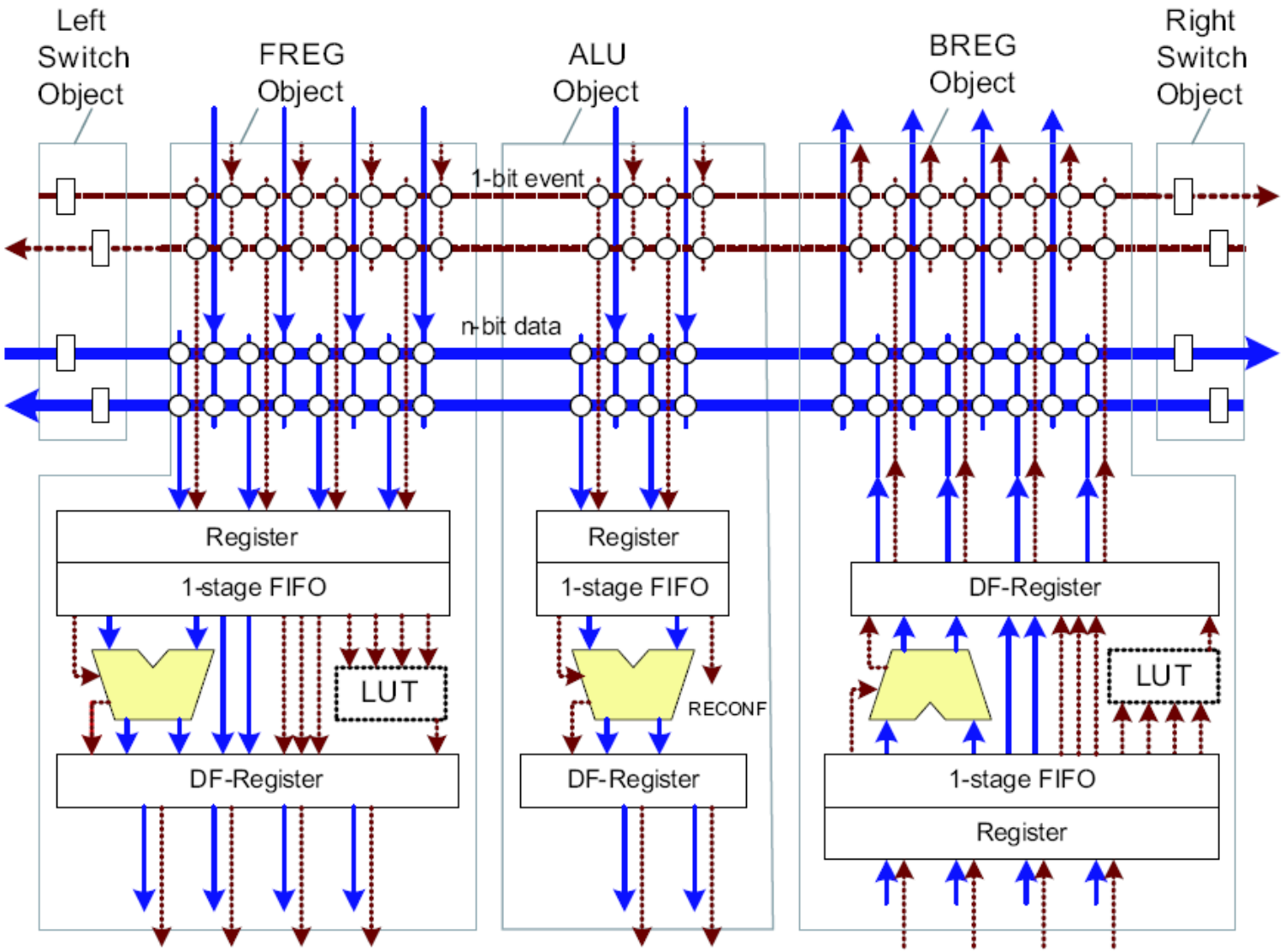
PACT: Static mapping, circuit-switch reconfigured NoC

PACT

x + iy = (a+ib) * (c+id)
= (ac - bd) + i (ad + bc)

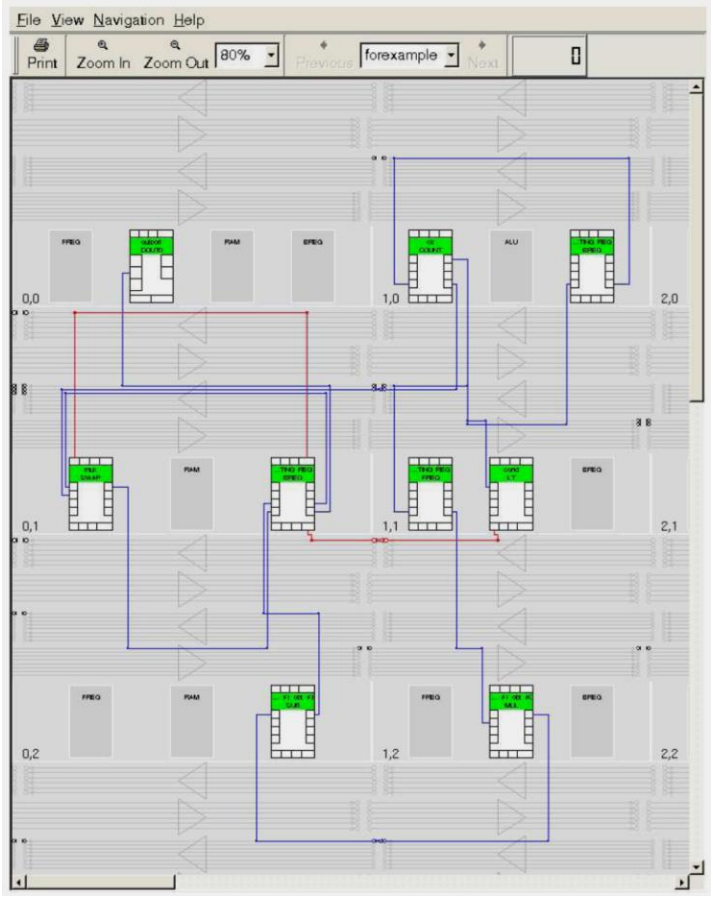


PACT ALU-PAE


Left
Switch
Object
FREG
Object
ALU
Object
BREG
Object
Right
Switch
Object
1-bit event
n-bit data
Register
1-stage FIFO
LUT
DF-Register
Register
1-stage FIFO
RECONF
DF-Register
DF-Register
LUT
1-stage FIFO
Register

PACT

- Static task mapping ☹
  - And a debug tool for that



PACT analysis

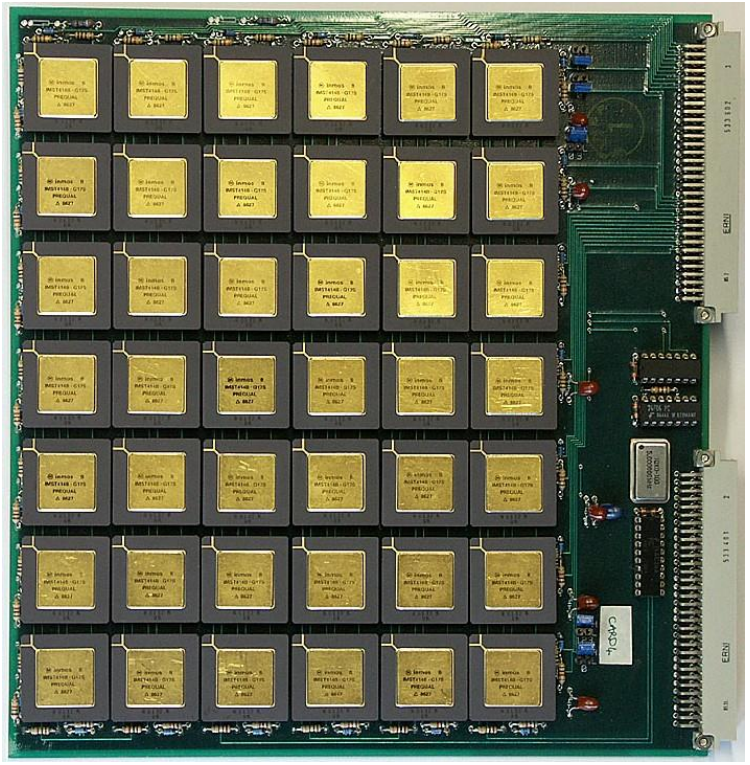
- Fine granularity computing ☺
- Heterogeneous processors ☹
- Static mapping

  → complex programming ☹
- Circuit-switched NoC → static reconfigurations

  → complex programming ☹
- Limited parallelism
- Doesn’t scale easily

picoChip

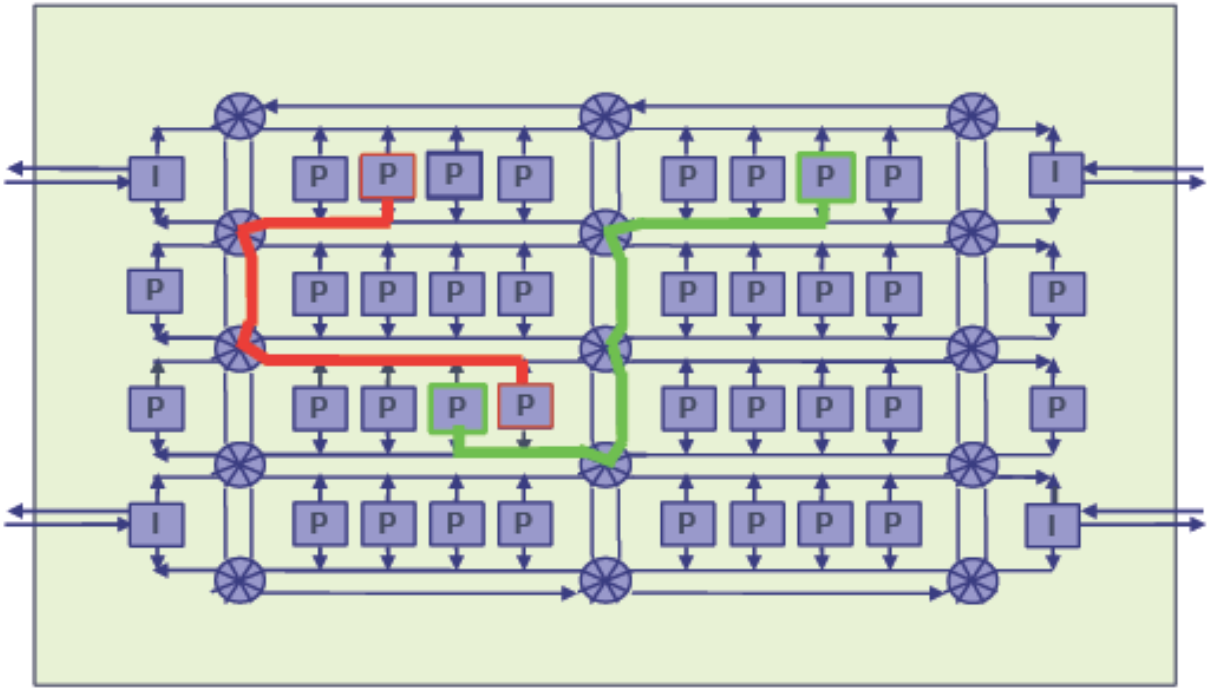
- UK company, sold to Mindspeed → MACOM
- Inspired by Transputers (1980s), David May



42x Transputers mesh 1980s

picoChip

The picoArray concept : Architecture overview



322x
16-bit LIW RISC

P Processor

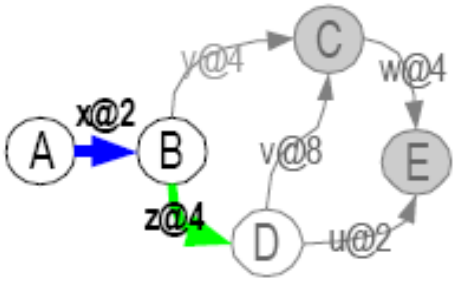
I Inter-picoArray Interface or Asynchronous Data Interface

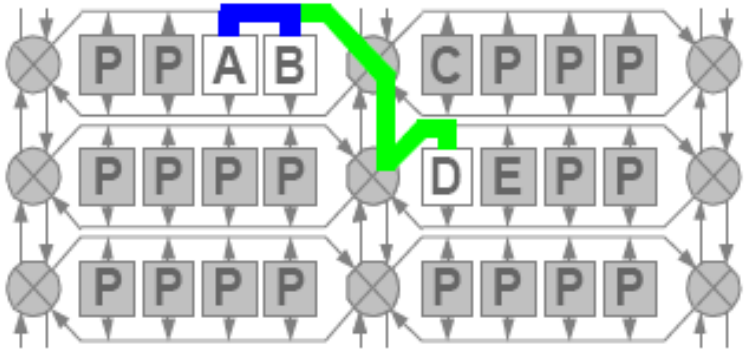
Switch Matrix

Example signal flows

picoChip

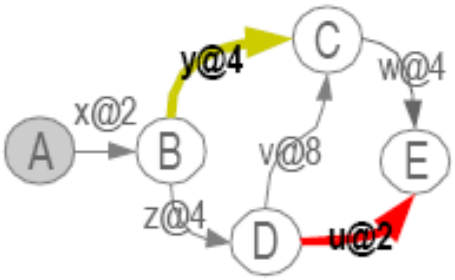
The picoArray concept : picoBus

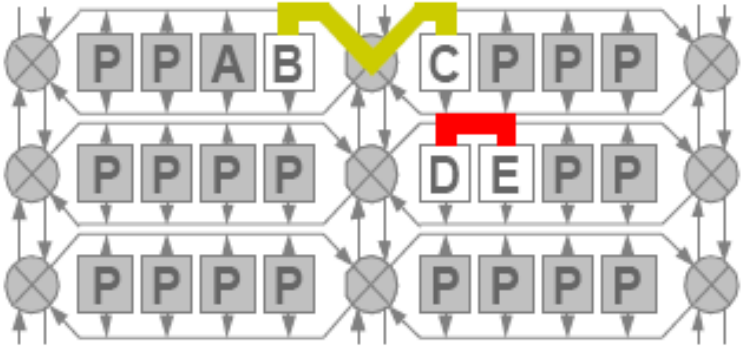




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

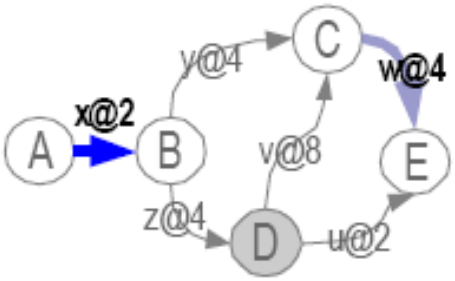
picoChip

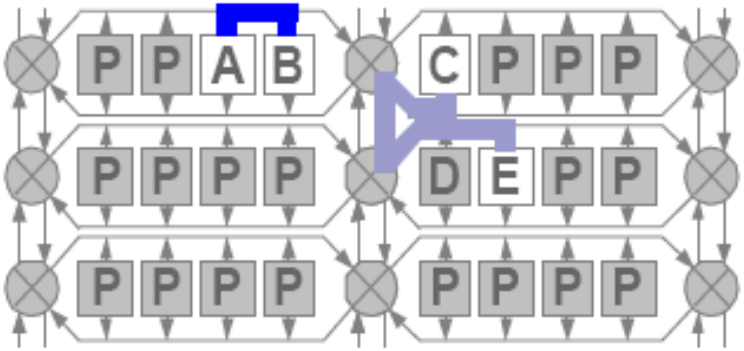




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

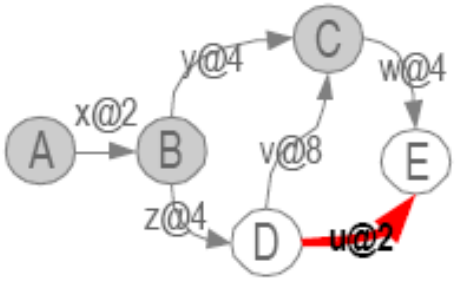
picoChip

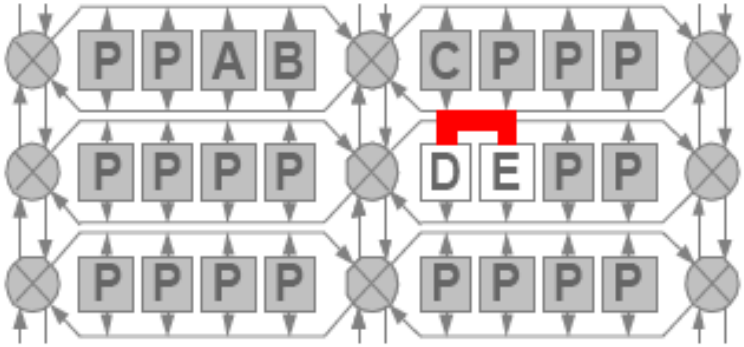




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

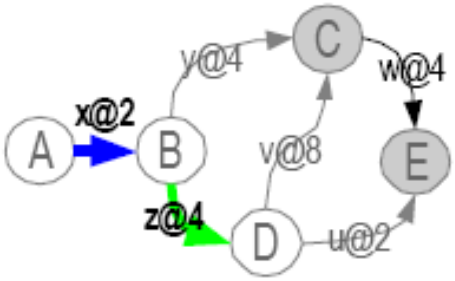
picoChip

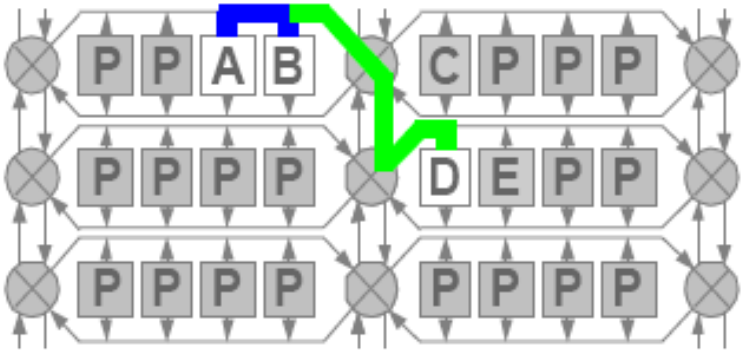




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

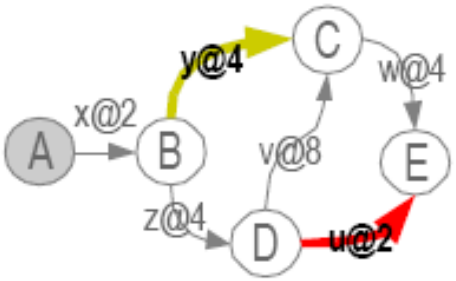
picoChip

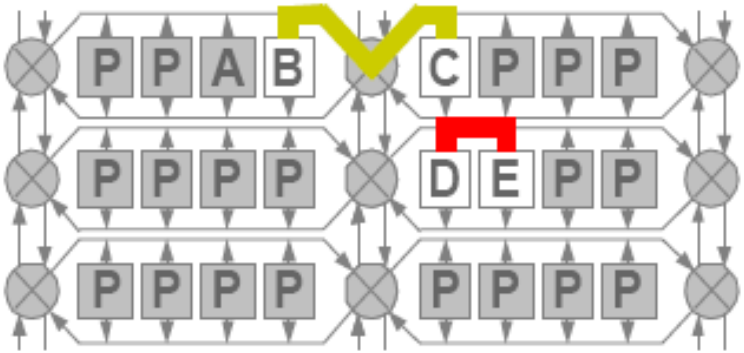




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

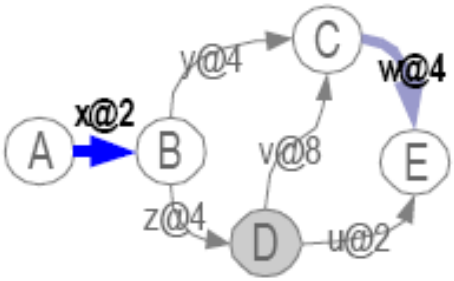
picoChip

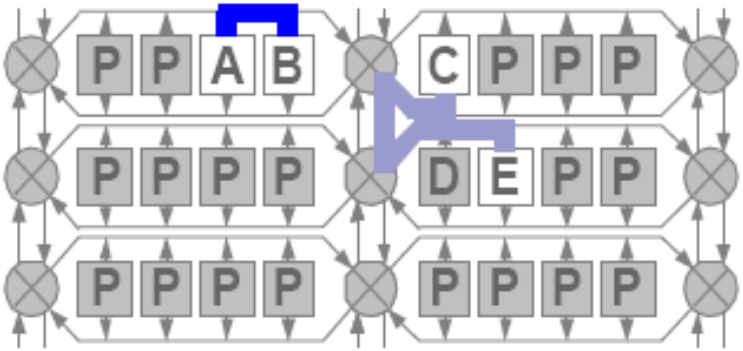




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

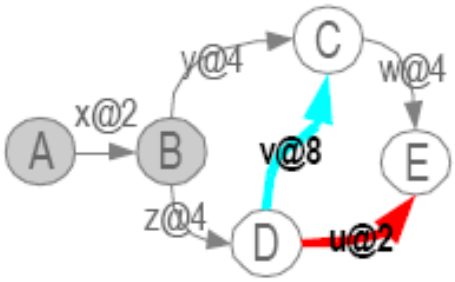
picoChip

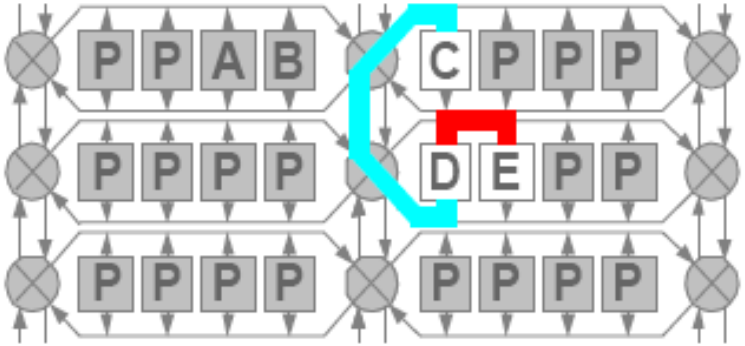




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

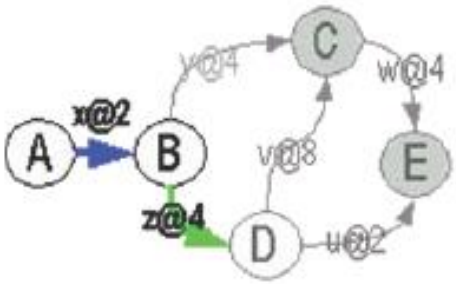
picoChip

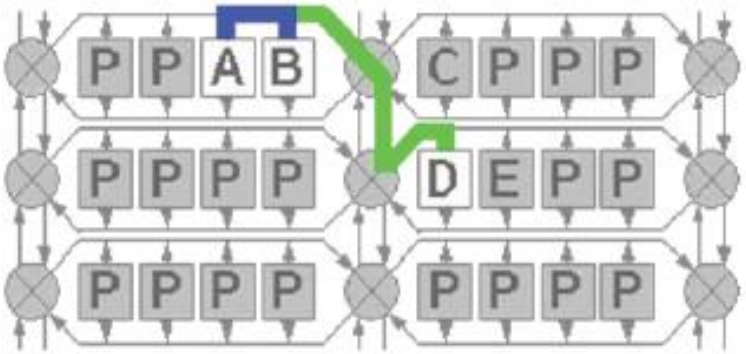




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

picoChip

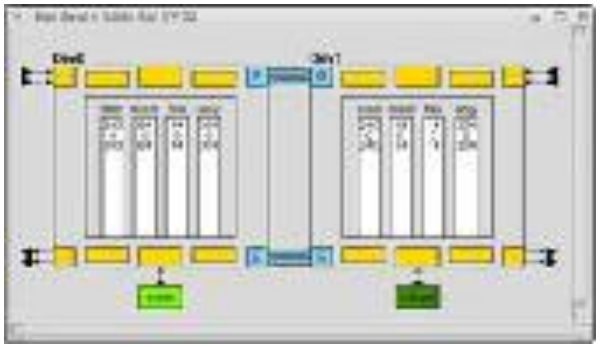




	0	1	2	3	4	5	6	7
U								
V								
W								
X								
Y								
Z								

picoChip: Static Task Mapping ☹

Compile

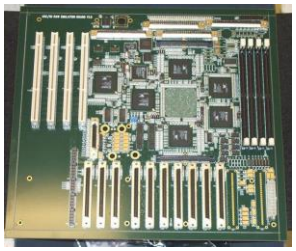


picoChip analysis

- MIMD, fine granularity, homogeneous cores ☺
- Static mapping
  → complex programming ☹
- Circuit-switched NoC → static reconfigurations
  → complex programming ☹
- Doesn't scale easily
  - Can we create / debug / understand static mapping on 10K?
- Resulted in restricted applicability
  - Users resisted adoption
  - Manufacturer driven to niche (LTE picocell)
    - Lost business flexibility. Buyer not known to benefit.

TILERA®

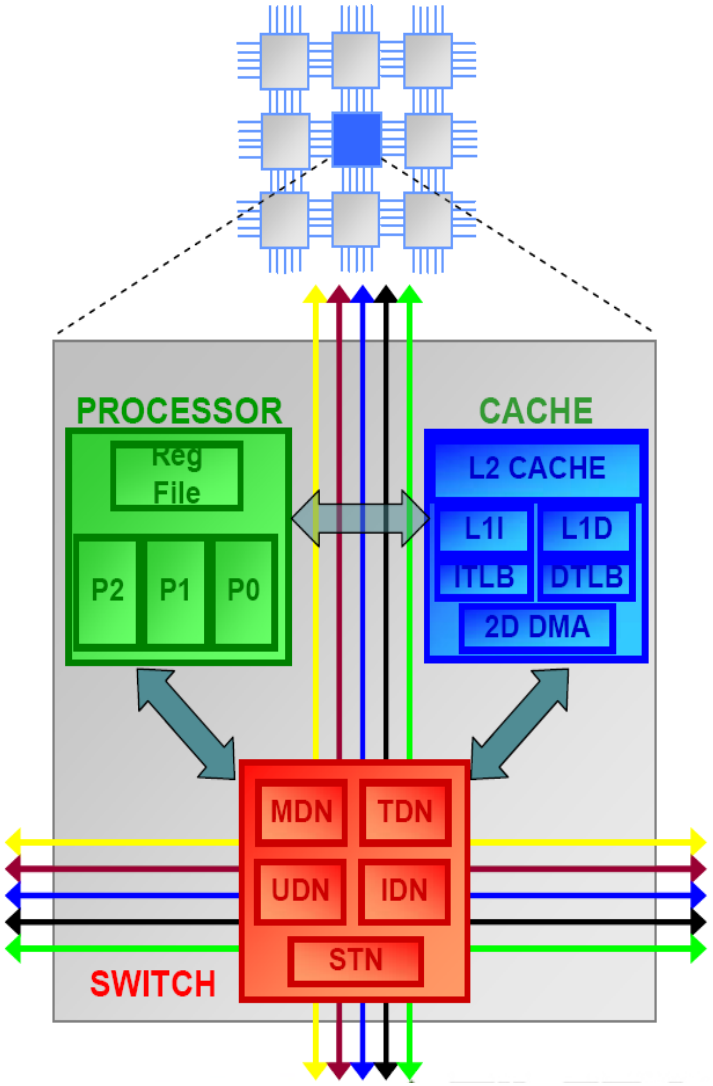
- USA company, sold to EzChip → Mellanox
- Based on RAW research @ MIT (A. Agarwal)

  - Heavy DARPA funding, university IP
- Classic homogeneous MIMD on mesh NoC
  - "Upgraded" Transputers with "powerful" uniprocessor features
    - Caches ☹
    - Complex communications ☹
- "tiles era"

TILERA Tiles

- Powerful processor → ARM
- High freq: ~1 GHz
  - High power (0.5W) ☹
- 5-mesh NoC
  - P-M / P-P / P-IO
- 2.5 levels cache ☹☹
  - L1+ L2
  - Can fetch from L2 of others
- Variable access time
  - 1 – 7 – 70 cycles

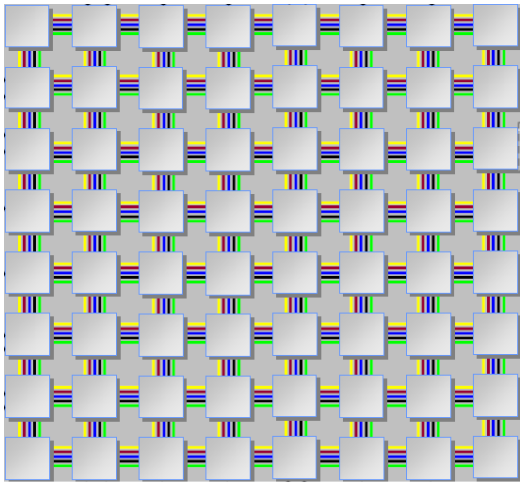


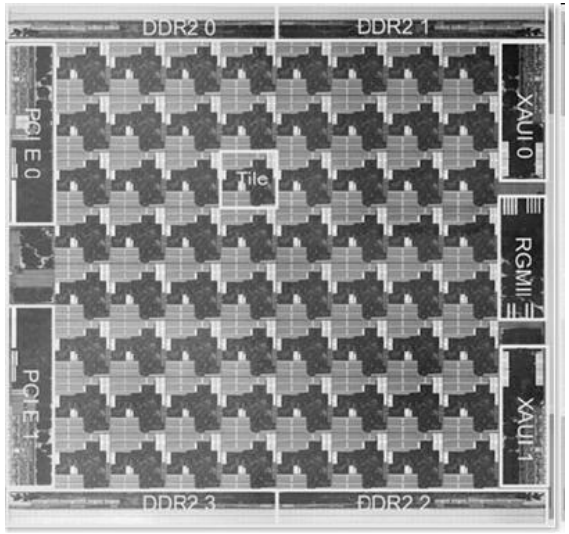
Cache-per-core may degrade performance

- Cache is great for a single processor
  - Exploits locality (in time and space)
- Locality only happens locally on many-cores
  - Other (shared) data are buried elsewhere
- Caches help speed up parallel (local) phases
- Need to program many looong tasks on same data
  - Hard ! That’s the software gap in parallel programming

TILERA Array

- 36-64 processors
  - MIMD / SIMD ☹
- Total 5+ MB memory
  - In distributed caches
- High power
  - ~27W ☹☹

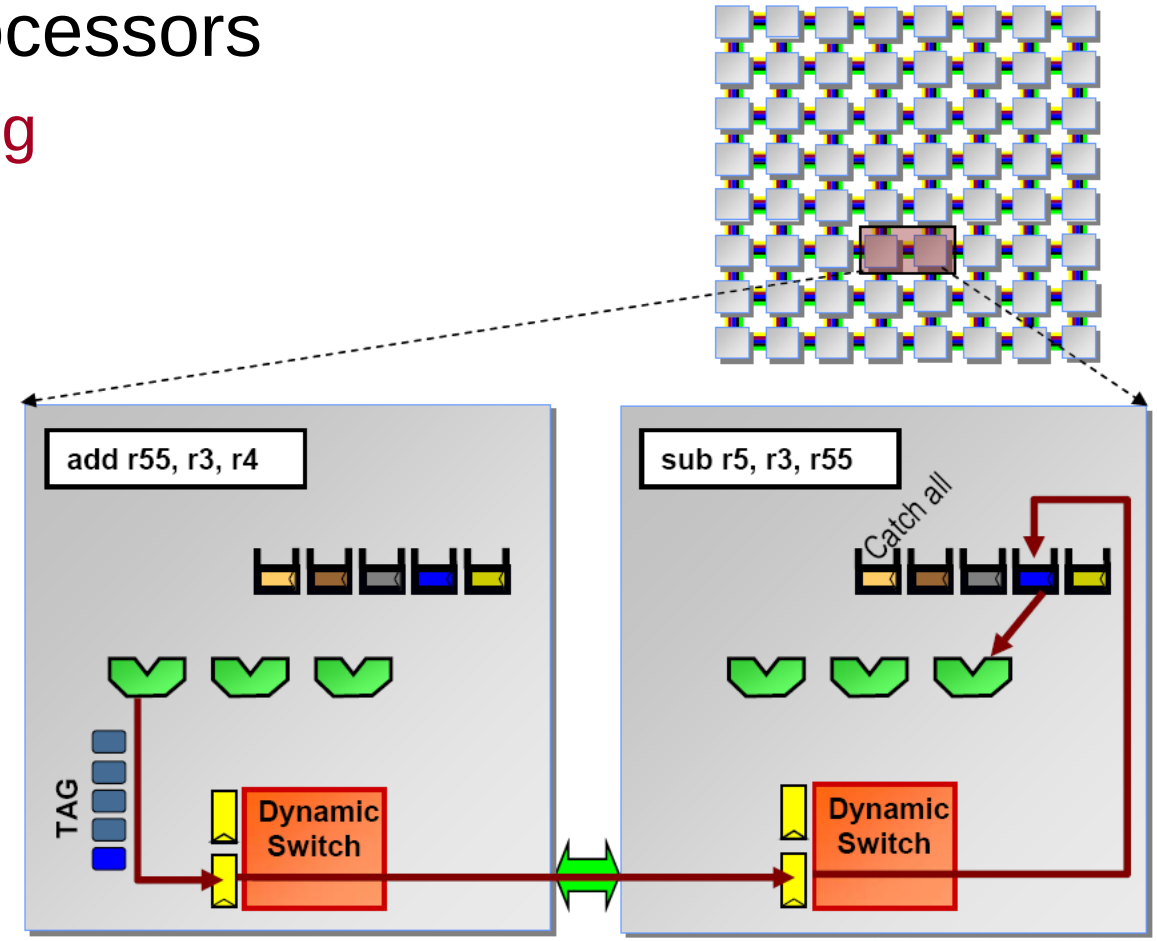




Die photo

TILERA allows statics

- Pre-programmed streams span multi-processors
  - Static mapping

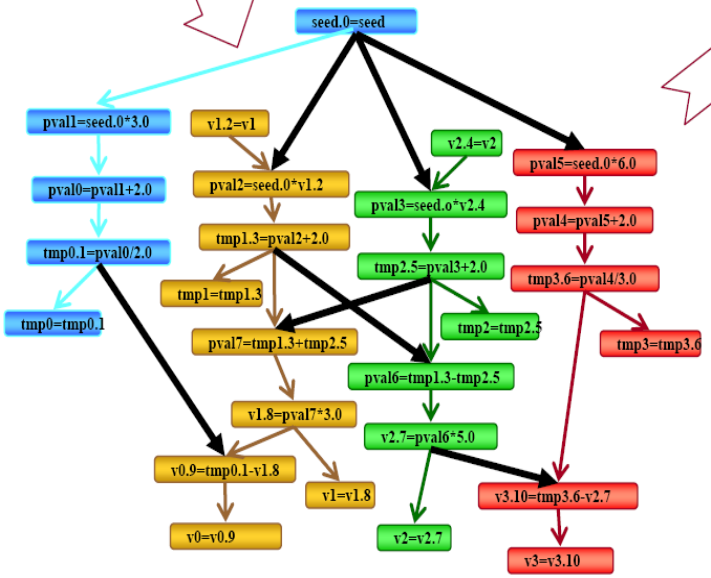


TILERA co-mapping: code, memory, routing ☹

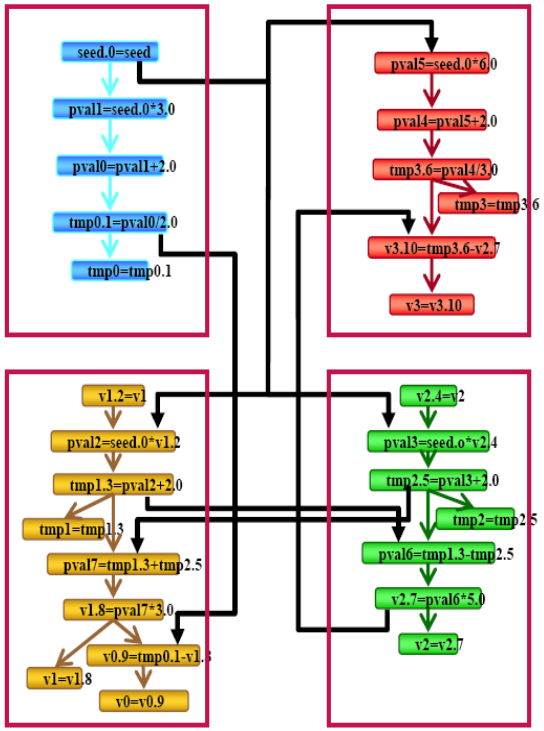
```
tmp0 = (seed*3+2)/2
tmp1 = seed*v1+2
tmp2 = seed*v2 + 2
tmp3 = (seed*6+2)/3
v2 = (tmp1 - tmp3)*5
v1 = (tmp1 + tmp2)*3
v0 = tmp0 - v1
v3 = tmp3 - v2
```

C

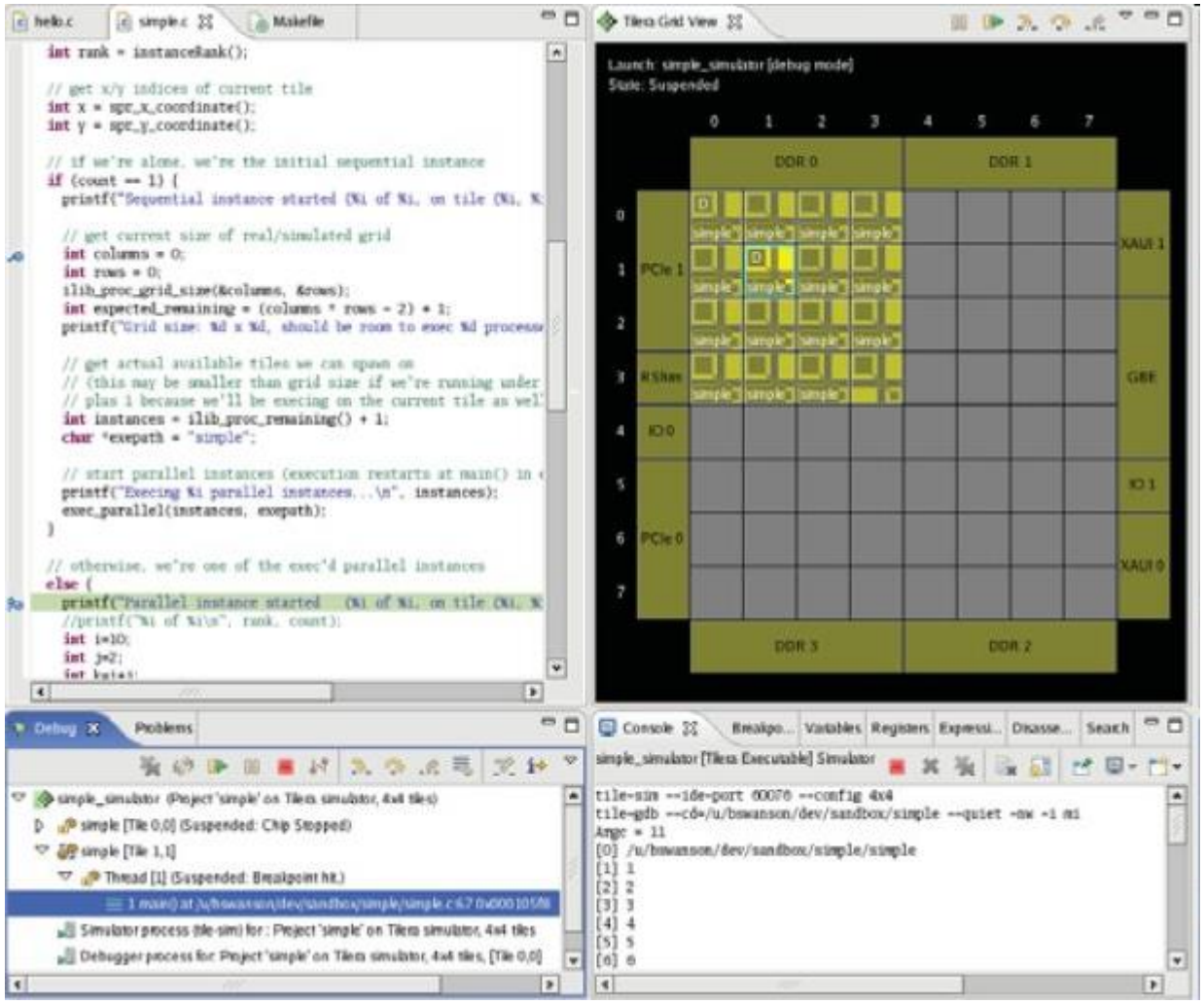
Partitioning



Place, Route, Schedule



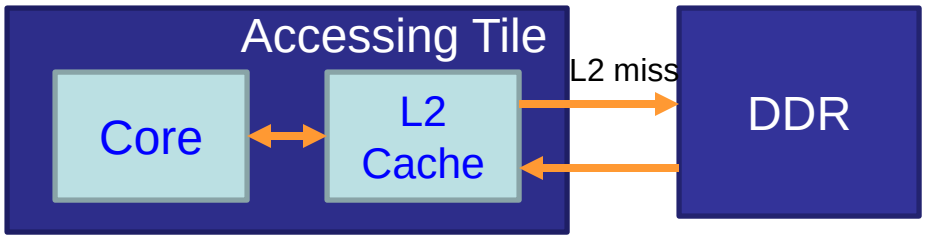
TILERA static mapping debugger ☹



Three Caching Schemes (1)

1. Local Homing

- No L3 cache
- Lower L2 cache-misses probability and no L3 overhead
- For non-highly-shared applications

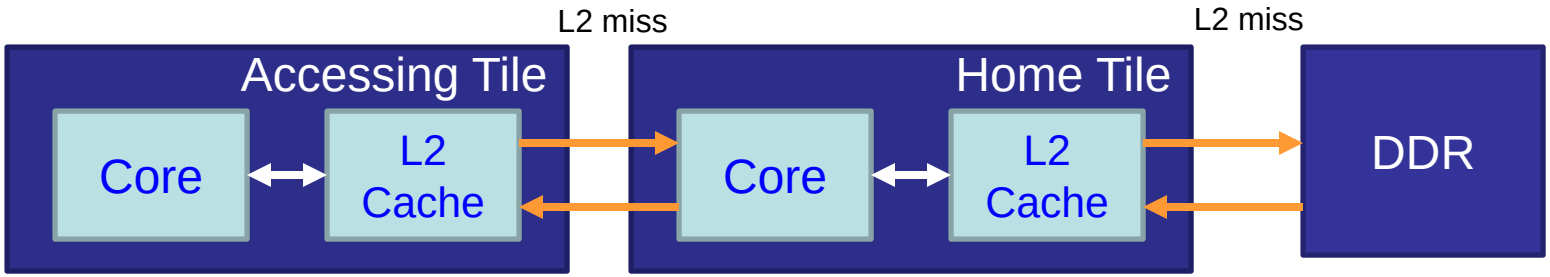


Slides from Ohn Tsarfati, TAU (2014)

Three Caching Schemes (2)

2. Remote Homing

- Each cache page matches one Home Tile
- The NoC knows where and goes there

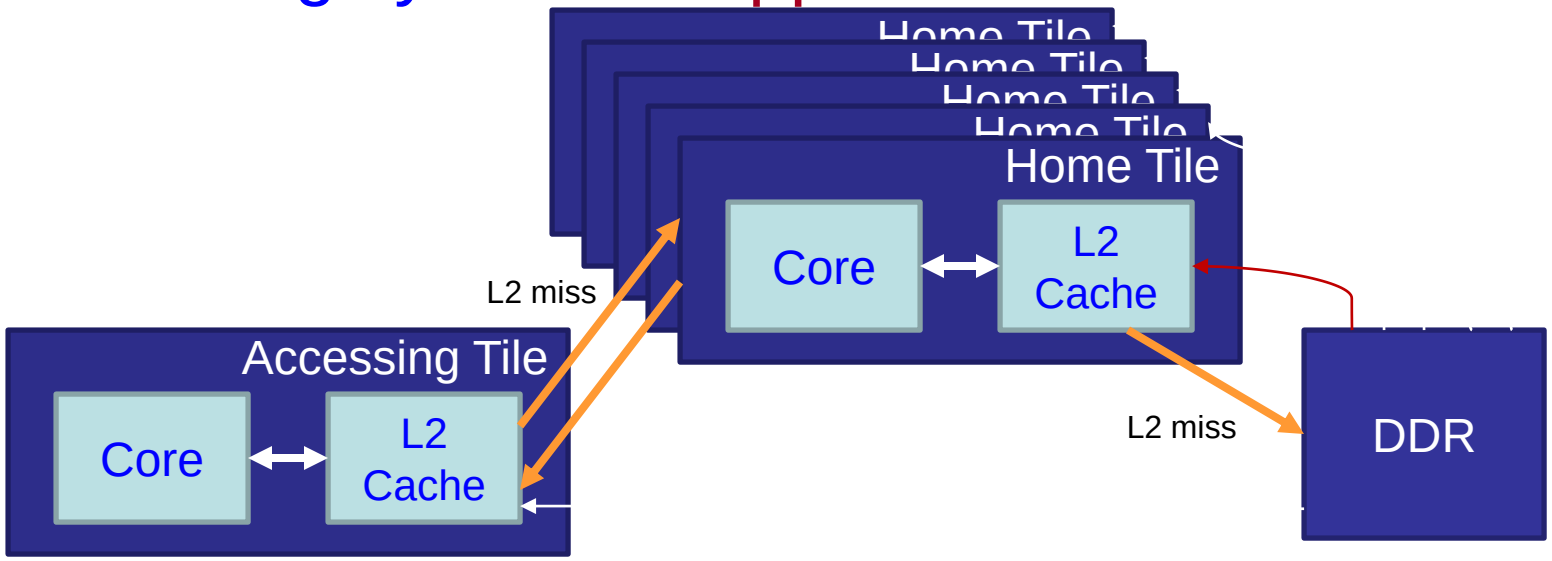


Slides from Ohn Tsarfati, TAU (2014)

Three Caching Schemes (3)

3. Hash-For-Home strategy

- The default mode
- Memory pages are spread out to lines
- Each line is cached in a different tile
- For highly-shared applications



Slides from Ohn Tsarfati, TAU (2014)

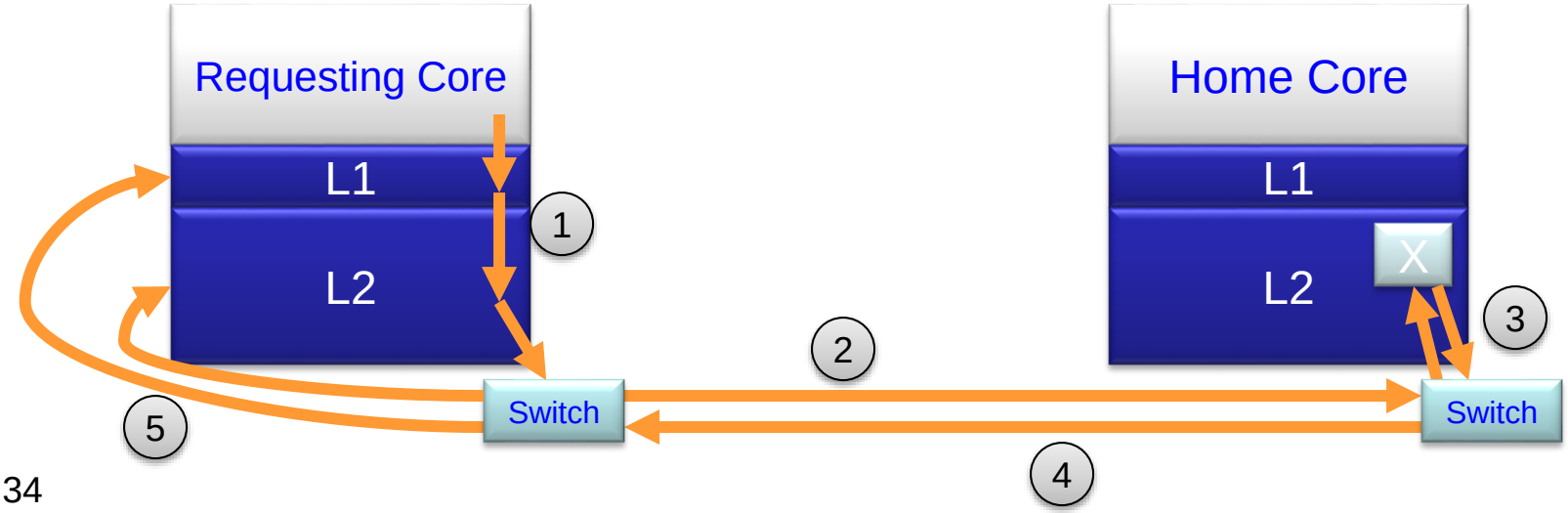
L3 read

- Operation of Remote Homing / Hash-for-Home
- L3 Cache coherence is preserved using the inter-tile network.
- Read example

Slides from Ohn Tsarfati, TAU (2014)

L3 read

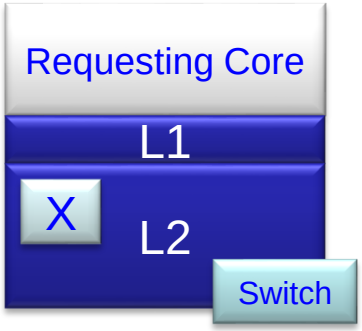
- Read example (2 messages)
  1. Core A misses on access to line “X”
  2. Request is sent to home tile of “X”
  3. Home tile of “X” retrieves cache line “X”
  4. Home tile of “X” sends cache line “X” to core A
  5. Cache line “X” is buffered in L2 and L1 of core A

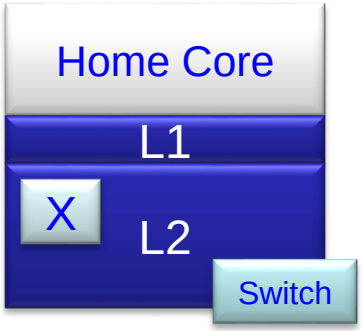


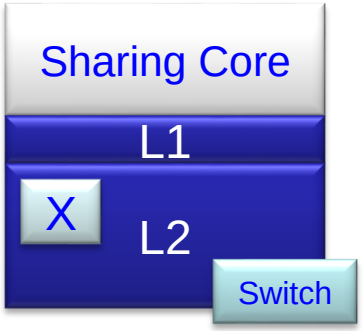
Slides from Ohn Tsarfati, TAU (2014)

L3 write

- Write example with 1 sharing core



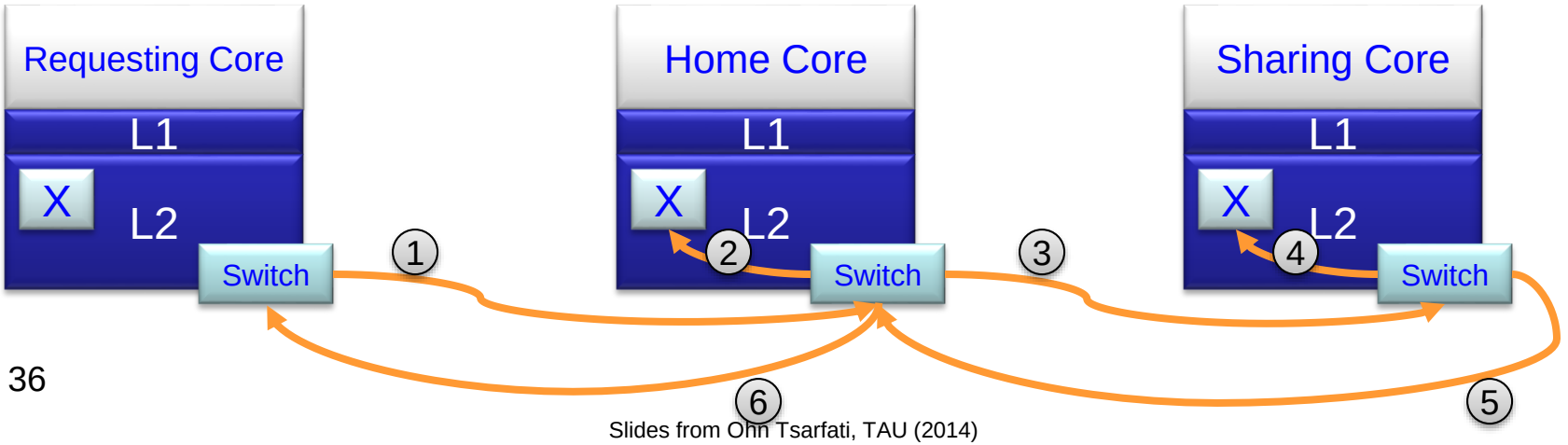




Slides from Ohn Tsarfati, TAU (2014)

L3 write

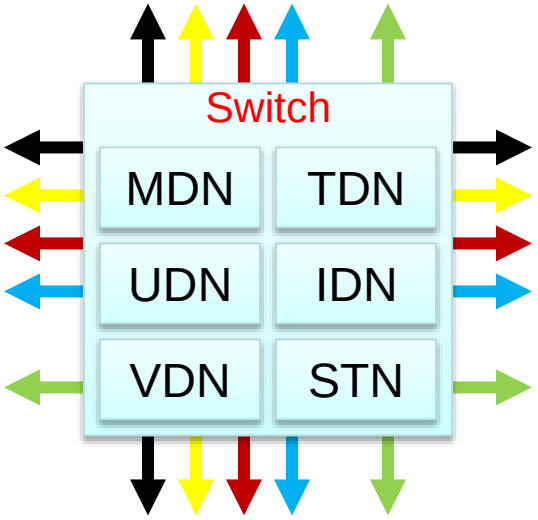
- Write example with 1 sharing core (4 messages)
  1. Requesting Core writes new data to “X”, and sends write-request to Home Core
  2. Home Core updates data in “X”. It knows other sharing cores of data “X”
  3. Home Core sends invalidation message to Sharing Core
  4. Sharing Core invalidates cache line “X”
  5. Sharing Core sends invalidation ACK to the Home Core
  6. Home Core sends write ACK to Requesting Core..



Slides from Ohn Tsarfati, TAU (2014)

The Tilera multi-NoC

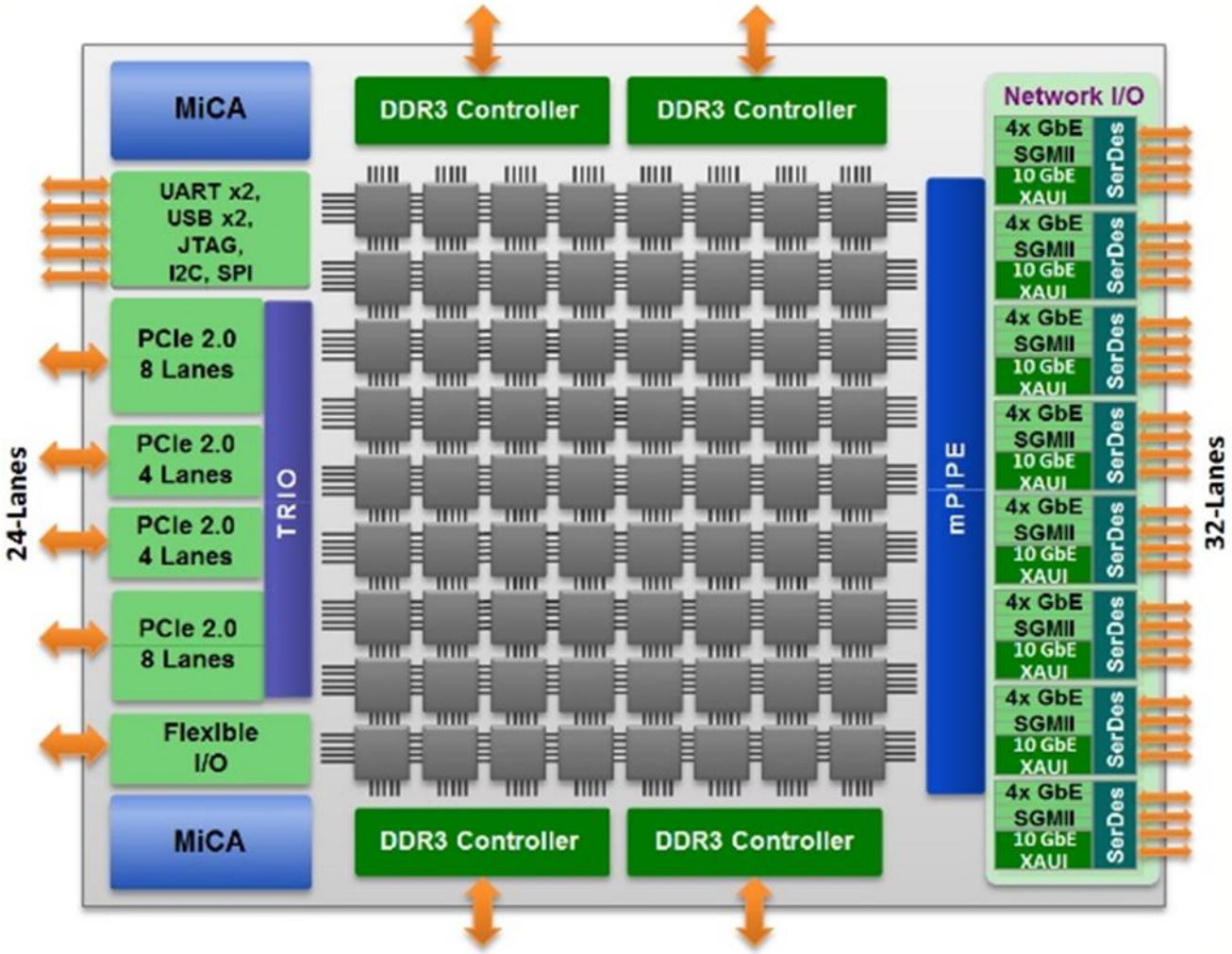
- The grid consists of 6 networks:
  - Memory (MDN, Memory Dynamic Net)
  - Tile L3 requests (TDN, Tile Dynamic Net)
  - Cache Coherence Invalidations (VDN, Validation Dynamic Net)
  - Message Passing (UDN, User Dynamic Net)
  - I/O (IDN, IO Dynamic Net)
  - Static Network for configurations (STN)



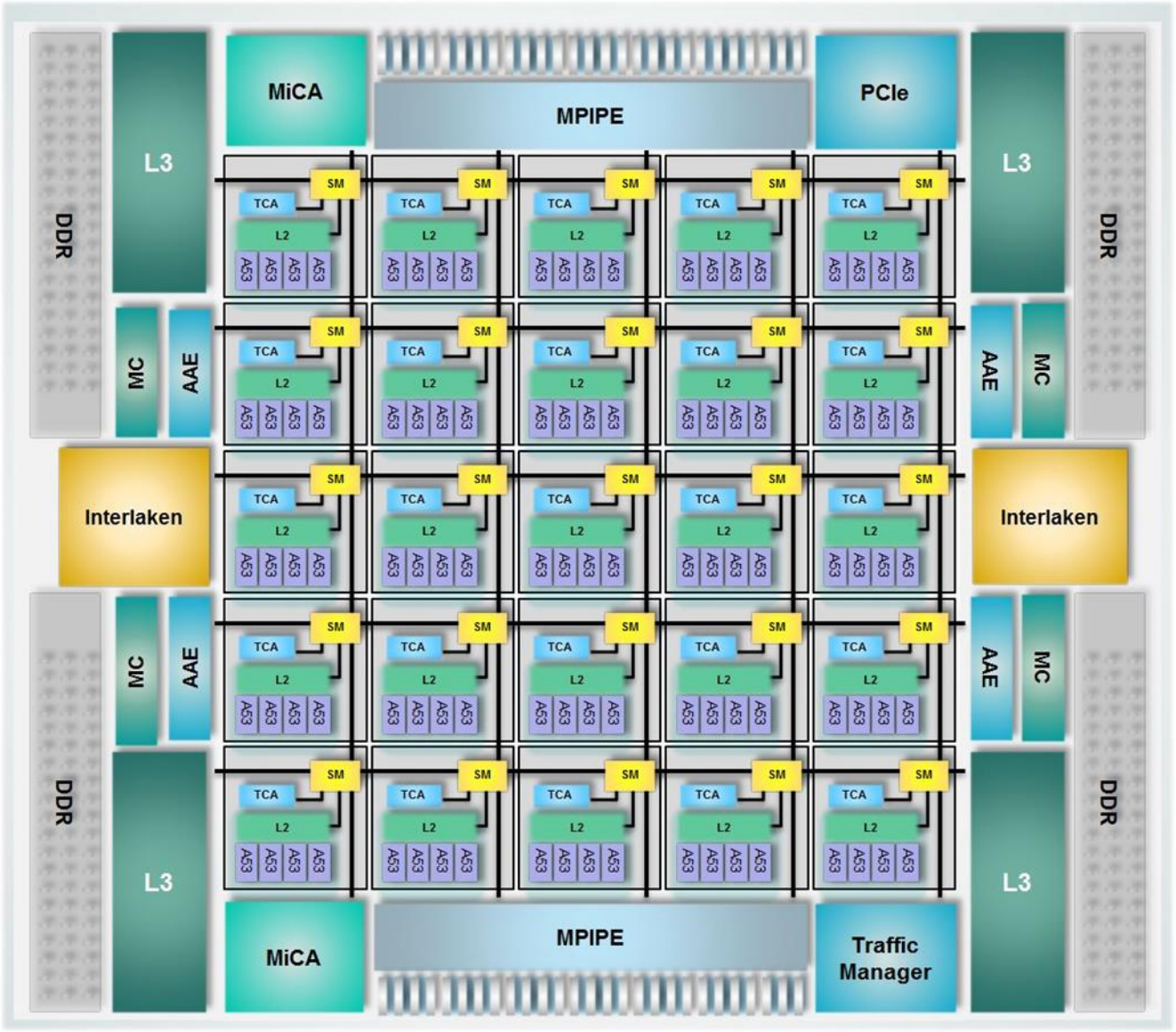
Slides from Ohn Tsarfati, TAU (2014)

TILERA® analysis

- Achieves good performance
- Bad on power
- Hard to scale
- Hard to program
- Resulted in restricted applicability
  - Users resist adoption
  - Manufacturer driven to deliver “solutions”
    - Boards/modules/systems, software/algorithms
    - Customized
  - Driven to niche/custom markets
    - Mostly communications
    - Some HPC


MiCA
DDR3 Controller
DDR3 Controller
Network I/O
UART x2,
USB x2,
JTAG,
I2C, SPI
PCIe 2.0
8 Lanes
PCIe 2.0
4 Lanes
PCIe 2.0
4 Lanes
PCIe 2.0
8 Lanes
TRIO
24-Lanes
Flexible
I/O
MiCA
DDR3 Controller
DDR3 Controller
mPIPE
4x GbE
SGMII
10 GbE
XAUI
SerDes
32-Lanes

Tilera→EzChip→Mellanox: 100×A53



Kalray MPPA®

Massively Parallel Processing Array

Next-Generation Accelerated Computing

February 2015

Accelerated Computing

Motivations of Accelerated Computing

- Computing challenges are performance and energy efficiency
  - Portable electronics (mobile phones, tablets, health)
  - High-performance embedded computing
  - Video compression and transcoding
  - High-speed networking and storage
  - Numerical supercomputing
  - Big data processing
- Parallel computing increases performance or energy efficiency
  - $P = \frac{1}{2} C_{sw} V_{dd} \Delta V f + I_{st} V_{dd} + I_{static} V_{dd}$
  - N units operating at f/N consume less energy
    - Decreasing f allows to reduce V_{dd} and ΔV
  - N units operating at f compute N times faster
    - Assuming computation is concurrent
- Other improvements from the specialization of compute units

HPC and Enterprise Applications (Intel)

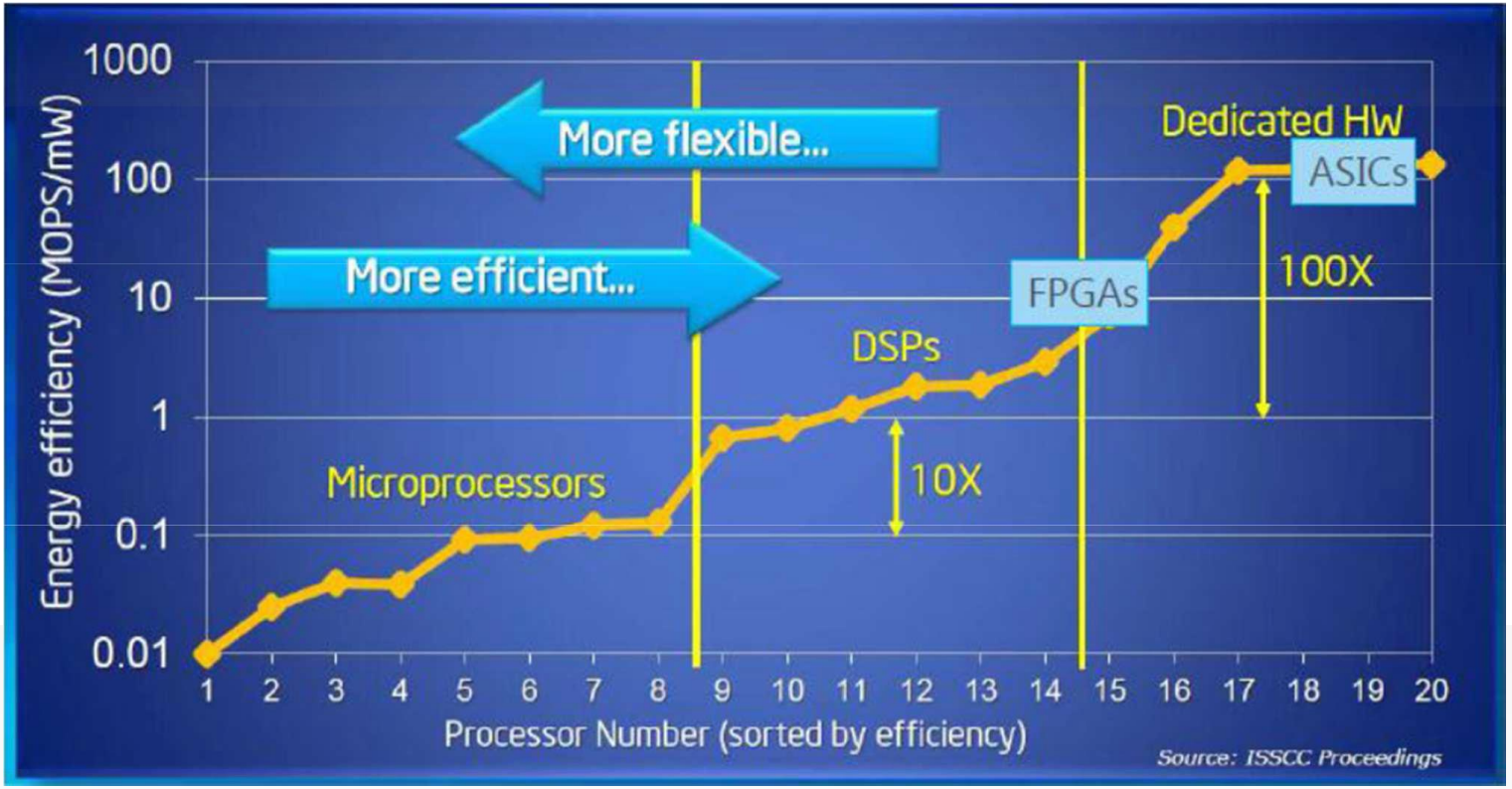
High Performance Embedded Computing (TI)

KALRAY

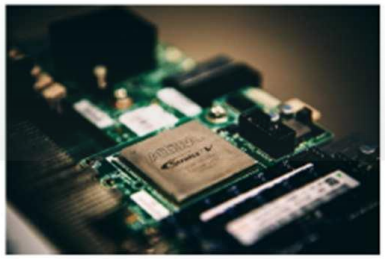
Classes of Computing Accelerators

- Application-Specific Integrated Circuits (ASIC)
  - Most effective accelerators, based on specialization
  - Long design times, not adaptable to evolving applications / standard
- Field-Programmable Gate Arrays (FPGA)
  - Most effective on bit-level computations
  - Require HDL programming
- Digital Signal Processors (DSP)
  - Most effective on fixed-point arithmetic
  - Require assembler-level programming
- Graphics Processing Units (GPU)
  - Most effective on regular computations with dense memory accesses
  - Require massive parallelism and CUDA or OpenCL programming
- Intel Many Integrated Core (MIC)

Efficiency of CPUs, DSPs, FPGAs, ASICs



FPGA Acceleration in Datacenters (Nallatech)

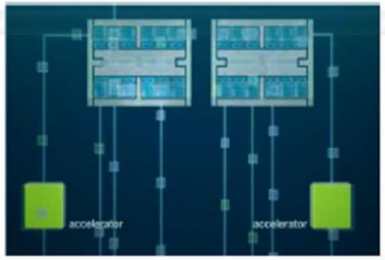


- "Catapult"
- Accelerate BING
- 2x search acceleration:
  - ½ the number of servers
  - <30% of added server TCO
  - ~10% increase in power

Microsoft bing

- Intel Xeon + FPGA MCM
- Announced by Diane Bryant
  - SVP & GM of Intel Datacenter Group (DCG)
- 10x acceleration for certain tasks
- QPI gives 2x speed up again

intel

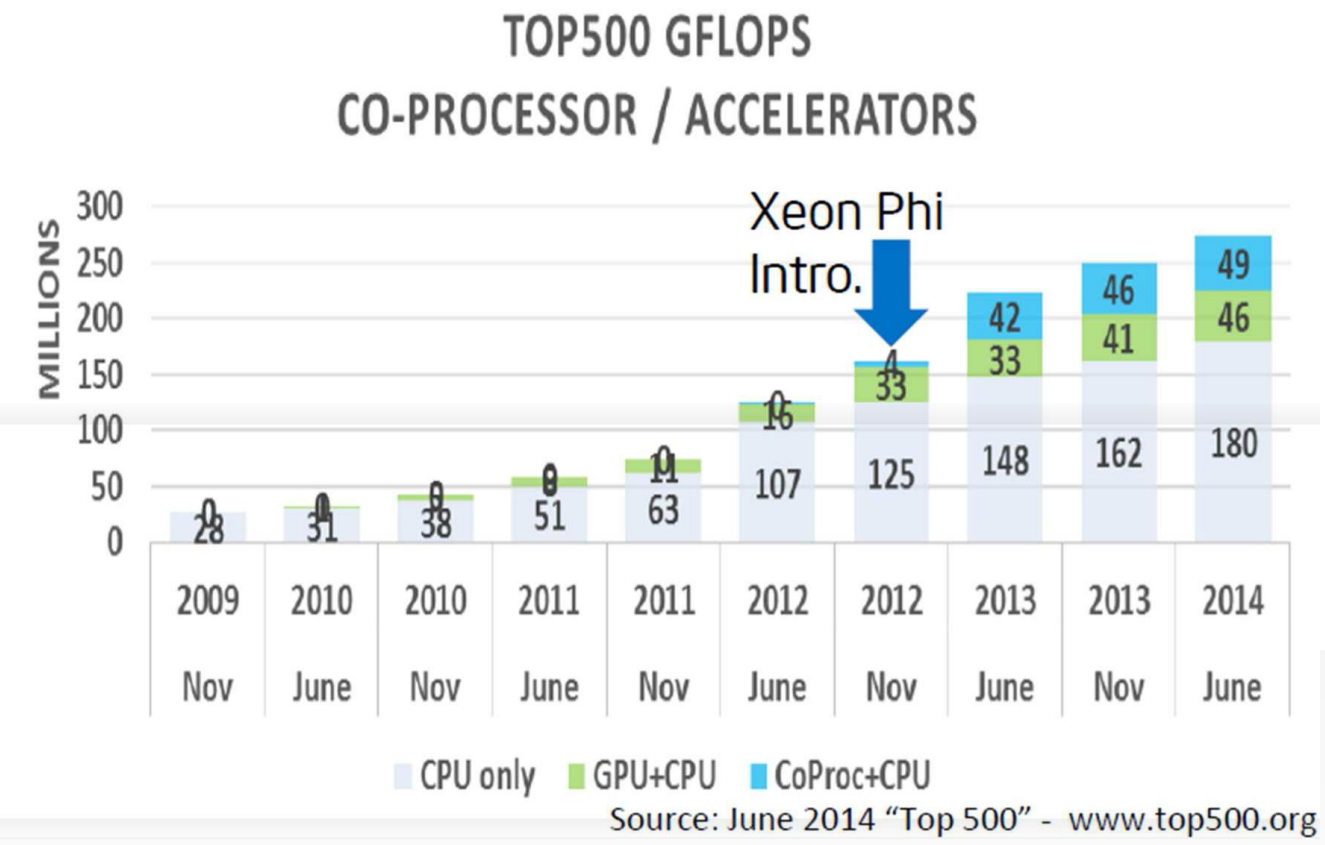


- IBM POWER architecture now open
- Coherent Accelerator Processor Interface (CAPI)
  - FPGA card now a processor peer
  - Coherent memory access (1TB)
  - PCIe latency reduced by 90%

IBM

OpenPOWER™

GPU and MIC Acceleration in HPC (Top500)



DSP vs GPU and MIC Acceleration (TI 2012)

TI Quad Shannon PCIe ~50 W (2011

~12.8 Gflops/W SP
~3.2 Gflops/W DP

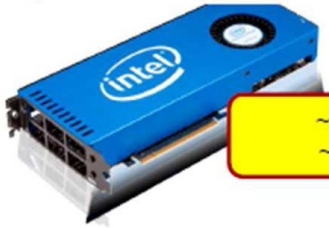
Nvidia Kepler ~250 W (2012)

- Dominates acceleration today
- Powers #2 Supercomputer (Titan)

~12 Gflops/W SP
~4 Gflops/W DP

Intel Xeon PHI (MIC) ~250 W (2012)

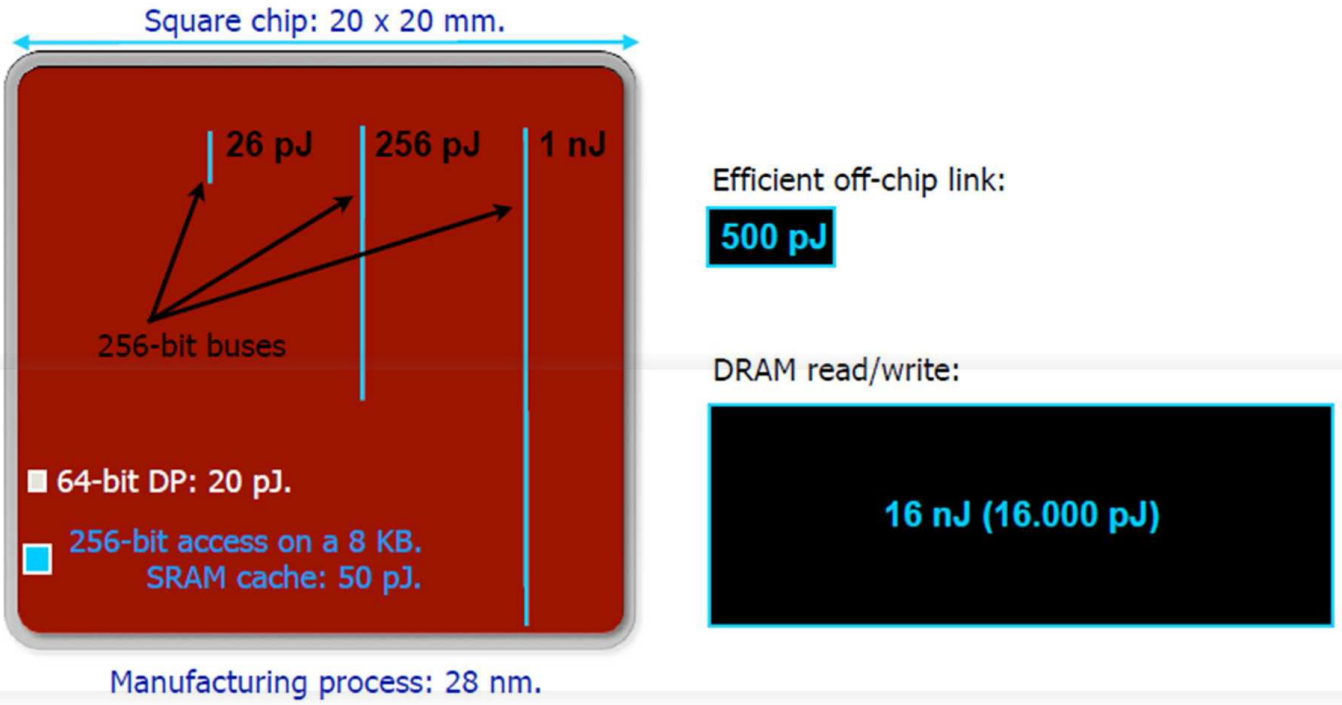
- Unveiled at SC'12
- Powers #1 Supercomputer (Tian 2)



~8 Gflops/W SP
~4 Gflops/W DP

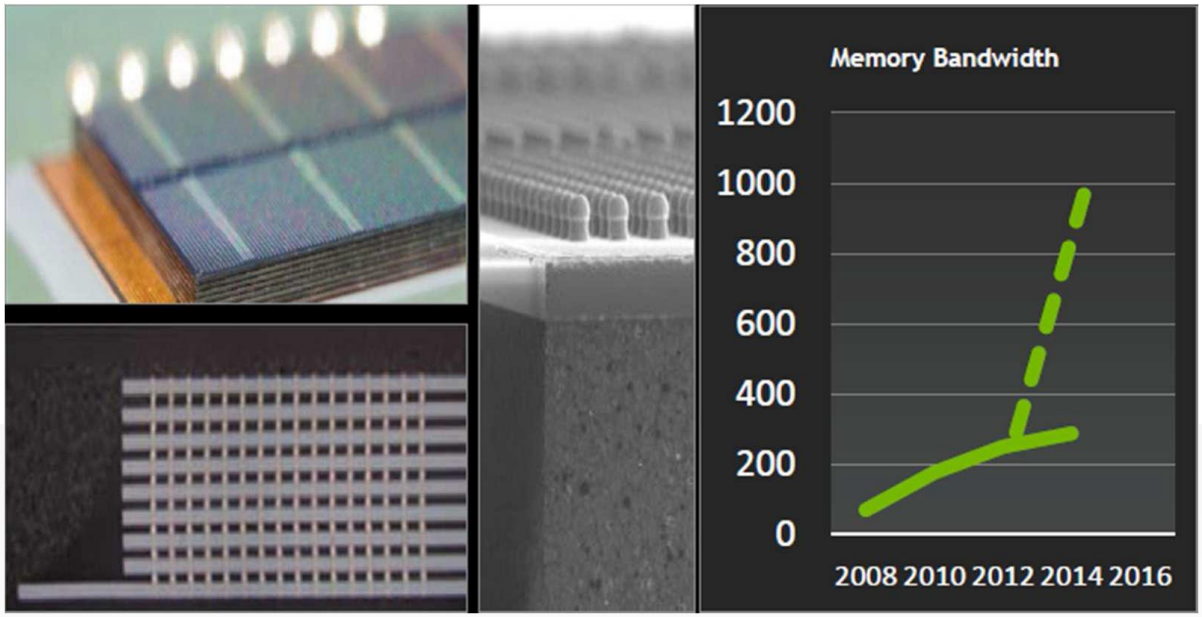
Energy Efficiency Drives Locality (NVIDIA)

- Communications take the bulk of power consumption
- Architectures should expose a memory hierarchy to the programmer



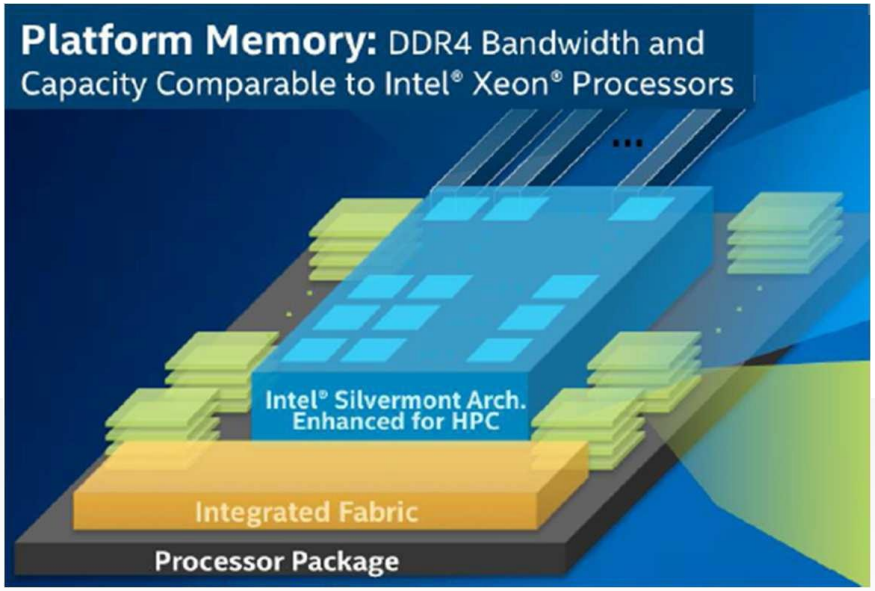
3D Memory: 5x Bandwidth of DDR4

- Micron: Hybrid Memory Cube (HMC)
- SK hynix: High Bandwidth Memory (HBM)
- Tezzaron: 3DDRAM memory



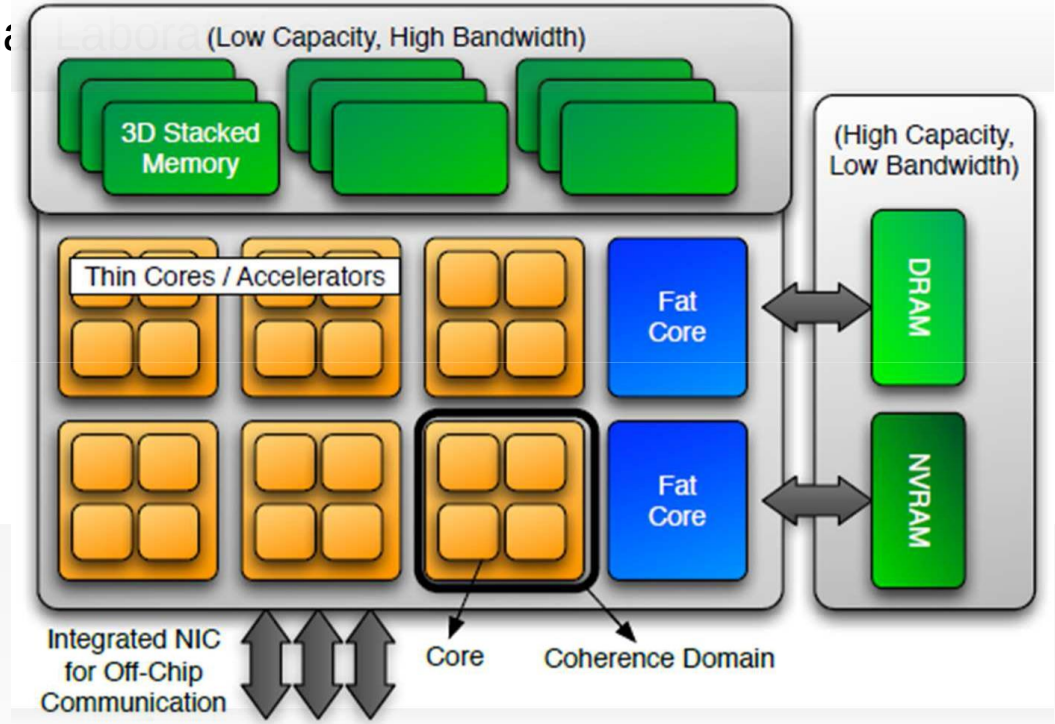
MIC Knights Landing (Intel 2015Q2)

- Xeon-Phi (MIC) accelerator, 60+ cores, 2D mesh fabric
- System in package with Micron HMC / low-power interface
- DDR4 external memory



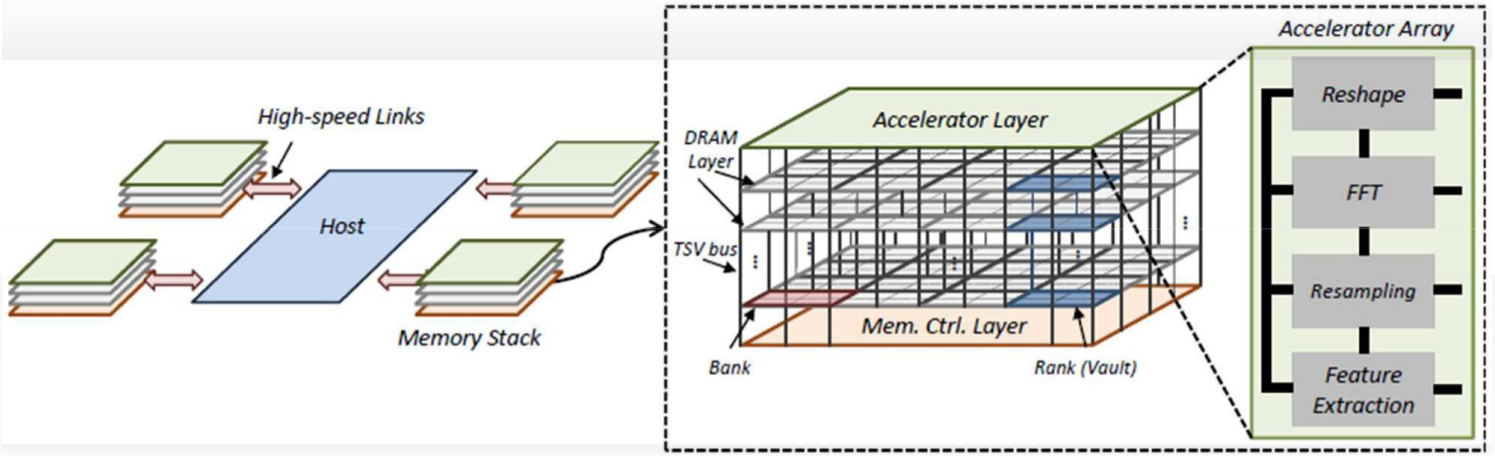
Abstract Machine Model of an Exascale Node

- « Abstract Machine Models and Proxy Architectures for Exascale Computing » 2014 report from Sandia and Lawrence Berkeley Nationa



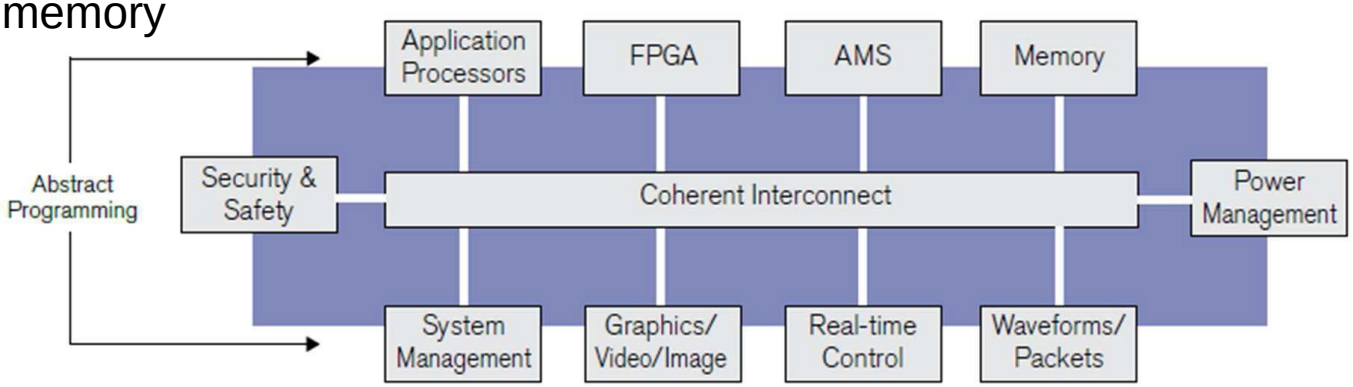
3D Stacked Memory-Side Acceleration (2014)

- Near-data computing reduces the overhead on performance and energy consumption incurred by the data movement between memory and computational elements
- Acceleration logic should be co-located with the 3D memory stack
- Best applied to Micron HMC type of 3D memory, where a logic layer exists for packet switching between the host and the memory chips



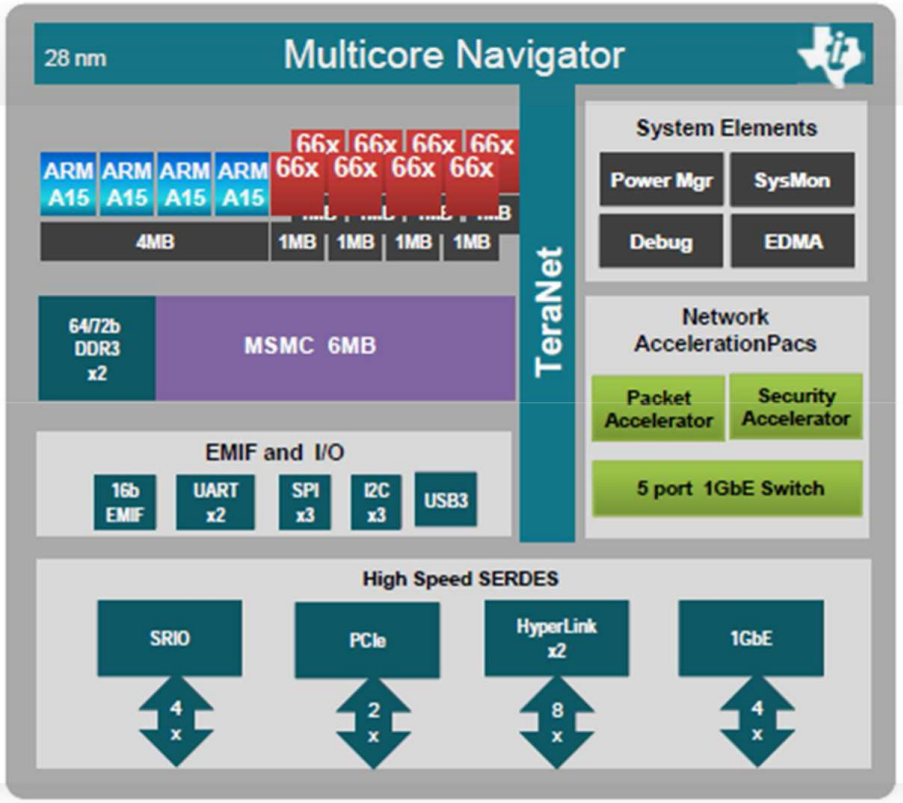
Xilinx Zinq UltraScale MPSoC Architecture

- Five Processing Bottlenecks
  - Digital Signal Processing
  - Graphics processing
  - Network processing
  - Real-time processing
  - General computing
- Scalable from 32 to 64 bits
- Coherent interconnect and memory
- FPGA ASIC-class logic fabric
  - Extreme real-time performance
- Multi-level security and safety
  - Enhanced anti-tamper features
  - Trust and information assurance
- Advanced power management
- High-level design abstractions
  - Based on C, C++, OpenCL



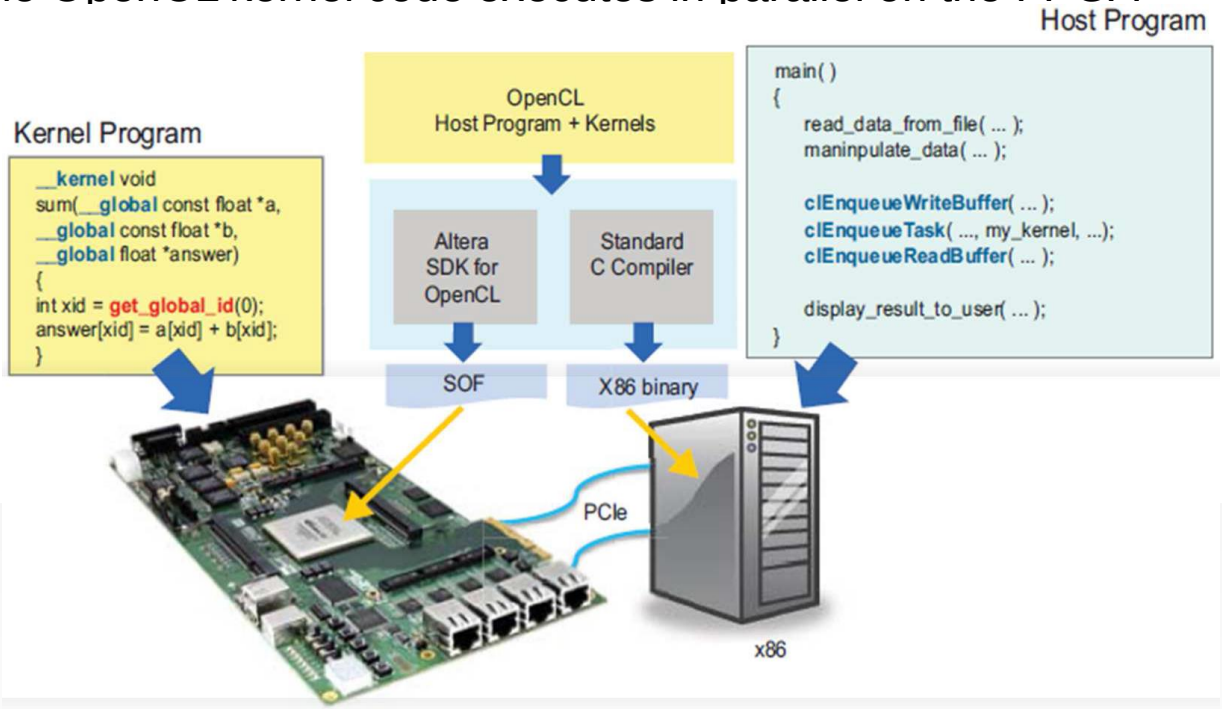
TI KeyStone-II 66AK2H12 SoC

- 15-20W typical
- 4x ARM A15 1.2 GHz
  - 4MB L2 shared by cores
  - 19.2/9.6 GFLOPS SP/SP
- 8x TI C66x DSP 1.2GHz
  - 1MB L2 private per core
  - 160/60 GFLOPS SP/DP
- Shared memory
  - 6MB on-chip
  - 2x DDR3
- Peripherals
  - 4x 1G Ethernet
  - 4x SRIO 2.1
  - 2x PCIe

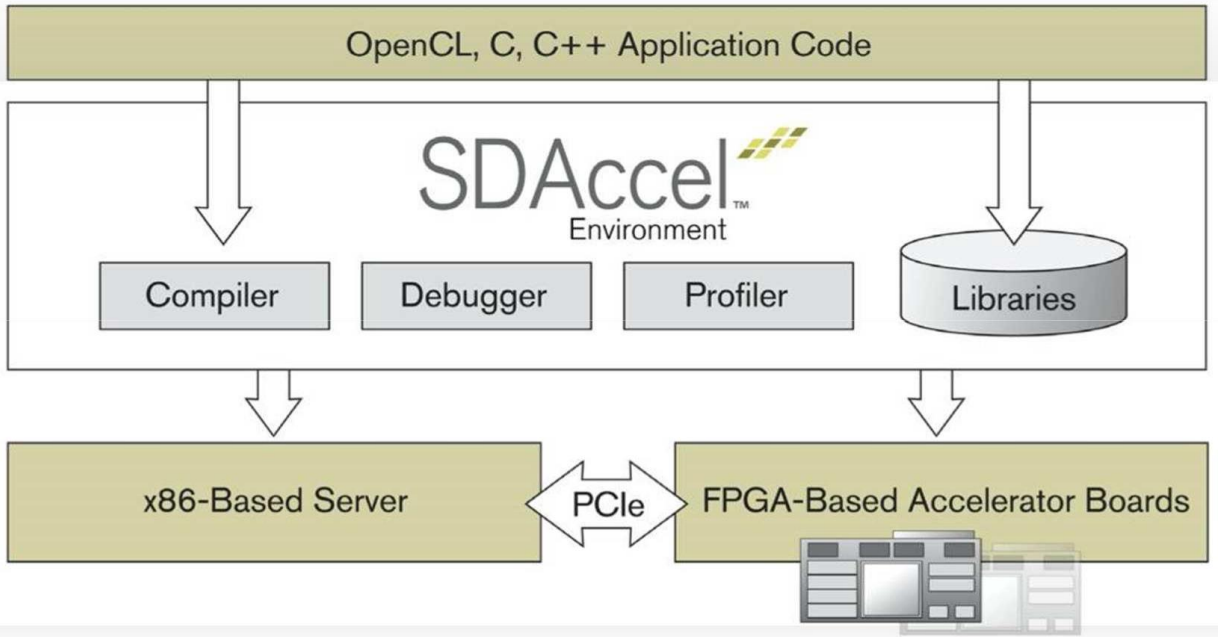


Altera FPGA OpenCL Tool Chain

- The OpenCL host program is a pure C/C++ x86 software routine
- The OpenCL kernel code executes in parallel on the FPGA

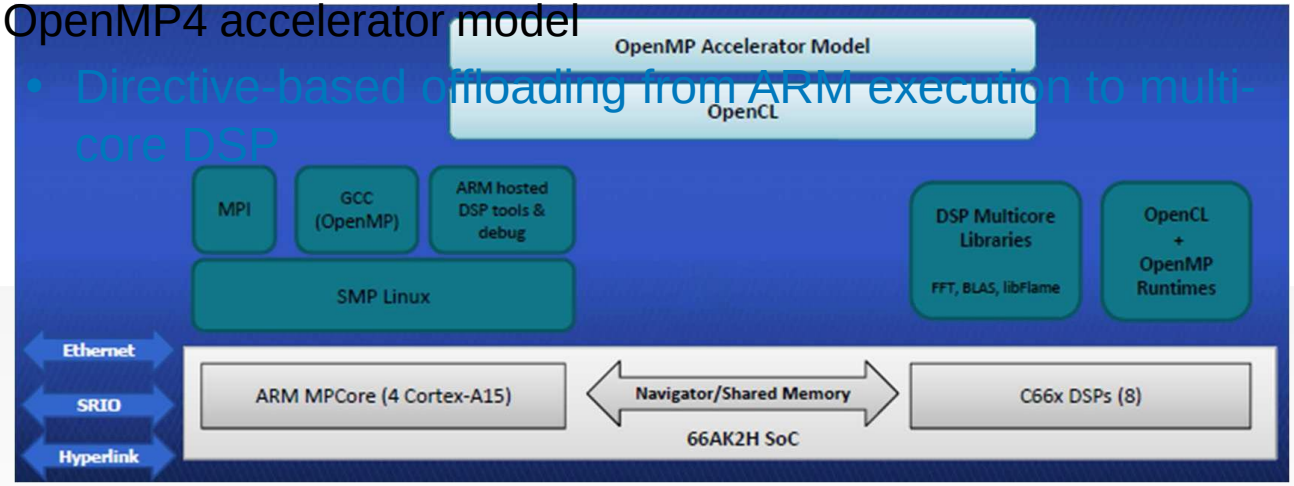


Xilinx FPGA C/C++/OpenCL Tool Chain



TI KeyStone-II OpenCL/OpenMP Tool Chain

- Classic OpenCL
  - The OpenCL host program is a pure C/C++ ARM software routine
  - The OpenCL kernel code executes in parallel on multiple DSPs
- OpenMP4 accelerator model
  - Directive-based offloading from ARM execution to multi-core DSP



Kalray MPPA® Technology

Kalray MPPA® Acceleration Highlights

- DSP type of acceleration
  - Energy efficiency
  - Timing predictability
  - Software programmability
- CPU ease of programming
  - C/C++ GNU programming environment
  - 32-bit or 64-bit addresses, little-endian
  - Rich operating system environment
- Integrated multi-core processor
  - 256 application cores on a chip
  - High-performance low-latency I/O
- Massively parallel computing
  - MPPA® processors can be tiled together
  - Acceleration benefits from thousands of cores

Kalray MPPA®, a Global Solution

MPPA MANYCORE

Powerful, Low Power and Programmable Processors

MPPA CORE

Core IP for programmable processor array based systems.

MPPA ACCESSCORE

C/C++ based Software Development Kit for massively parallel programing

MPPA DEVELOPER

Development platform
Reference Design Board

MPPA BOARD

Reference Design board
Application specific boards
Multi-MPPA® or Single-MPPA® boa

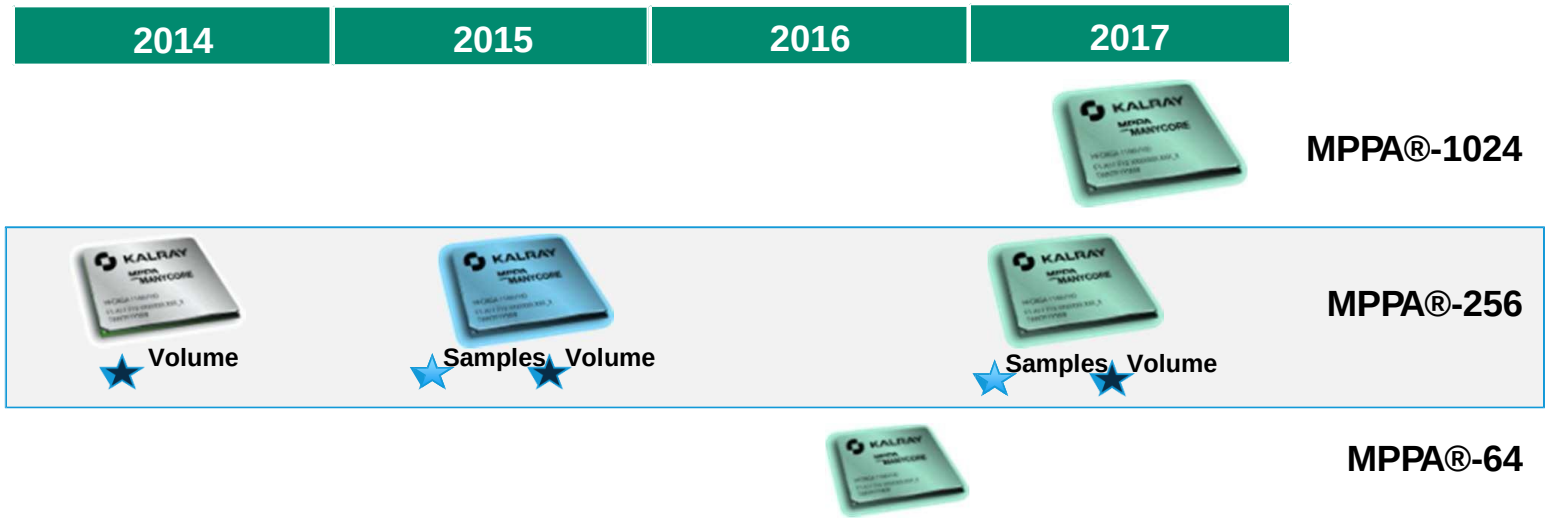
MPPA®-256 Andey Processor in CMOS 28nm

400MHz – 211 SP GFLOPS – 70 DP FLOPS – 8W to 15W

- High processing performance
- Very low power consumption
- High predictability & low latency
- High-level software development
- Architecture and software scalability

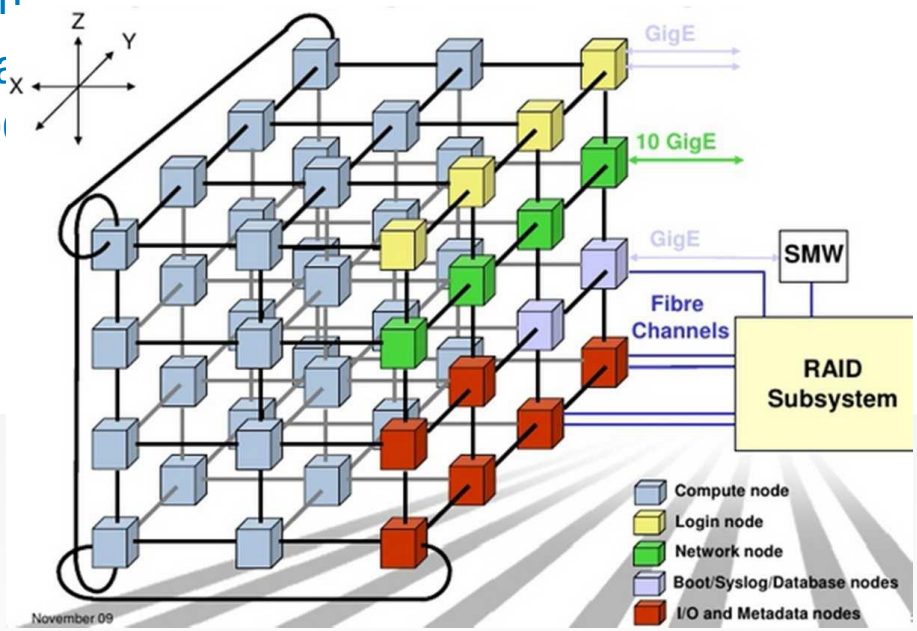
Production since 2013 in TSMC

Kalray MPPA® Processor Roadmap



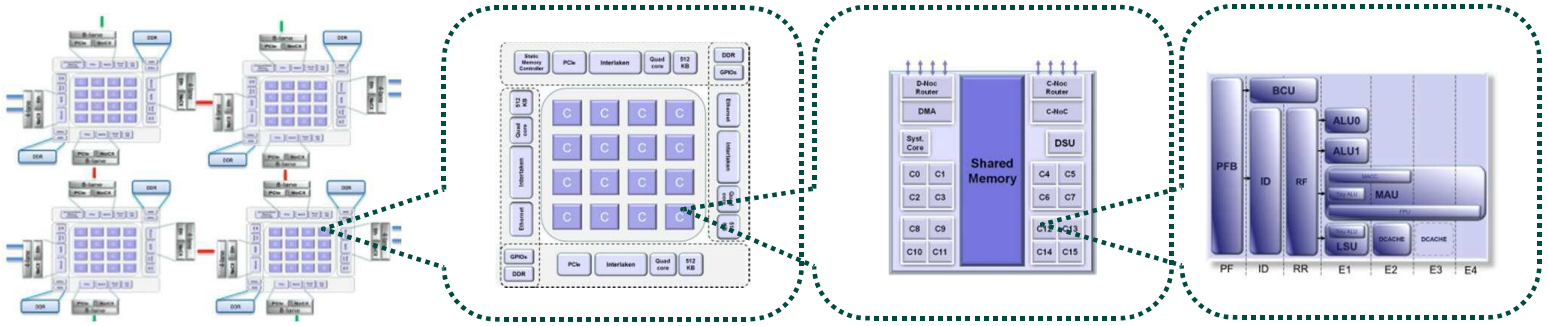
Supercomputer Clustered Architecture

- IBM BlueGene series, Cray XT series
  - Compute nodes with multiple cores and shared memory
  - I/O nodes with high-speed devices and a Linux operating system
  - Dedica and the I/O no



Quad MPPA®-256 TURBOCARD Architecture

1024 Kalray application cores and 8 DDR3-1600 under 80W



TURBOCARD Board

- 4 MPPA-256 processors
- 8 DDR3 SODIM
- PCIe Gen3 16x to host
- NoCX between MPPA

Manycore Processor

- 16 compute tiles
- 4 I/O tiles with quad-core SMP and DDR memory
- 2 Networks-on-Chip (NoC)

Compute Tile

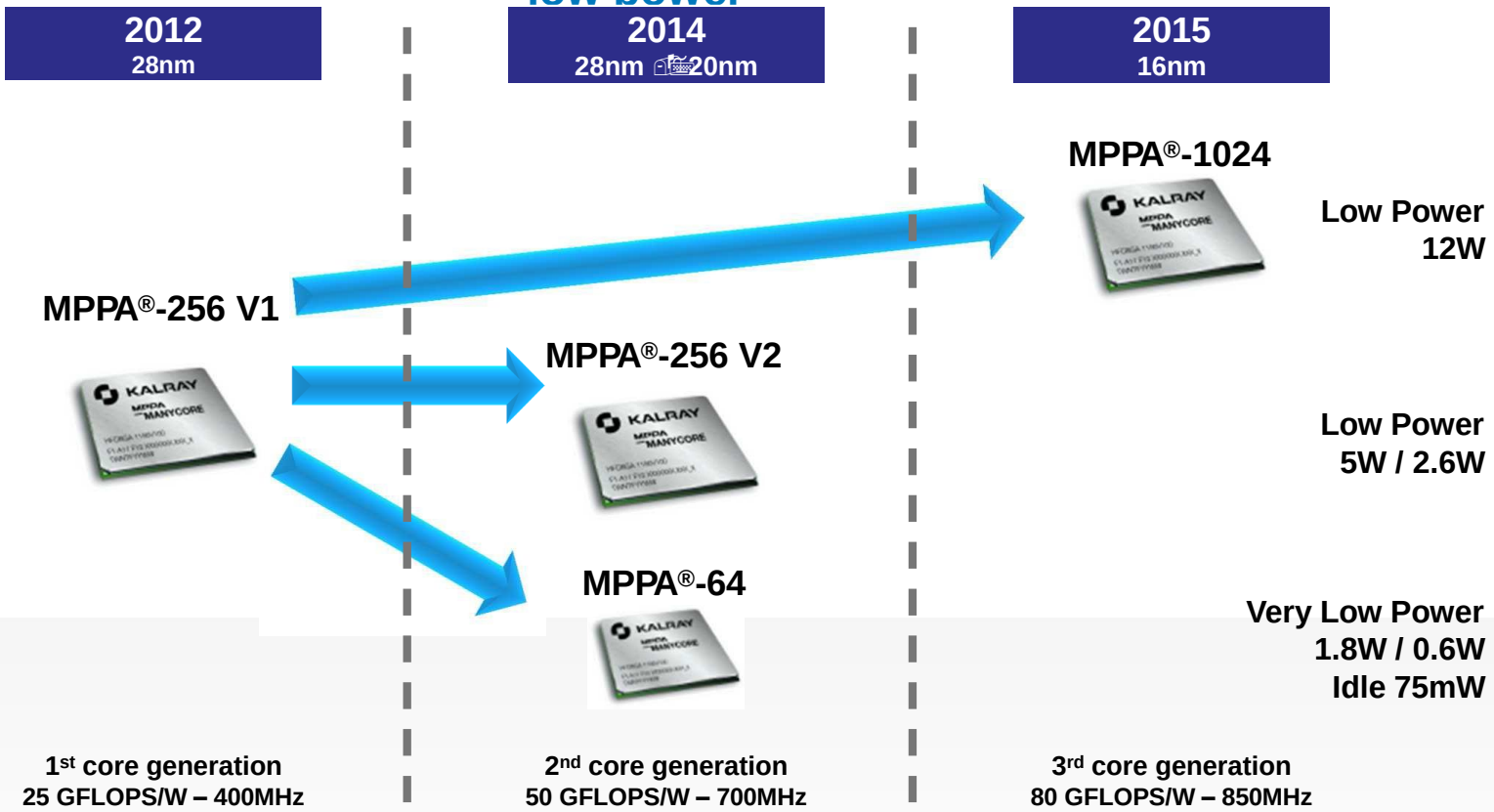
- 16 PE cores + 1 RM core
- NoC Tx and Rx interfaces
- Debug Support Unit (DSU)
- 2 MB of shared memory

VLIW Core

- 5-issue VLIW architecture
- Fully timing compositional
- 32-bit/64-bit IEEE 754 FPU
- MMU for rich OS support

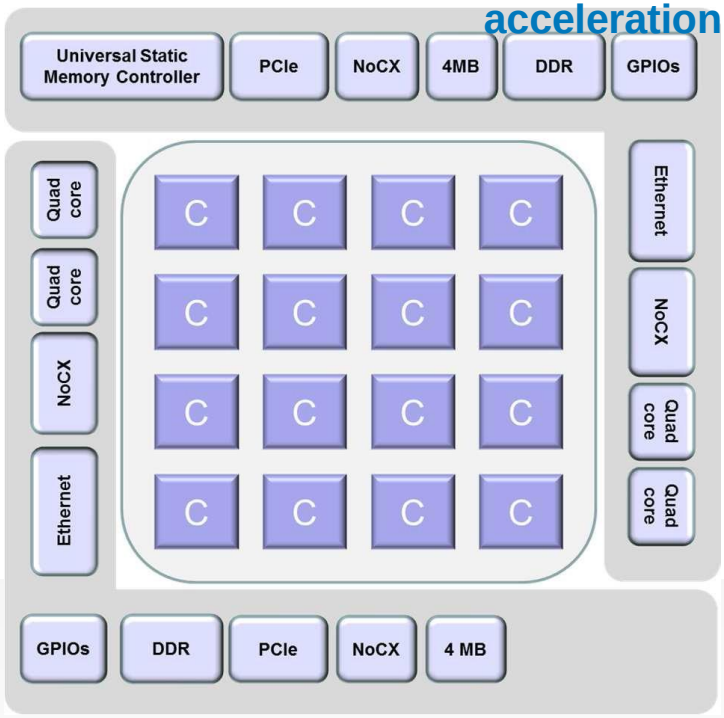
MPPA MANYCORE Roadmap

Architecture scalability for high performances and low power



MPPA®-256 Bostan Processor in CMOS 28nm

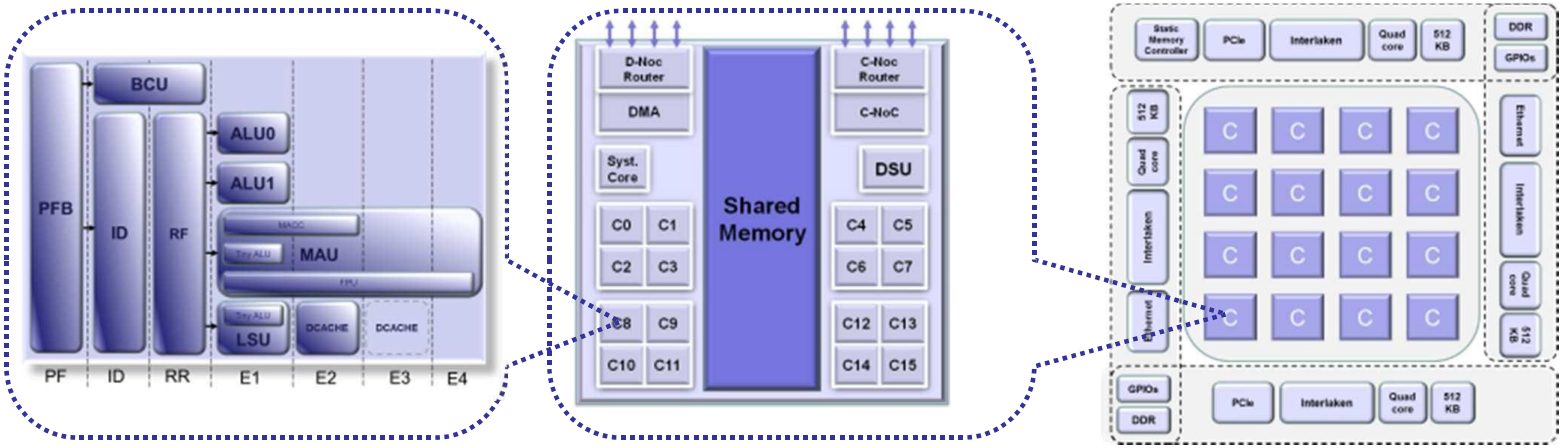
64-bit VLIW cores –Ethernet + PCIe subsystem – crypto acceleration



- 288 64-bit VLIW cores
  - **600MHz / 800MHz (overdrive)**
- Floating-point performance
  - **Simple** 845 GFLOPs @800 MHz
  - **Double** 422 GFLOPs @800 MHz
- Network on Chip (NoC) bandwidth
  - **East West** 2x 12,8 GB/s @800 MHz
  - **North South** 2x 12,8 GB/s @800 MHz
  - **Bits** 2 x 40 Gbps
  - **Packets** 2 x 50 Mpps @ 64B
- Ethernet processing throughput
- Mission critical support
  - **Time predictability**
  - **Functional safety**

MPPA®-256 Processor Hierarchical Architecture

VLIW Core | Compute Cluster | Manycore Processor

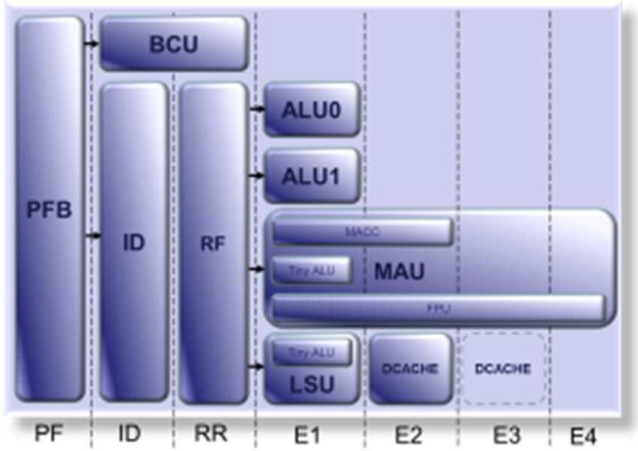


Instruction Level Parallelism

Thread Level Parallelism

Process Level Parallelism

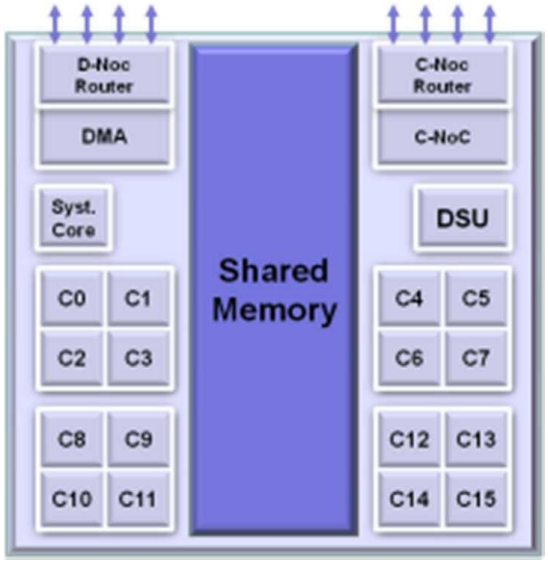
MPPA®-256 VLIW Core Architecture



- 5-issue VLIW architecture
- Predictability & energy efficiency
- 32-bit/64-bit IEEE 754 FPU
- MMU for rich OS support

- Data processing code
  - Byte alignment for all memory accesses
  - Standard & effective FPU with FMA
  - Configurable bitwise logic unit
  - Hardware looping
- System & control code
  - MMU single memory port no function unit clustering
- Execution predictability
  - Fully timing compositional core
  - LRU caches, low miss penalty
- Energy and area efficiency
  - 7-stage instruction pipeline, 400MHz
  - Idle modes and wake-up on interrupt

MPPA®-256 Compute Cluster

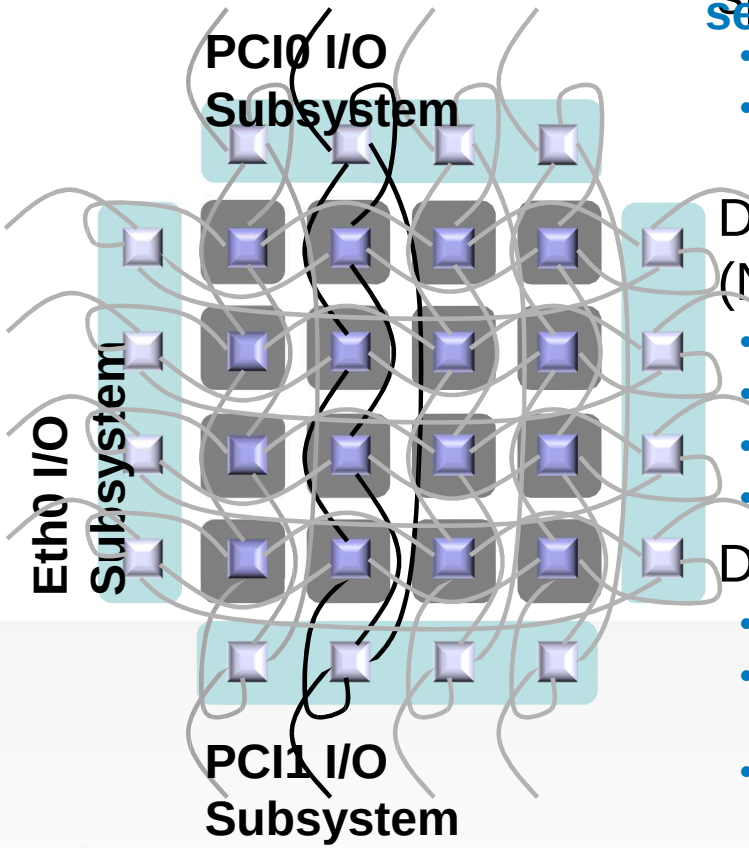


- 16 PE cores + 1 RM core
- NoC Tx and Rx interfaces
- Debug Support Unit (DSU)
- 2 MB of shared memory

- Multi-banked parallel memory
  - 16 banks with independent arbitrer
  - 38,4GB/s of bandwidth @400MHz
- Reliability
  - ECC in the shared memory
  - Parity check in the caches
  - Faulty cores can be switched off
- Predictability
  - Multi-banked address mapping either interleaved (64B) or blocked (128KB)
- Low power
  - Memory banks with low power mode
  - Voltage scaling

MPPA®-256 Clustered Memory Architecture

Explicitly addressed NoC with AFDX-like guaranteed services



- 20 memory address spaces

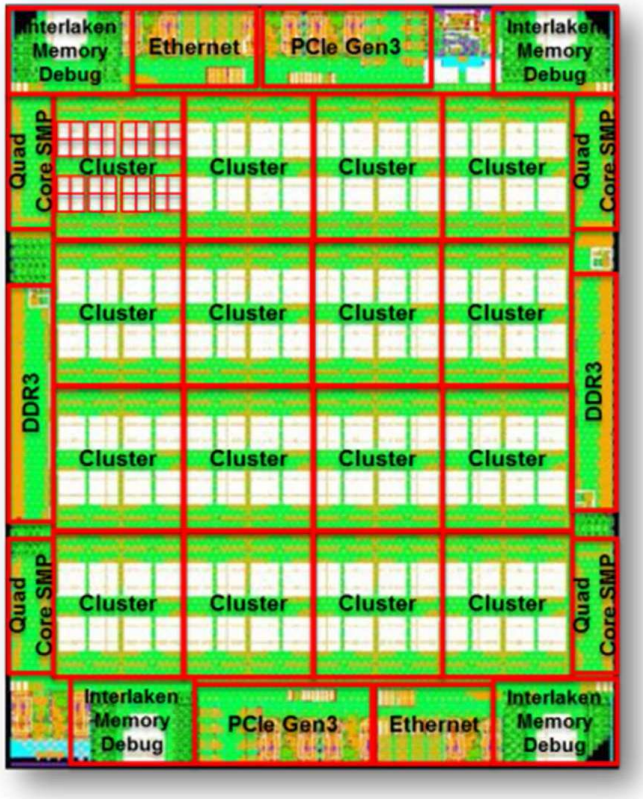
Dual Network-on-Chip (NoC)

- Data NoC & Control NoC
- Full duplex links, 4B/cycle
- 2D torus topology + extension links
- Unicast and multicast transfers

Data NoC QoS

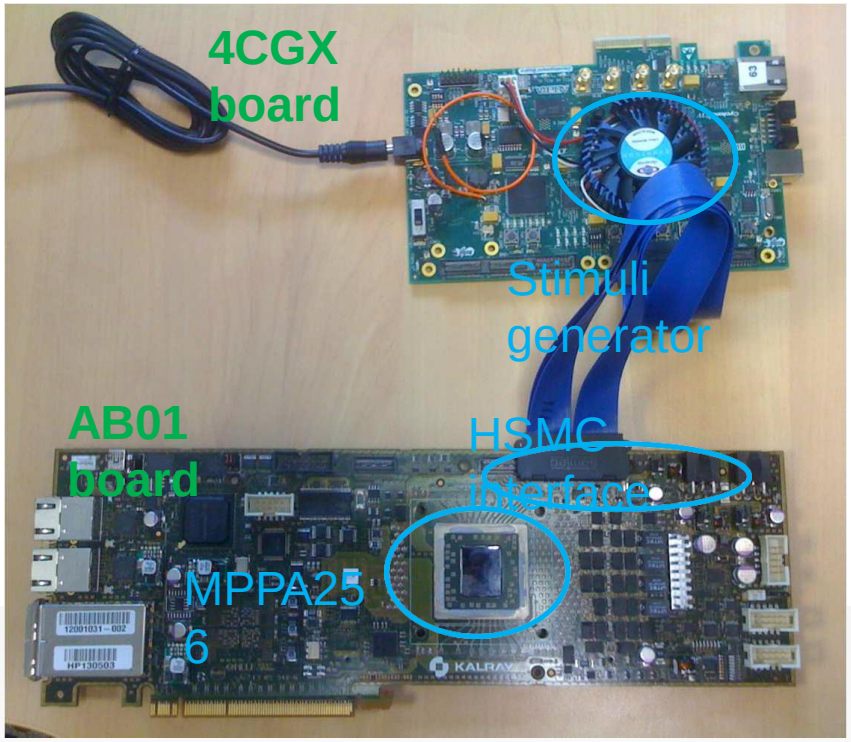
- Flow control and routing at source
- Guaranteed services by application of network calculus
- Oblivious synchronization

MPPA®-256 Processor I/O Interfaces



- DDR3 Memory interfaces
- PCIe Gen3 interface
- 1G/10G/40G Ethernet interfaces
- SPI/I2C/UART interfaces
- Universal Static Memory Controller (NAND/NOR/SRAM)
- GPIOs with Direct NoC Access (DNA) mode
- NoC extension through Interlaken interface (NoC Express)

MPPA®-256 Direct NoC Access (DNA)

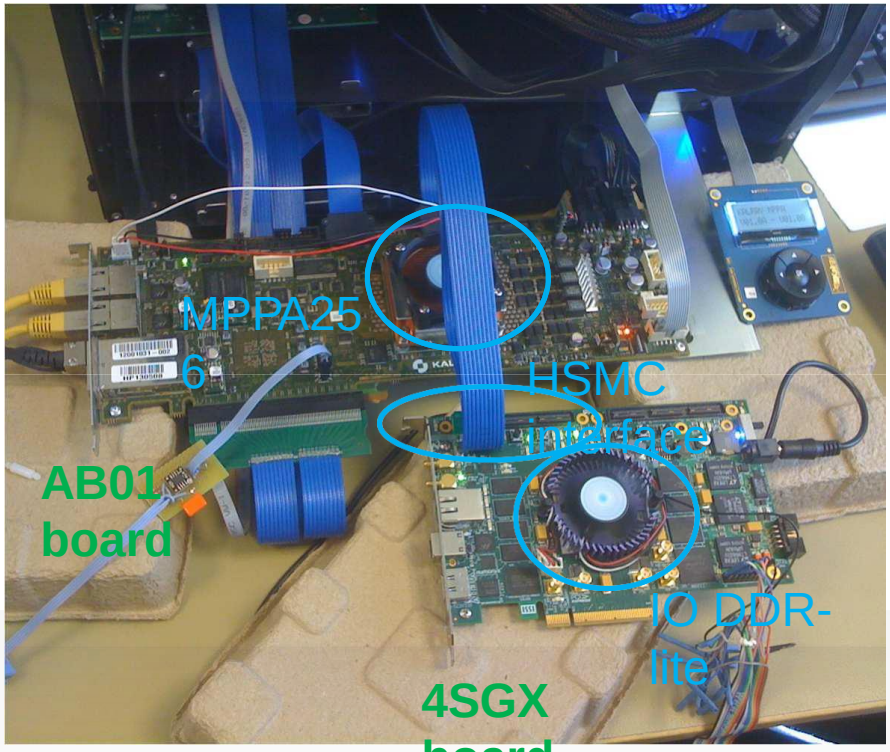


NoC connection to GPIO
- Full-duplex bus on 8/16/24 bits + notification + ready + full bits
- Maximum 600 MB/s @ 200 MHz
- Direct to the GPIO 1.8V pins
- Indirect through low-cost FPGA

Data sourcing
- Input directed to a Tx packet shaper on I/O subsystem 1
- Sequential Data NoC Tx configuration

Data processing
- Standard data NoC Rx configuration
  - Application flow control to GPIO
  - Input data decounting
  - Communication by sampling

MPPA®-256 Sample Use of NoC Extensions (NoCX)



Mapping of IO DDR-lite on FPGA

- Altera 4SGX530 development board
- Interlaken sub-system (x3 lanes up to 2.5-Gbit/sec)
- 300MHz DDR3
- x1 RM + x1 DMA + 512-Kbyte SMEM @ 62.5MHz
- Single NoC plug

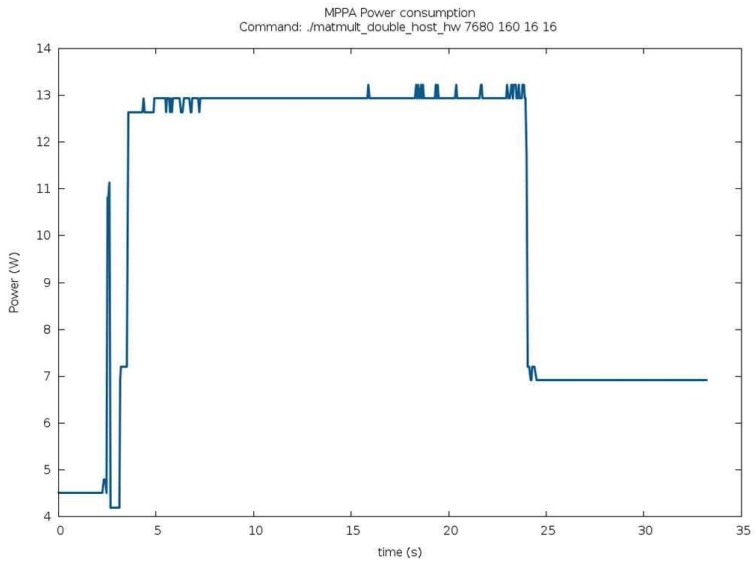
4K video through HDMI emitters

- Interlaken configured in Rx & Tx
- 1.6-Gbit/sec effective data NoC bandwidth reached
  - Limiting factor is the FPGA device internal frequency
  - Effective = 62.5MHz * 32-bit * 80%
- Output of uncompressed 1080p video @ 60-frame/sec

Measuring Power and Energy Consumption

- k1-power tool for power measurement crom command shell

```
$ k1-power -h
-o oFILE        output file
-f FORMAT       plot in FORMAT
-s              standalone mode
-p              plot double
$ k1-power -o plot_double -gpdf \
-- ./matmult_double_host_hw \
7680 160 16 16
...
Time        : 33.20 s
Power (avg) : 10.45 W
Energy      : 346.89 J
...
$ display plot_double.pdf
$ k1-power
```



- libk1power.so shared library, control measure from user code
  - `int k1_measure_callback_function(int (*cb_function) (float time, double power));`
  - `int k1_measure_start(const char *output_filename);`
  - `int k1_measure_stop(k1_measure_t *measures);`

```
typedef struct {
  float time;
  double power;
  float energy;
} k1_measure_t;
```

KALRAY

Manycore Technology Comparison

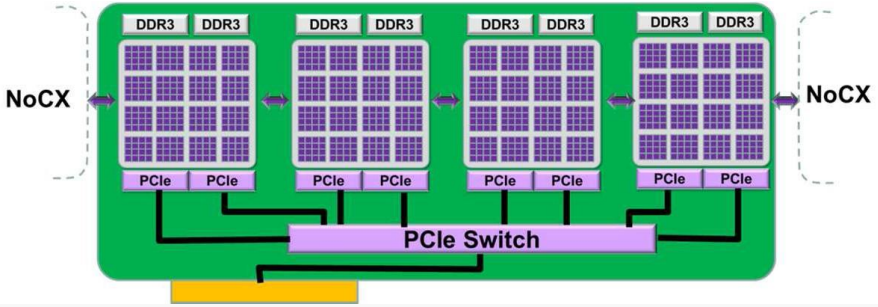
	Cores	GFLOPS (SP)	Active Power	Real Time	DDR	Ethernet
Intel Xeon Phi	52 x86	2147	300W	No	GDDR 5 1866	No
Tilera TileGx	72	80	60W	No	4 DDR3 1600	8 10G
NVIDIA Tegra4	4 A15 72 SC	45 75	8W	No	2 DDR3 1866	1G
TI KeystoneII	4 A15 8 C66x	45 154	25W	Yes	2 DDR3 1600	10G
Kalray MPPA	288 K1	230	10W	Yes	2 DDR3 1600	8 10G

Kalray TURBOCARD Family

Kalray TURBOCARD2 (2014)

- 4 Andey MPPA®-256 processors
- 0.83 TFLOPS SP / 0.28 TFLOPS DP
- 8x DDR3L @1233 MT/s => 79 GB/s

- First engineering samples: Q4-14
- Volume Production: Q1-15



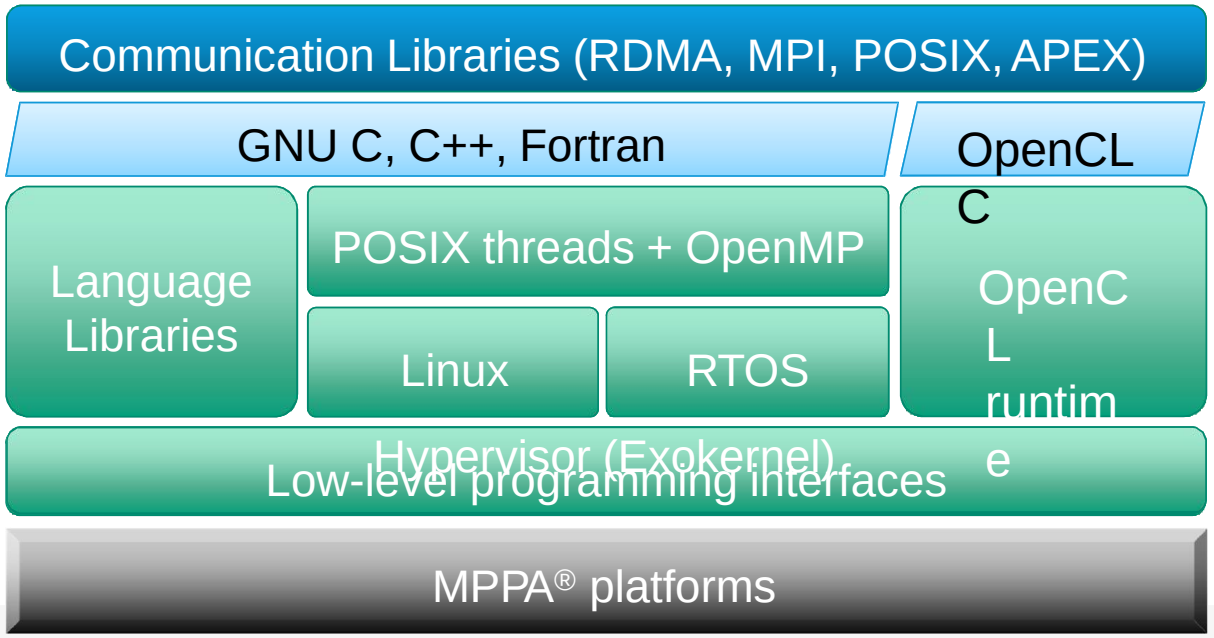
Kalray TURBOCARD3 (2015)

- 4 Bostan MPPA®-256 processors
- 2.5 TFLOPS SP / 1.25 TFLOPS DP
- 8x DDR3 @ 2133 MT/s => 136 GB/s

- First engineering samples: Q2-15
- Volume Production: Q3-15

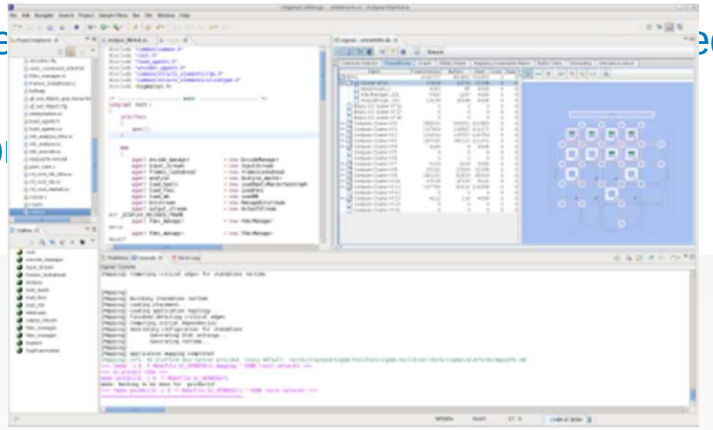
KALRAY

MPPA® Software Stack



Standard C/C++ Programming Environment

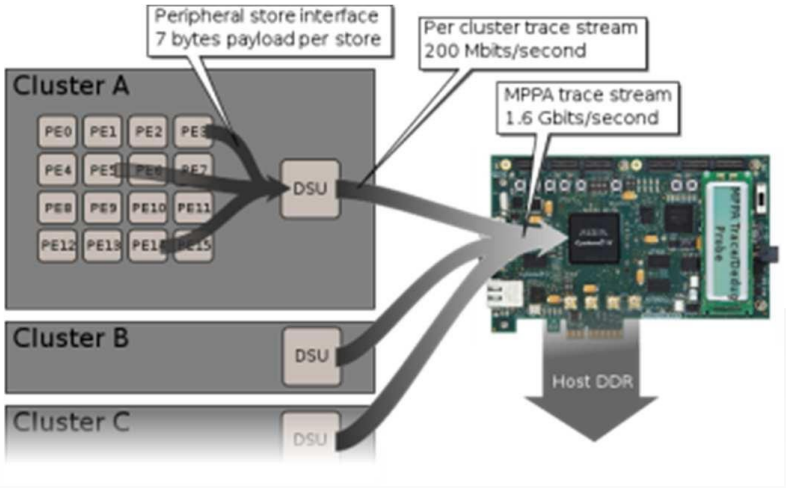
- Eclipse Based IDE and Linux-style command line
- Full GNU C/C++ development tools for the Kalray VLIW core
  - GCC 4.7 (GNU Compiler Collection), with C89, C99, C++ and Fortran
  - GNU Binary utilities 2011 (assembler, linker, objdump, objcopy, etc.)
  - GDB (GNU De ed code debugging
  - Standard C lib functions



Simulators, Debuggers & System Trace

- Platform simulators
  - Cycle-accurate, 400KHz per core
  - Software trace visualization tool
  - Performance view using standard Linux kcachegrind & wireshark
- Platform debuggers
  - GDB-based, follow all the cores
  - Debug routines of each core activate the tap controller for JTAG output
- System trace acquisition
  - Each cluster DSU is able to generate 200Mb/s of trace
  - 8 simultaneous observable trace flows out of 20 sources

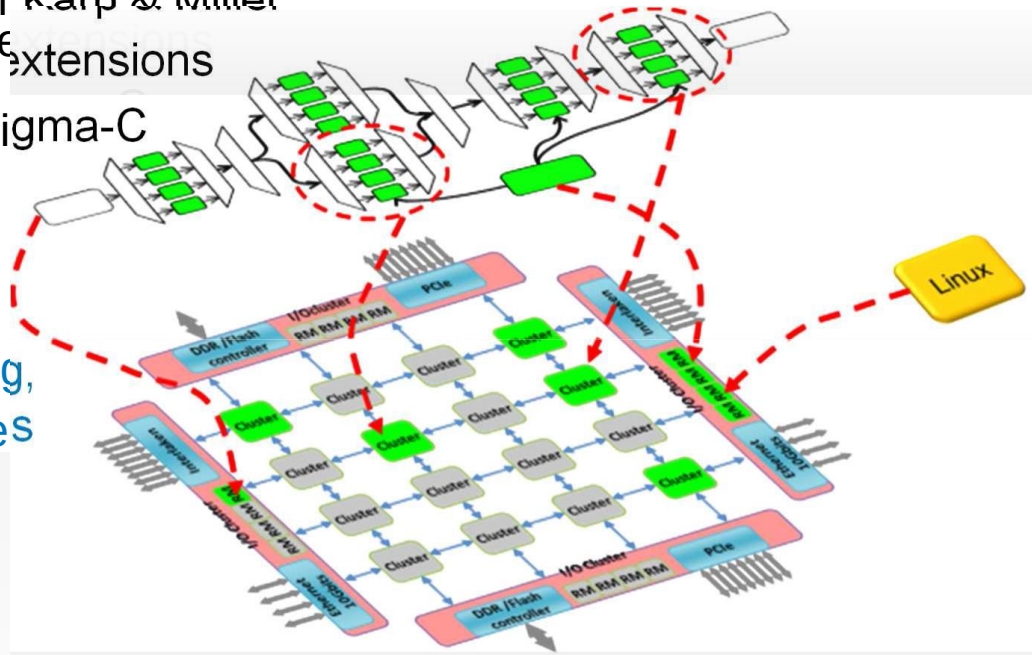
- System trace display
  - Based on Linux Trace Toolkit NG (low latency, on demand activation)
  - System trace viewer (customized view per programming model)



Dataflow Programming Environment

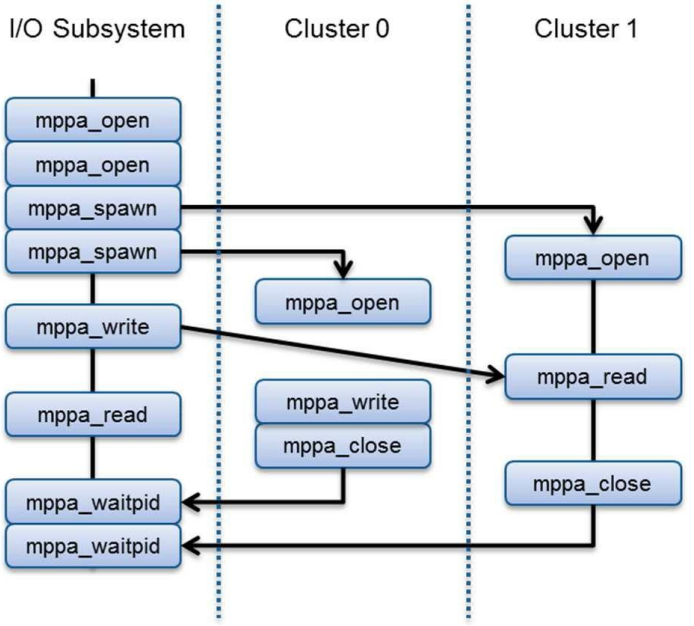
- Computation blocks and communication graph written in C
- Cyclostatic data production & consumption
- Firing thresholds of Karp & Miller
- Dynamic dataflow extensions
- Language called Sigma-C

Automatic mapping on MPPA® memory, computing & communication resources



POSIX-Level Programming Environment

- POSIX-like process management
  - Spawn 16 processes from the I/O subsystem
  - Process execution on the 16 clusters start with main(argc, argv) and environment
- Inter Process Communication (IPC)
  - POSIX file descriptor operations on 'NoC Connectors'
  - Inspired by supercomputer communication and synchronization primitives
- Multi-threading inside clusters
  - Standard GCC/G++ OpenMP support
    - #pragma for thread-level parallelism
    - Compiler automatically creates threads
  - POSIX threads interface
    - Explicit thread-level parallelism



KALRAY

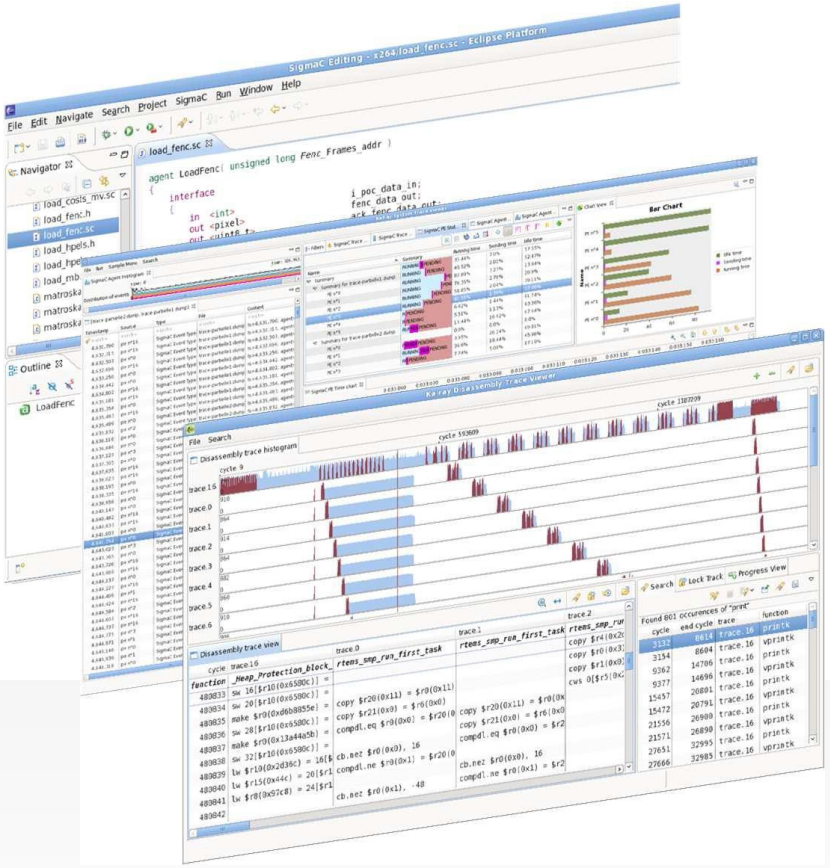
POSIX IPC with NoC Connectors

- Build on the 'pipe & filters' software component model
  - Processes are the atomic software components

Connector	Purpose	Tx:Rx Endpoints	Resources
Sync	Half synchronization barrier	N:1, N:M (multicast)	CNoC
Portal	Remote memory window	N:1, N:M (multicast)	DNoC
Sampler	Remote circular buffer	1:1, 1:M (multicast)	DNoC
RQueue	Remote atomic enqueue	N:1	DNoC+CNoC
Channel	Zero-copy rendez-vous	1:1	DNoC+CNoC

  - Synchronous operations: open(), close(), ioctl(), read(), write(), pwrite()
- Asynchronous I/O operations on Portal, Sampler, RQueue
  - Based on aio_read(), aio_error(), aio_return()
  - NoC Tx DMA engine activated by aio_write()

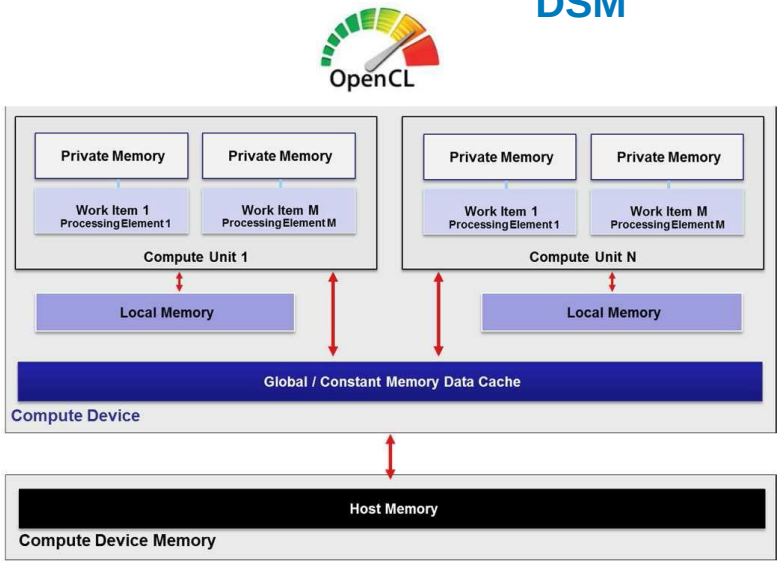
Compilers, Debuggers & System Trace

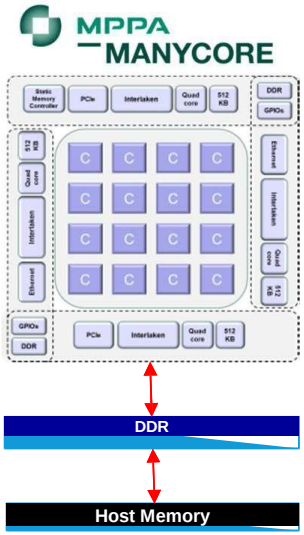


- Compilers
  - GCC version 4.9
  - C89, C99, C++ and Fortran
  - GNU Binary utilities 2011 (assembler, linker, objdump, objcopy, etc.)
  - C libraries: uClibc, Newlib + F.P. tuning
  - Linear Assembly Optimizer (LAO) for VLIW Optimizations
- Debuggers
  - GDB version 7.3
  - Each core seen as a thread
  - Watchpoint & breakpoint
- System trace
  - Linux Trace Toolkit NG based
    - Low-latency trace points
    - On-demand activation
  - Eclipse-based system trace viewer
  - Deep insight of execution at system level

OpenCL Programming Environment

MPPA® OpenCL global memory implemented as software DSM





- Processing Element
- Compute Unit
- Compute Device

→

- VLIW Core
- Compute Tile
- MPPA Processor

MPPA® OpenCL Roadmap

- Mega OpenCL Device
  - The MPPA® OpenCL global memory is implemented by software DSM
  - This implementation allows to pool several DDR memory controllers
  - A single OpenCL device can be composed of several MPPA® processors connected by their NoC extensions
  - For the Kalray TURBOCARD with 4 MPPA® processors, the OpenCL device will offer 1K VLIW cores and 64GB of DDR global memory

- Time-Critical OpenCL
  - The OpenCL host API can be hosted on one I/O subsystem quad-core
  - The OpenCL kernels can be dispatched on the same MPPA® processor
  - The OpenCL run-time can be implemented with controlled worst-cases

KALRAY

Streaming Programming Environment

- MPPA® support of OpenCL
  - Task parallel model: one kernel per compute cluster
    - Native kernel mode: clEnqueueNativeKernel()
    - Standard task parallel mode: clEnqueueTask()
  - Emulate global memory with Distributed Shared Memory (DSM)
    - Use the MMU on each core, assume no false sharing
    - Use the MMU on each core, resolve false sharing like Treadmarks

- MPPA® Bulk Synchronous Streaming
  - Adapt Bulk Synchronous Parallel (BSP) model to the MPPA®
    - Execute a number of cluster processes > number of clusters
    - Double buffering to overlap cluster process execution and swapping

KALRAY

OpenCL Programming Environment

- Standard OpenCL has two programming models
  - Data parallel, with one work item per processing element (core)
  - Task parallel, with one work item per compute unit (multiprocessor)
    - In native kernels, may use standard C/C++ static compiler (GCC)
- MPPA® support of OpenCL
  - Task parallel model: one kernel per compute cluster
    - Native kernel mode: clEnqueueNativeKernel()
    - Standard task parallel mode: clEnqueueTask()
  - Emulate global memory with Distributed Shared Memory (DSM)
    - Use the MMU on each core, assume no false sharing
    - Use the MMU on each core, resolve false sharing like Treadmarks

KALRAY

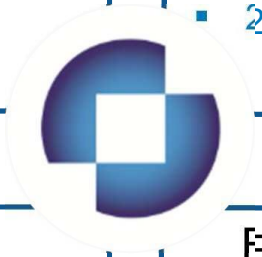
MPPA® Manycore Computing

Energy Efficiency

- 5-issue 32-bit / 64-bit VLIW core
- Optimum instruction pipelining
- Shallow memory hierarchy
- Software cache coherence

Performance Scalability

- Linear scaling with number of cores
- Network on chip extension (NoCX)
- 2x DDR controllers per processor
- 24x 10 Gb/s lanes per processor



Execution Predictability

- Fully timing compositional cores
- Multi-banked parallel local memory
- Core-private busses to local memory
- Network on chip guaranteed services
- Configurable DDR address mapping

Ease of Use

- Standard GCC C/C++/Fortran
- Command-line and Eclipse IDE
- Full featured debug environment
- System trace based on LTTNG
- Linux + RTOS operating systems

Parallel Programming Model for the Epiphany Many-Core Coprocessor Using Threaded MPI

June 13th, 2015

James Ross

Engility Corporation, MD, USA

David Richie[1], Song Park[2], and Dale Shires[2]

[1]Brown Deer Technology, MD, USA

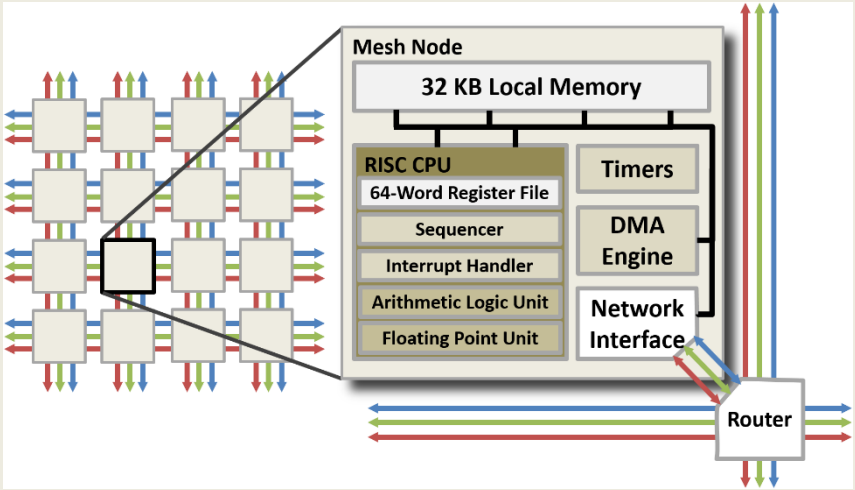
[2]U.S. Army Research Laboratory, MD, USA

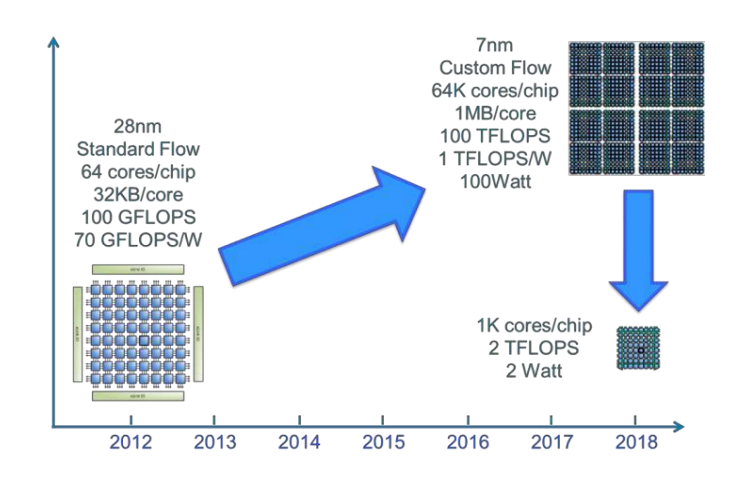
Outline

- Epiphany architecture
- Programming challenge
- Threaded MPI (Message Passing Interface)
- Hardware/software setup
- Benchmark performance
- Conclusions & future work

Adapteva Epiphany Architecture

- Design emphasizes simplicity, scalability, power-efficiency
- 2D array of RISC cores, 32KB per core, 2D Network on Chip (NoC)
- Fully divergent cores with distributed local memory
- Minimal un-core functionality, e.g., no data or instruction cache
- Existing design scales to thousands of cores
- Highest performance/power efficiency to date, ~50 GFLOPS/W





Programming Challenge

- **Programming challenge:**
  - **Power-efficiency achieved by simplifying the architecture**
  - **Distributed memory mapped cores with 32KB local memory per core**
  - **Non-uniform memory access (NUMA) to mapped local memory**
  - **Typical many-core programming models (OpenMP, OpenCL, CUDA) leverage common cache / memory hierarchy for SMP**
  - **Here, there is no hardware data/instruction cache**
  - **Best viewed as a "distributed cluster on chip"**
- **Additional challenges:**
  - **Device employed as co-processor – requires offload semantics**
  - **Resource constraints prevent running full process image**
    - **Contrast with MIC which runs Linux on each core**
  - **No hardware support for message passing (or much of anything else), so *all solutions must be in software***

Epiphany + MPI

- Two key observations:
  - Architecture resembles cluster on chip
  - Inter-core data movement is key to performance
- Proposition: re-purpose MPI as device-level programming model

- Conventional MPI:
  - Execute multiple process images
  - Inter-process data movement via messages passing

- Threaded MPI:
  - Offload multiple light-weight threads
  - Inter-thread data movement via messages passing

- *We are using MPI syntax and semantics for programming the device-level parallelism of a 2D RISC array co-processor*

Threaded MPI: Design

Design objectives:

- Maintain precise MPI syntax/semantics with minimal restriction
- Design must lead to efficient implementation for architecture
- Pragmatic initial design, focus on essential calls

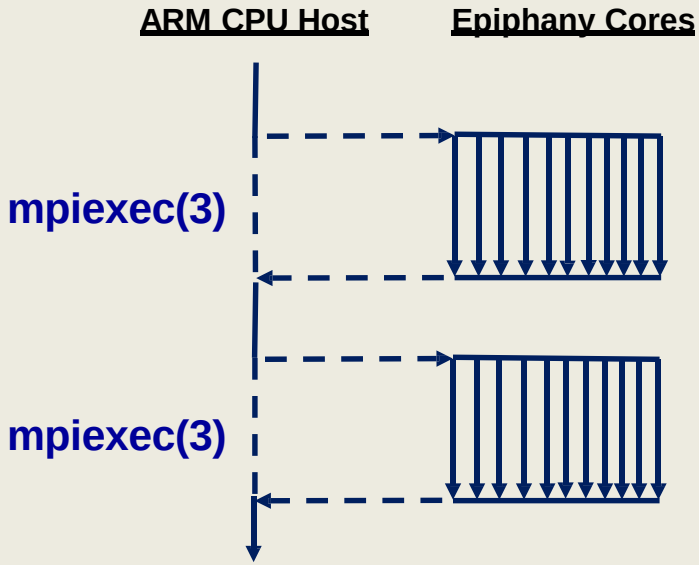
Design challenges:

- Device integrated as co-processor requiring offload semantics
  - Conventional MPI executes full process image in parallel
  - Epiphany cores do not execute full process image
- Significant local memory constraint (32KB per core)
  - Conventional MPI relies on large buffers (megabytes)
  - Epiphany cores must use buffers > 1000x smaller

Threaded MPI: Design

mpiexec(3)

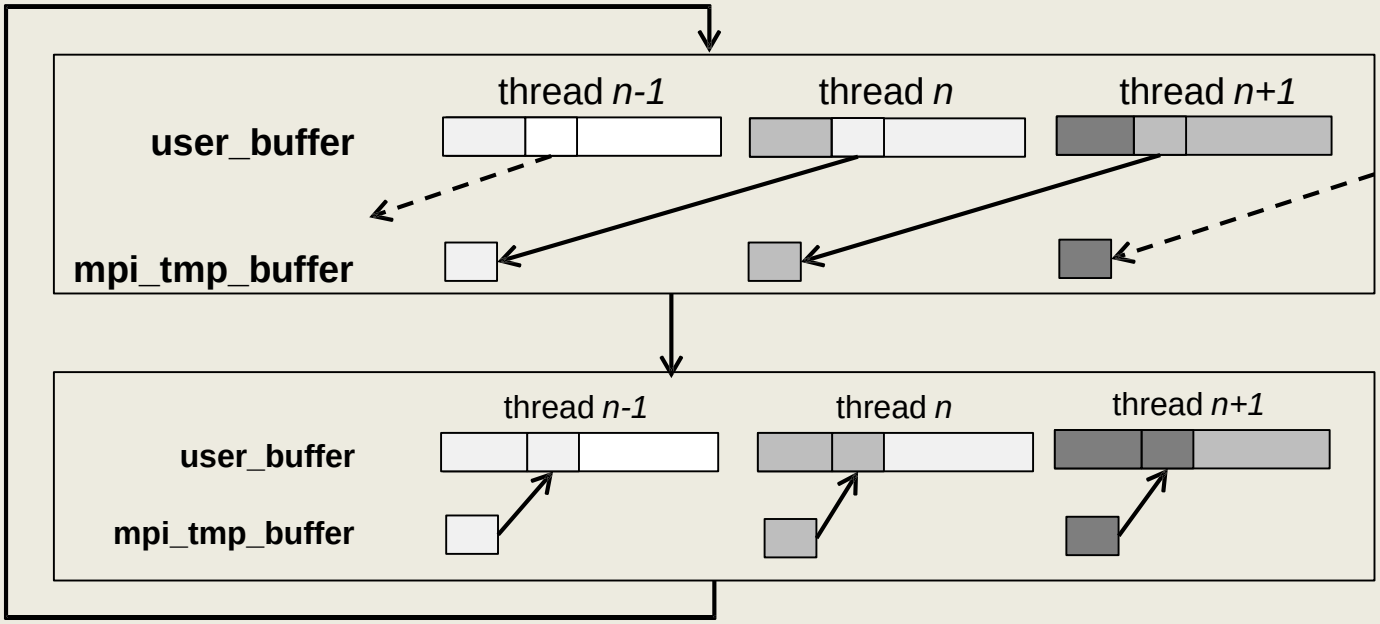
- Analogous to conventional mpiexec command
- Launches multiple threads on co-processor as opposed to multiple processes on nodes
- Hybrid between accelerator offload and conventional MPI program execution model
- Can be called multiple times from single host application
- Fork-join model, implemented with coprthr_ncreate(3)



Threaded MPI: Design

MPI_Sendrecv_replace(3)

- Implemented with a small MPI temporary buffer (< 1KB)
- Messages transparently broken up into multiple transactions
- Sending thread performs DMA write to local memory of receiving thread (architecture preference is write over read)
- MPI temporary buffer size is tunable



Repeat until user_buffer completely transferred

Threaded MPI: Implementation

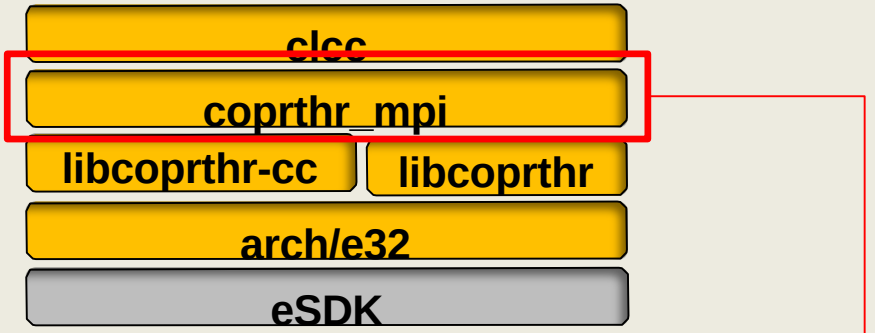
- First calls implemented (at the time of this work):

MPI Call	Comments
MPI_Init	Initialization, called once per thread
MPI_Finalize	Finalization, called once per thread
MPI_Comm_size	Get number of threads in the group
MPI_Comm_rank	Get rank of the current thread in the group
MPI_Cart_create	Create 1D and 2D communicator topologies
MPI_Comm_free	Free communicator
MPI_Cart_coords	Determines thread coordinates in topology
MPI_Cart_shift	Returns the shifted source and destination ranks in topology
MPI_Send	Blocking send, zero-copy implementation
MPI_Recv	Blocking receive, zero-copy implementation
MPI_Sendrecv_replace	Blocking send/ receive/replace, buffered implementation
MPI_Bcast	Broadcast, zero-copy implementation

- Practical application development
  - Port MPI algorithms to thread functions
  - Account for Pthread-style argument passing
  - Create host program to manage co-processor, shared memory and thread execution via mpiexec(3) calls

Hardware/Software Setup

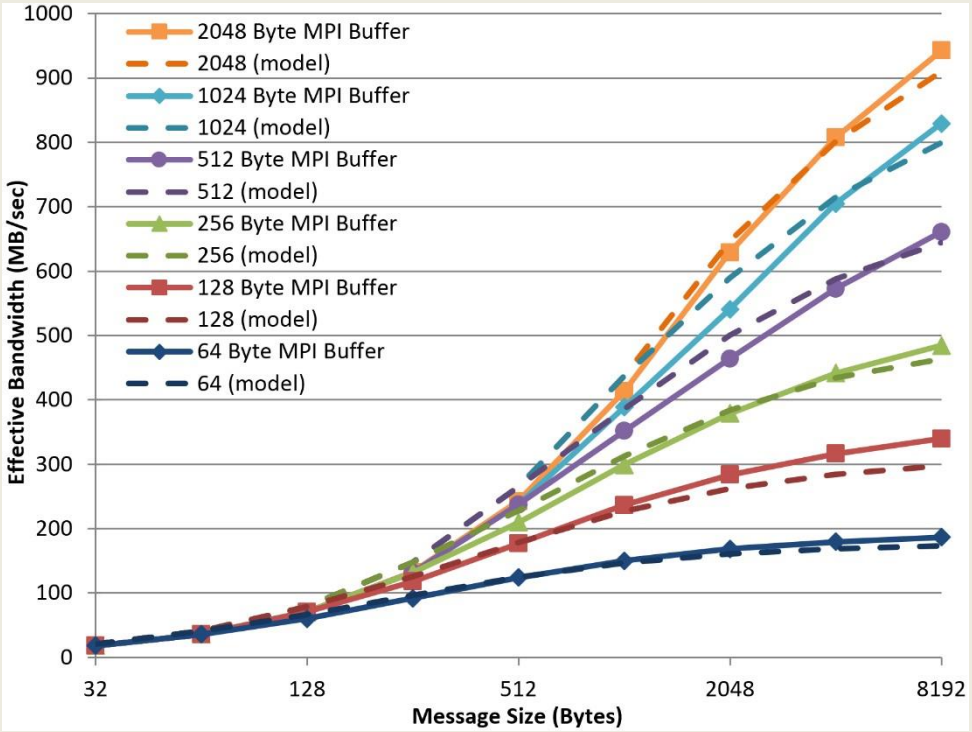
- Parallella board dev kit
- Dual-core ARM host processor
- 16-core Epiphany III co-processor
- 19.2 GFLOPS @ 600 MHz



- COPRTHR software stack
- Provides offload and threading support
- Includes threaded MPI

- **Goal: investigate performance and efficiency of algorithms**
- **Key objectives:**
  - **Adapt MPI-based implementations for device-level parallelism**
  - **Determine if programming model achieves good relative performance**

NoC Performance



Effective inter-core bandwidth of the MPI_Sendrecv_replace call for different MPI buffer sizes with a modified alpha-beta model:

$T = \alpha_0 + \alpha_1 \cdot k + \beta \cdot m$

α_0 **is fixed latency of the MPI call**

α_1 **is latency per buffered DMA transaction**

β^1 **is bandwidth**

m **is message size in bytes**

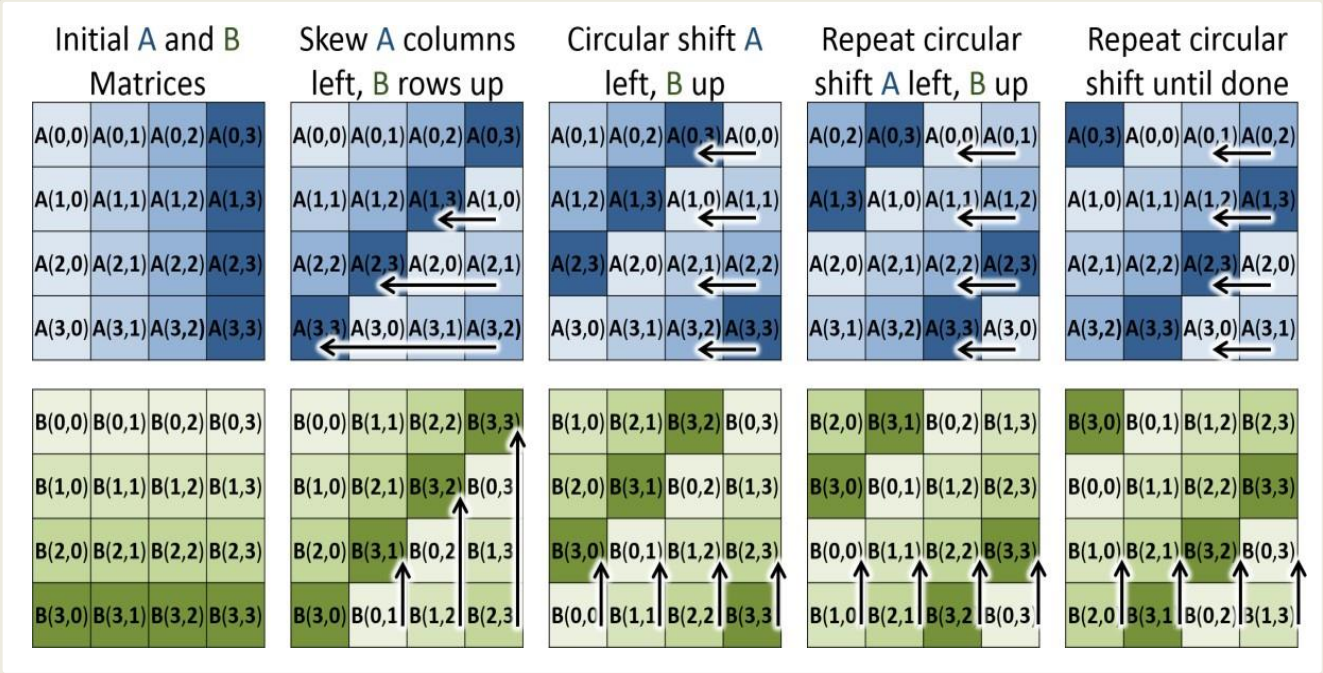
k **is the number of internal DMA transactions**

- **Key question: can good performance be achieved with small MPI buffers?**
  - **Typical MPI implementation has buffers measured in megabytes**
  - **Threaded MPI employs buffers over 1000x smaller**
- **Answer is *yes***

U.S. ARMY RDECOM

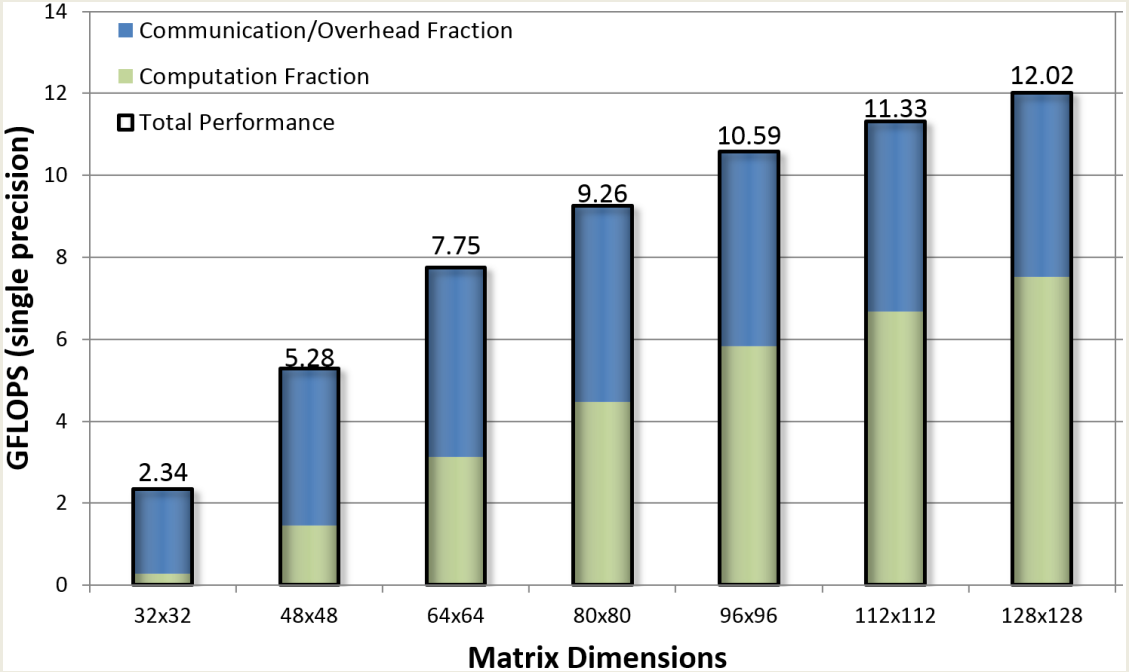
Matrix-Matrix Multiplication

ARL



- Cannon's algorithm typically used for distributed parallel cluster
  - Alternating sub-matrix multiplication and shift (2D periodic domain)
- We adapt this algorithm for programming 2D array of RISC cores
- Re-used previously developed MPI code for a distributed cluster with minimal modifications to the kernel itself (MPI code 10 years old!)
- Adapted to run within the threaded offload model
- Computation is $O(n^3)$ and Communication is $O(n^2)$

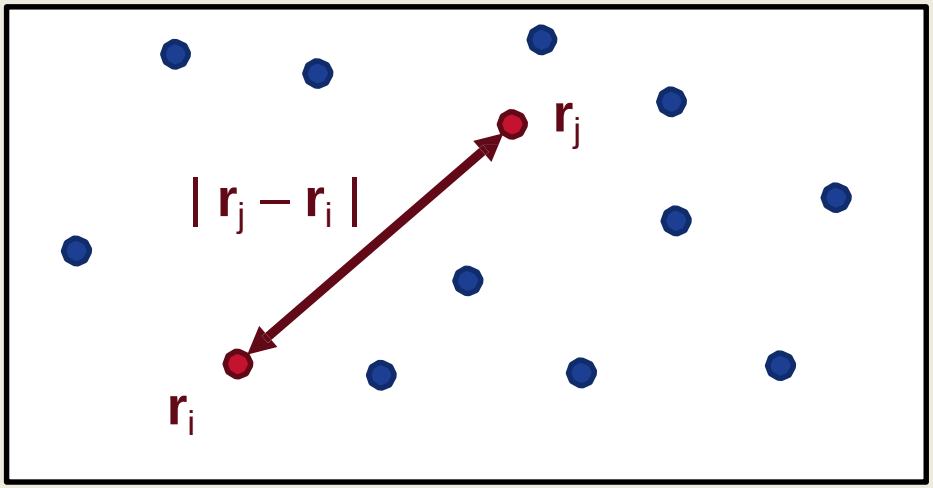
Matrix Multiply Performance Analysis



- Relative compute-communication time estimated for various matrix sizes
  - All timing is empirical using wall-clock timing on host
  - For matrix size of 128 x 128 results show a 60%-40% split
- Effective on-chip bi-directional inter-core bandwidth also measured
  - Peak inter-core bandwidth for device estimated to be ~1250 MB/sec)
  - Measured results are consistent with expected transfer bandwidth

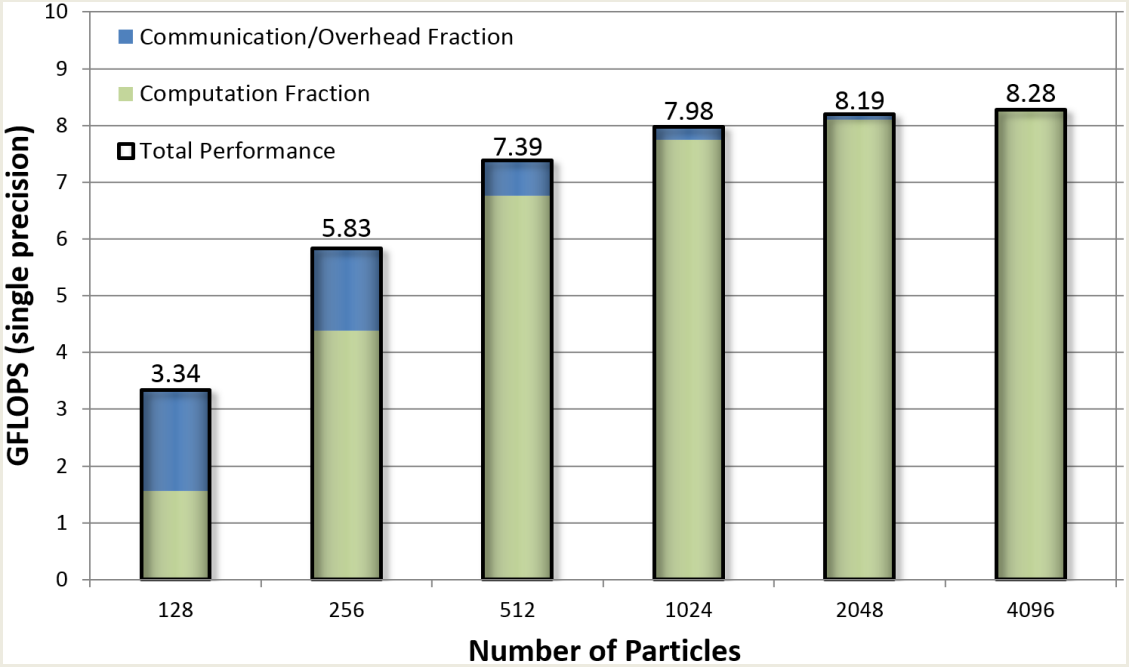
N-body Particle Interaction

$$\mathbf{f}_i = \sum m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$



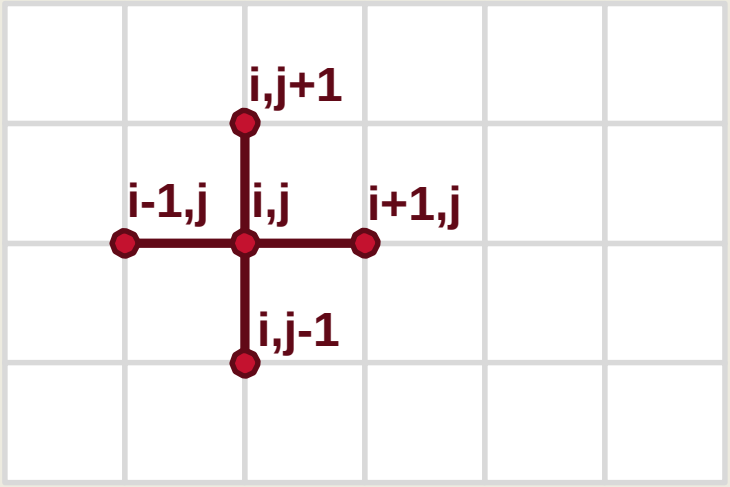
- Models motion of N particles subject to particle-particle interaction, e.g.,
  - Gravitational force
  - Charged particles
- Computation is $O(N^2)$ and Communication is $O(N)$
- Algorithm has two main steps:
  - Calculate total force on each particle
  - Update particle position/velocity over some small timestep (Newtonian dynamics)
- Entire (unoptimized) algorithm can be written in 20 lines of C code.

N-Body Performance Analysis



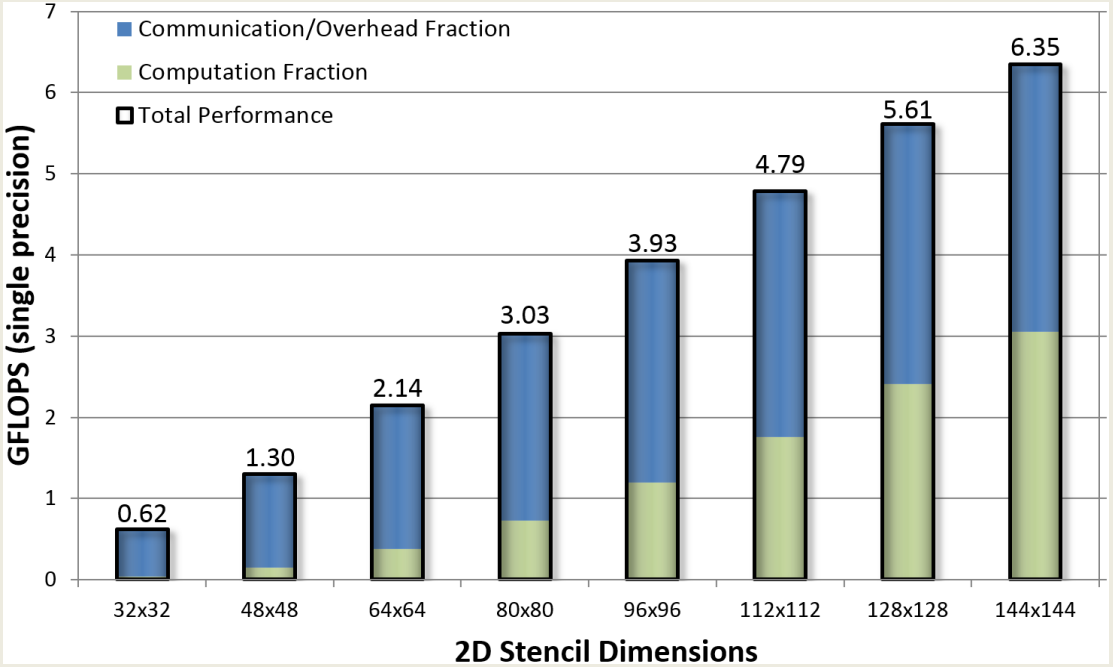
- Relative compute-communication time estimated for various work sizes
  - All timing is empirical using wall-clock timing on host
  - Large problems are completely compute-bound
- Favorable compute-communication ratio enables off-chip problem sizes with marginal performance penalty (<5%)
- 1D, periodic MPI domain decomposition

5-Point 2D Stencil



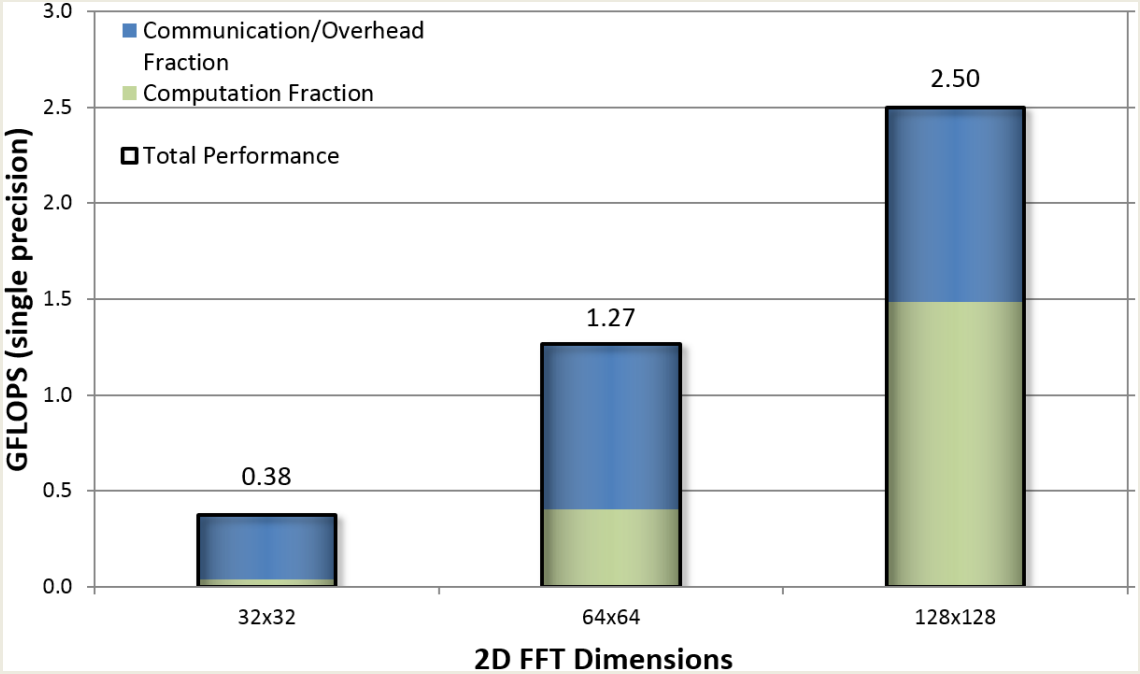
- 5-point stencil representative of explicit PDEs
- Iterative solution found using specified boundary conditions
- Shared edge values communicated to neighbors after each step
- Computation and Communication are O(n)
  - Unfavorable compute-communication ratio
- 2D MPI domain decomposition (not periodic)

Stencil Performance Analysis



- Relative compute-communication time estimated for various workloads
  - All timing is empirical using wall-clock timing on host
  - For largest stencil of 144 x 144 results approach 50%-50% split
- Effective inter-core bandwidth is low (small buffers)

2D Fast Fourier Transform



- **Cooley-Tukey algorithm: Parallelized over stripes, performs 1D radix-2 decimation in time (DIT) FFT, a corner turn, an additional DIT FFT, and a final corner turn**
- **Used conventional 2D MPI domain decomposition (not periodic)**
- **Computation scales $O(n^2 \cdot \log_2(n))$ and communication $O(n^2)$**
- **We feel 13% of peak performance is pretty good!**

Conclusions & Future Work

- Threaded MPI provides a very effective programming model for this class of architecture.
- A threaded MPI implementation adapted for coprocessor offload was developed.
- Existing MPI codes for matrix-matrix multiplication, N-body particle update, 5-point Stencil iterative update, and 2D FFT were adapted for Epiphany with minimal changes.
- Demonstrated performance 12+ GFLOPS with an efficiency of 20.2 GFLOPS/Watt (16C, 600MHz, 65nm, 594mW).
- Threaded MPI exhibits the highest performance reported using a standard parallel API.
- Future work will examine additional algorithms and MPI calls including asynchronous (non-blocking MPI_Isend/MPI_Irecv) and one-sided (MPI_Put/MPI_Get) communication.

Resources

Download the Codes:

https://github.com/USArmyResearchLab/mpi-epiphany

COPRTHR MPI Library:

http://browndeertechnology.com/code/bdt-libcoprthr_mpi-preview.tgz

COPRTHR (version 1.6.1):

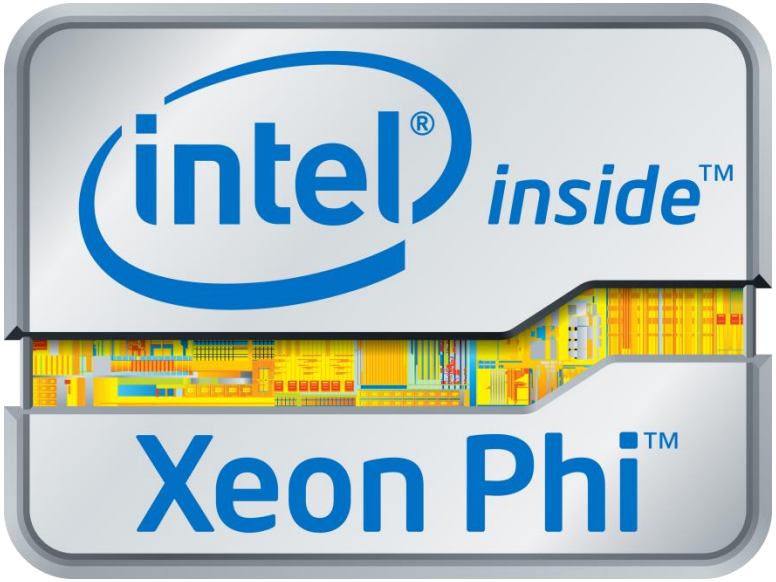
https://github.com/browndeer/coprthr

Parallella Platform:

http://parallella.org/

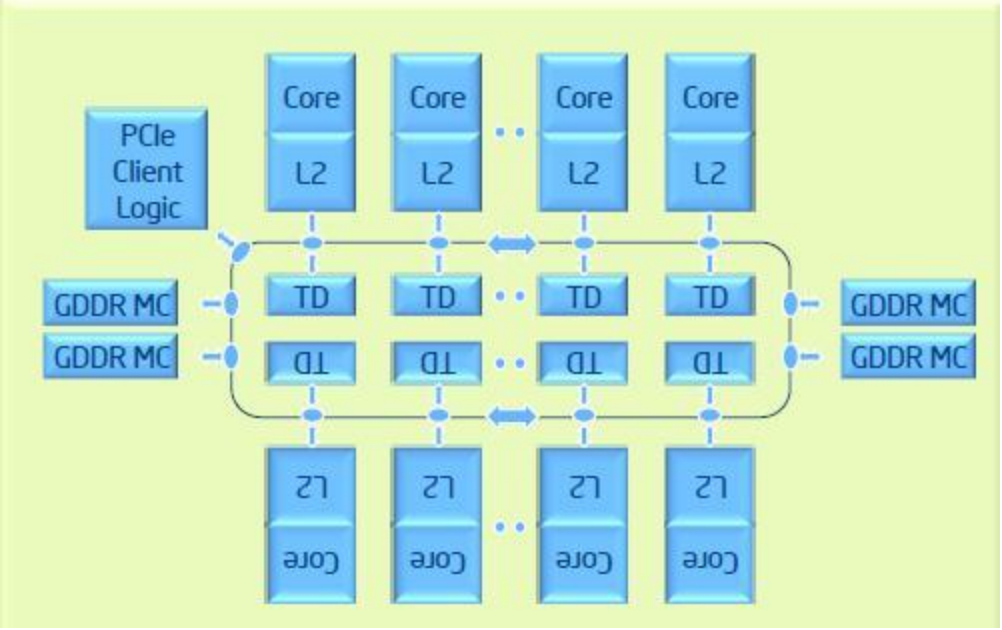
Back Ground

- Xeon Phi
  - Intel production
  - 22 nm process
  - 60 cores
    - 1.09 Ghz each
    - Four SMT threads
    - 512-bit vector-processing unit
    - 32 KB L1 cache
    - 512 KB L2 cache
    - Simple in-order cores
  - Internal network
    - Ring bus
  - Shared state
    - MESI coherency protocol



Microarchitecture

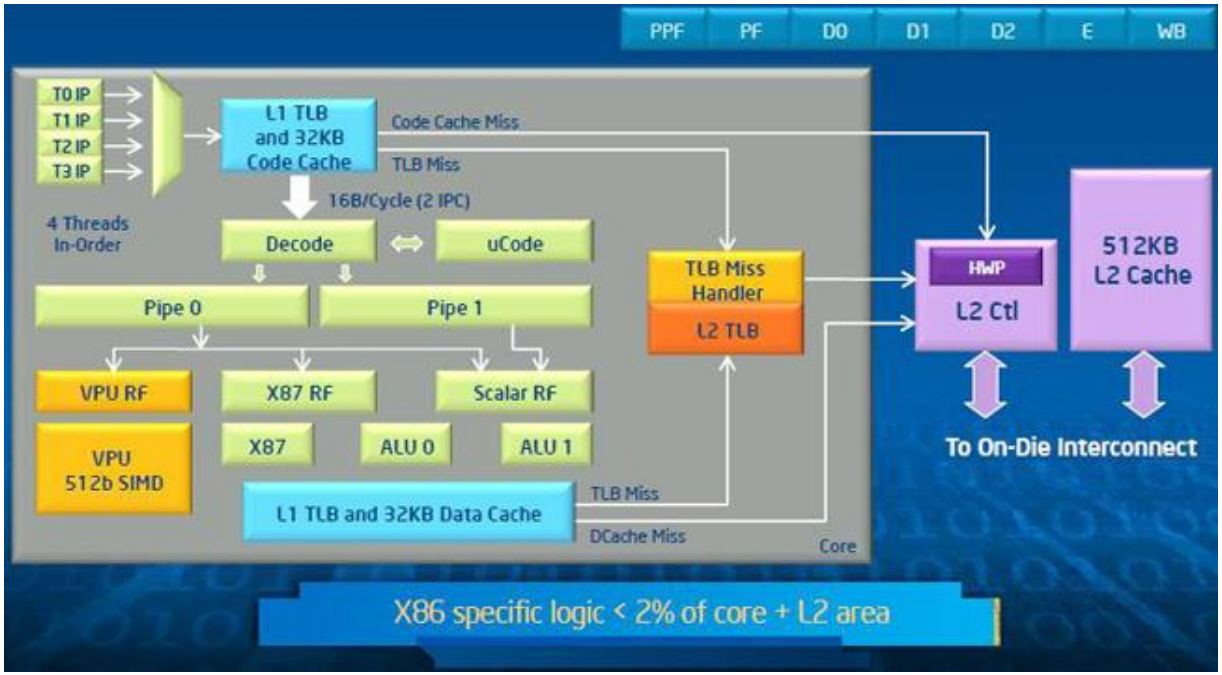
- Ring interconnect
  - High bandwidgth
  - Bidirectional
- L2 cash
  - Private
  - Coherency by
    - Global-distributed tag directory
  - Interface to GDDR5
    - Memory controllers
      - Coprocessor
    - PCIe client logic
      - PCIe bus



Xeon Phi Microarchitecture

Core

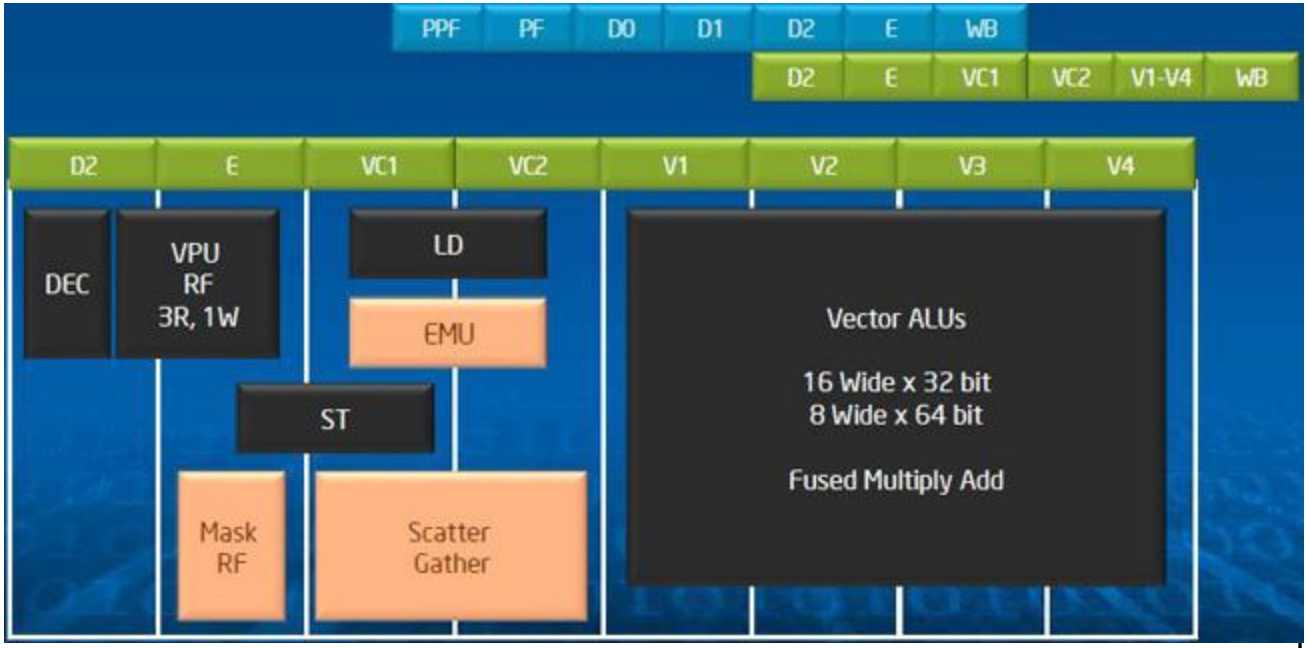
- Power efficient
- High throughput
  - Parallel workload
- Short pipeline
  - In-order
- Supporting four HW threads
- Per core VPU
- Support legacy architecture
  - Low cost



Intel® Xeon Phi™ Coprocessor Core

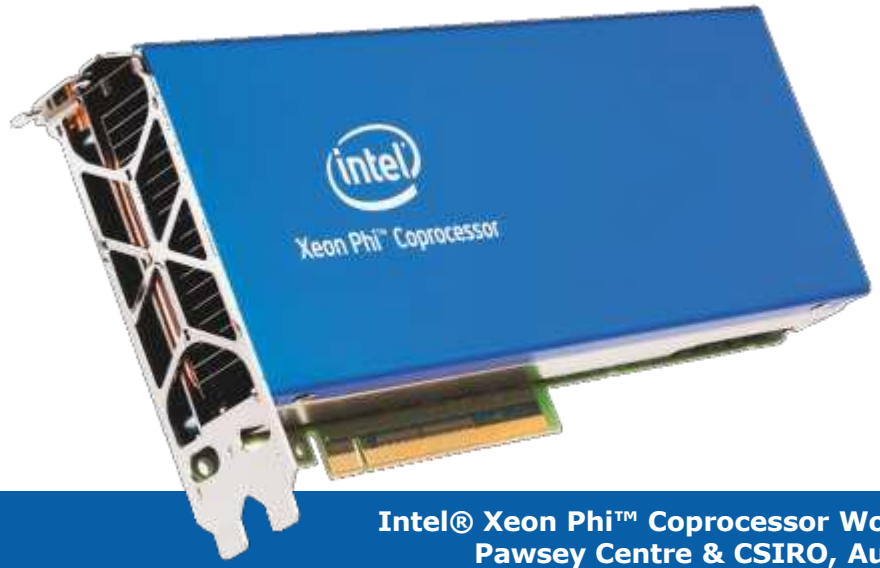
Vector Processing Unit

- 512-bit SIMD
- 16X32bit vector
- 8X64bit vector
- Fused Multiply-Add
  - Per cycle operations
    - 32 SP floating point
    - 16 DP floating point
- Supports integers



Vector Processing Unit

Intel® Xeon Phi™ Coprocessor Architecture Overview



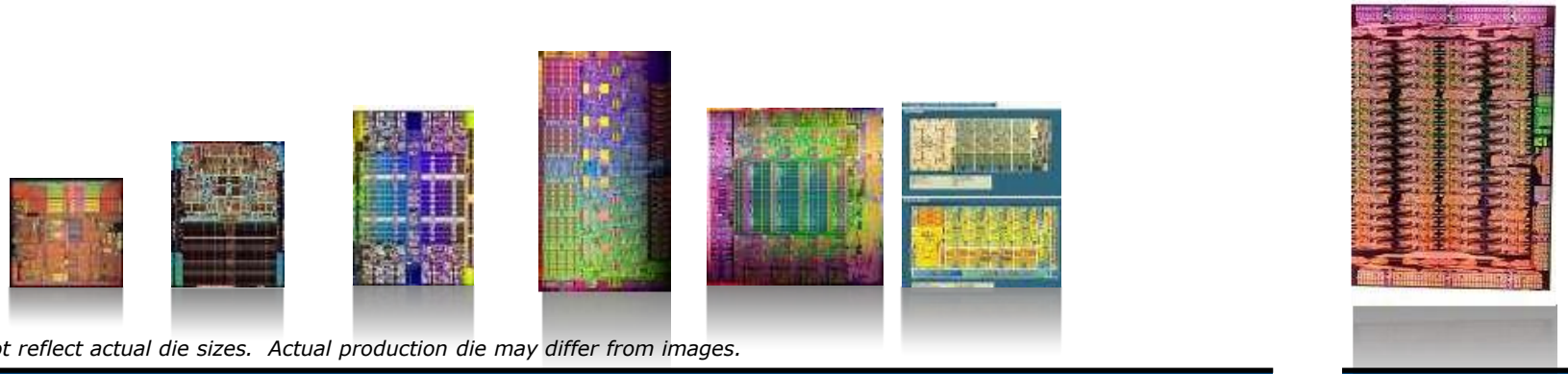
Architecture Topics

- Intel® Many Integrated Core (MIC) Architecture
- Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Setting Expectations
- Performance
- Summary

Module Outline

- **Intel® Many Integrated Core (MIC) Architecture**
- Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Setting Expectations
- Performance
- Summary

Intel Architecture Multicore and Manycore
More cores. Wider vectors. Coprocessors.

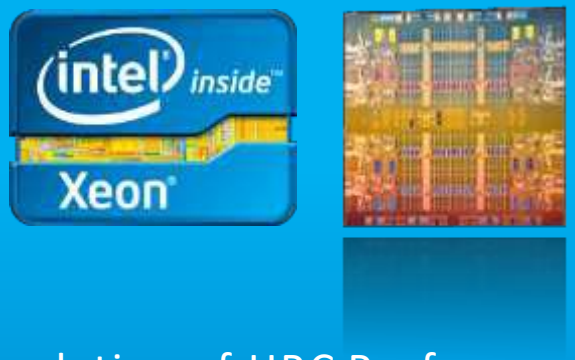


Images do not reflect actual die sizes. Actual production die may differ from images.

	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor E5 Product Family	Intel® Xeon® processor code name Ivy Bridge	Intel® Xeon® processor code name Haswell	Intel® Xeon Phi™ Coprocessor
Core(s)	1	2	4	6	8	10	To be determined	61
Threads	2	2	8	12	16	20		244

Intel® Xeon Phi™ coprocessor extends established CPU architecture and programming concepts to highly parallel applications

Intel® Multicore Architecture

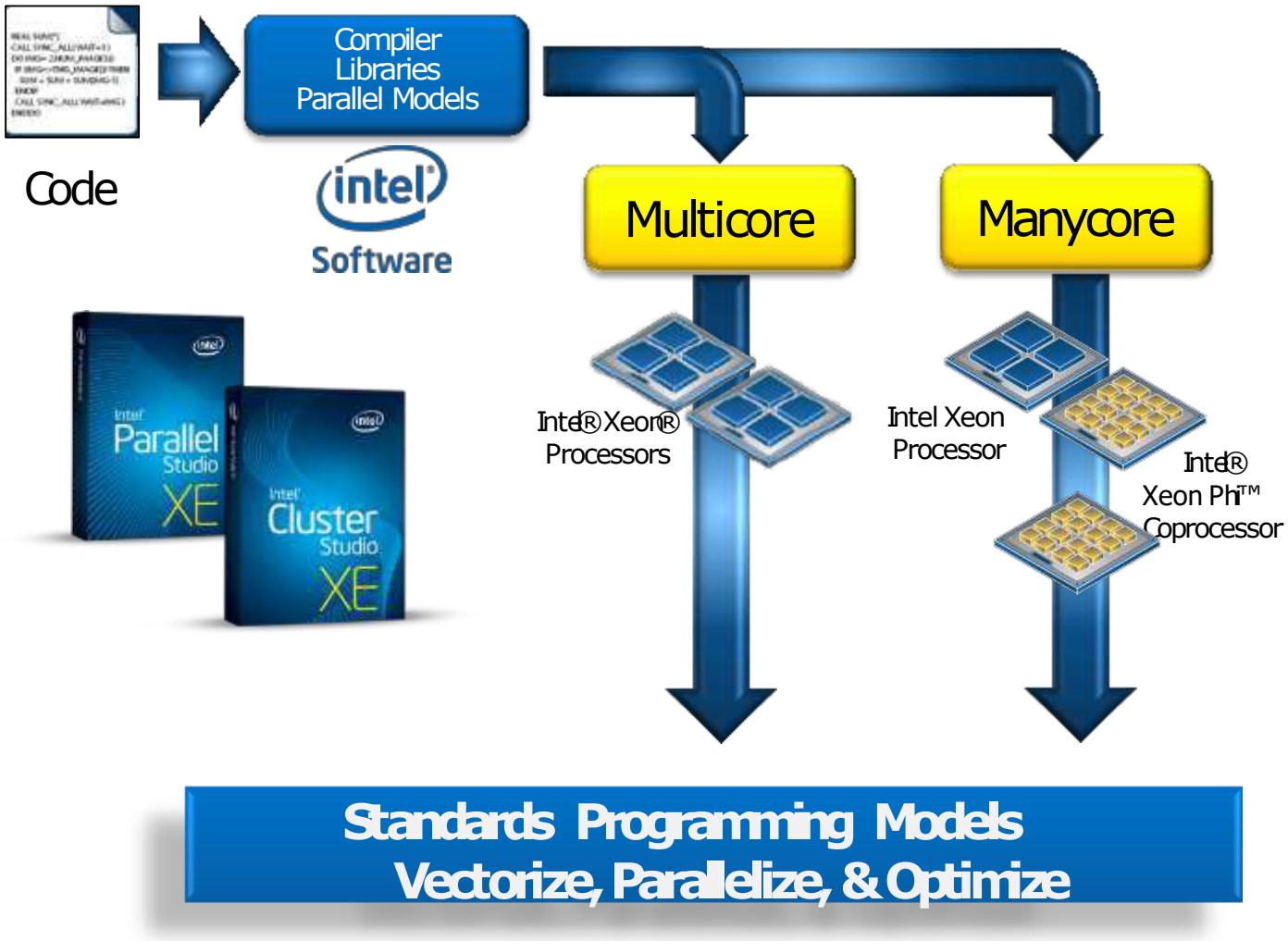


- Foundation of HPC Performance
- Suited for full scope of workloads
- Industry leading performance and performance/watt for serial & parallel workloads
- Focus on fast single core/thread performance with "moderate" number of cores

Intel® Many Integrated Core Architecture

- Performance and performance/watt optimized for highly parallelized compute workloads
- Common software tools with Intel® Xeon® architecture enabling efficient application readiness and performance tuning
- Intel® Architecture extension to Manycore
- Many cores/threads with wide SIMD

Consistent Tools & Programming Models



Module Outline

- Intel® Many Integrated Core (MIC) Architecture
- **Intel® Xeon Phi™ Coprocessor Overview**
- Core and Vector Processing Unit
- Setting Expectations
- Performance
- Summary

Introducing Intel® Xeon Phi™ Coprocessors

Highly-parallel Processing for Unparalleled Discovery

Groundbreaking differences

Up to 61 Intel® Architecture cores/1.1 GHz/ 244 threads

Up to 8 GB memory with up to 352 GB/s bandwidth

512-bit SIMD instructions

Linux* operating system, IP addressable

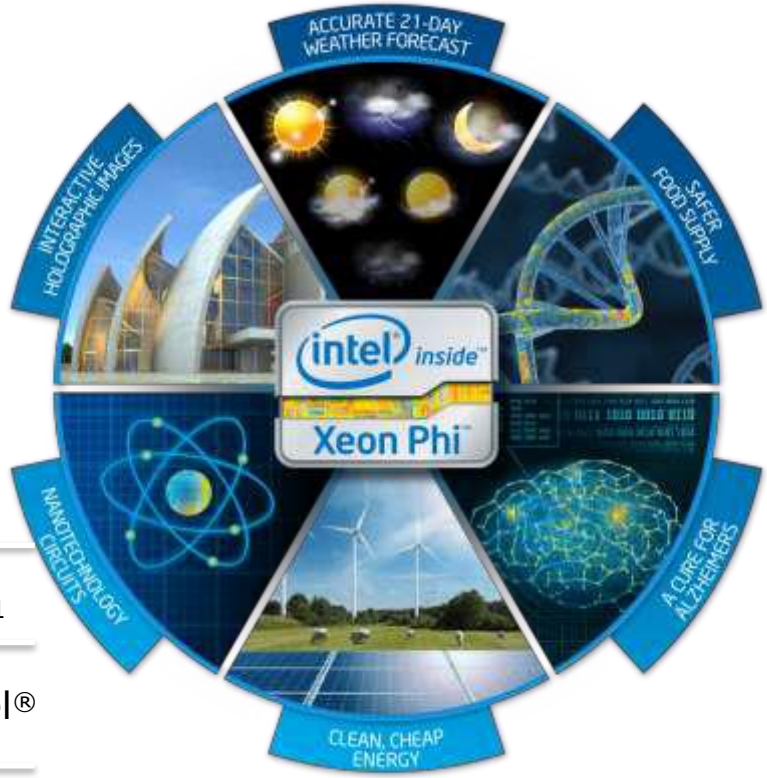
Standard programming languages and tools

Leading to Groundbreaking results

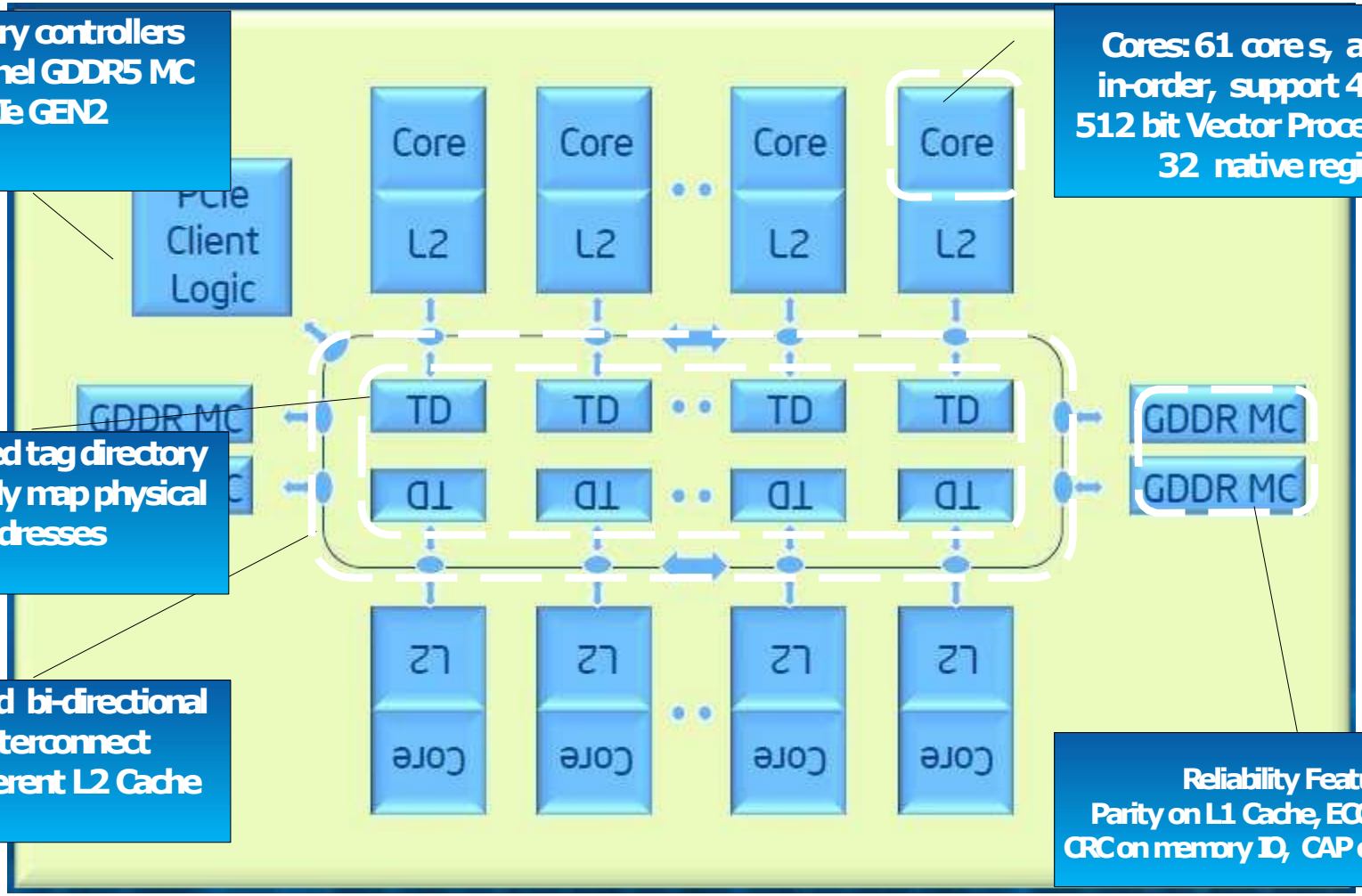
Over 1 TeraFlop/s double precision peak performance[1]

Up to 2.2x higher memory bandwidth than on an Intel® Xeon® processor E5 family-based server[2]

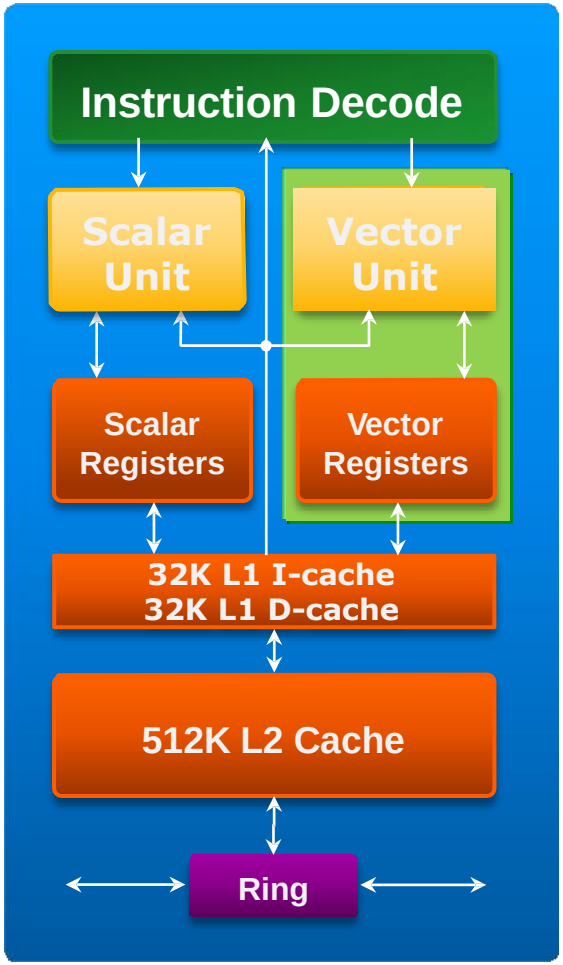
Up to 4x more performance per watt than with an Intel Xeon processor E5 family-based server[3]



Intel® Xeon Phi™ Architecture Overview

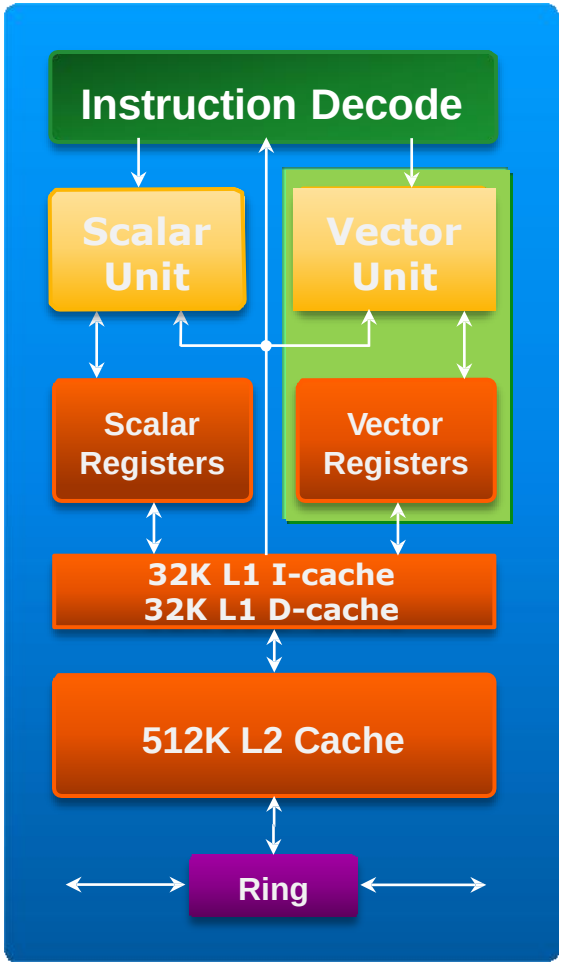


Core Architecture Overview



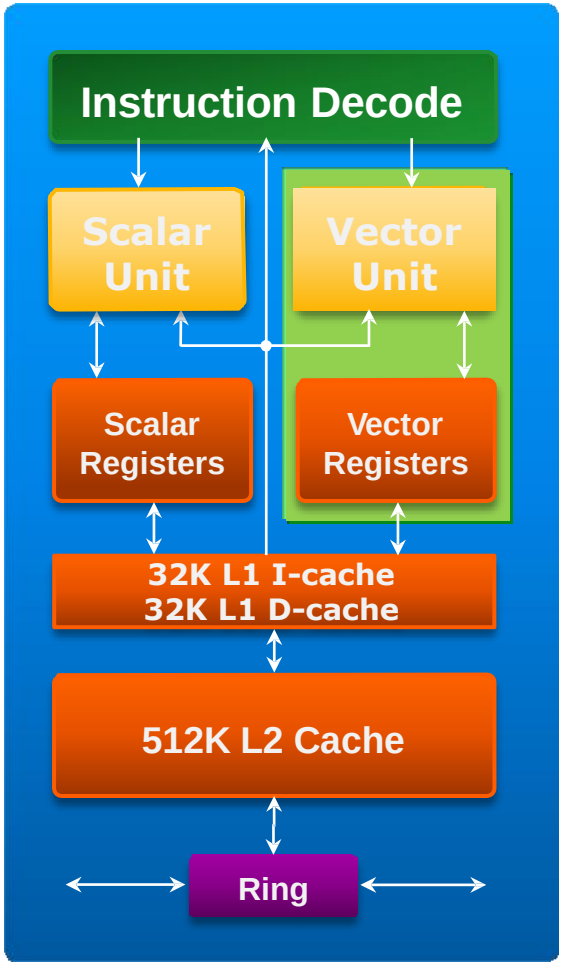
- 60+ in-order, low-power Intel® Architecture cores in a ring interconnect
- Two pipelines
  - Scalar Unit based on Pentium® processors
  - Dual issue with scalar instructions
  - Pipelined one-per-clock scalar throughput
- SIMD Vector Processing Engine
- 4 hardware threads per core
  - 4 clock latency, hidden by round-robin scheduling of threads
  - Cannot issue back-to-back inst in same thread
- Coherent 512 KB L2 Cache per core

Core Architecture Overview



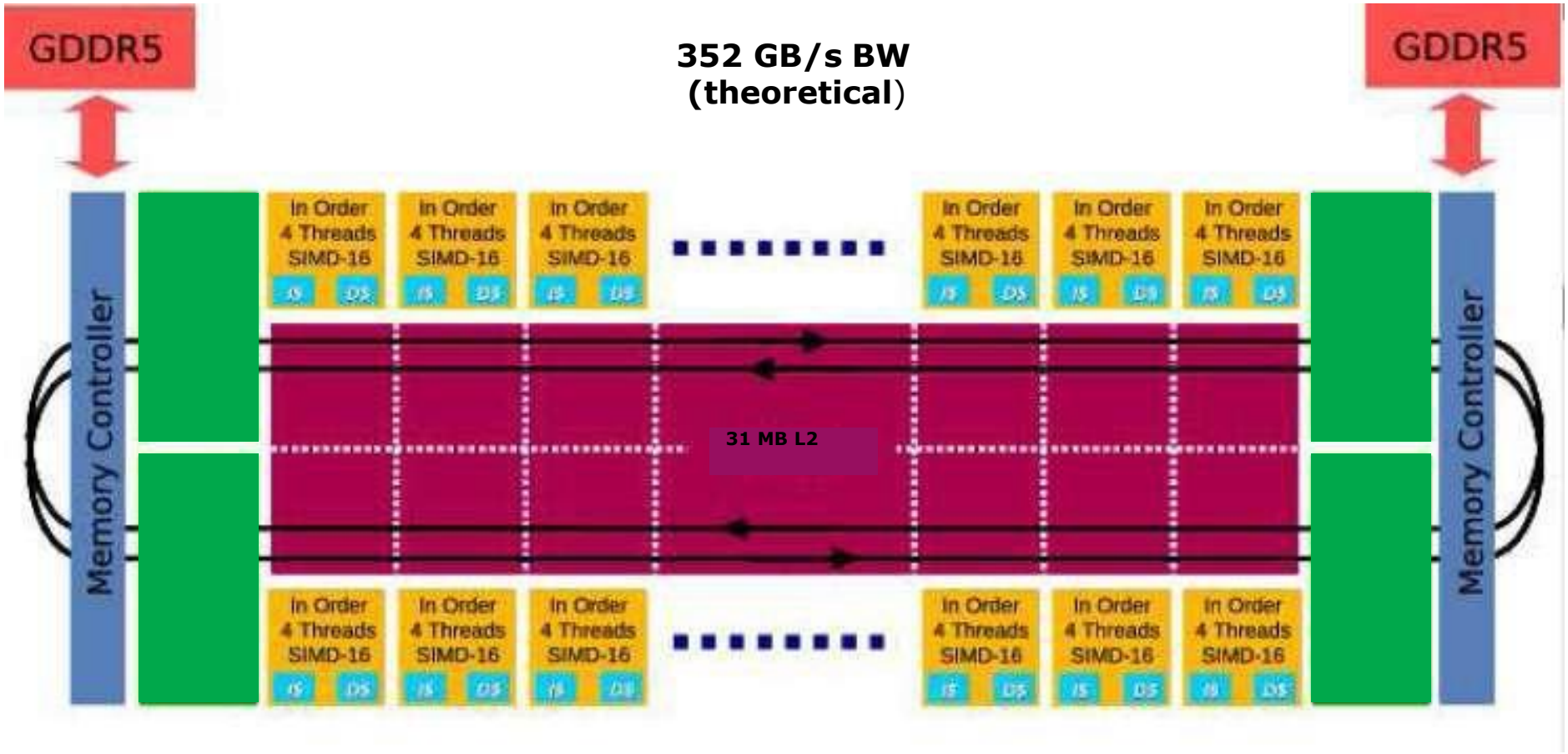
- 2 issue (1 scalar/1 vector)
- 2 cycle decoder: no back-to-back cycle issue from the same context (thread)
- Most vec instructions have 4 clock latency
- At least two HW contexts (thread/proc) to fully utilize the core

Core Architecture Overview

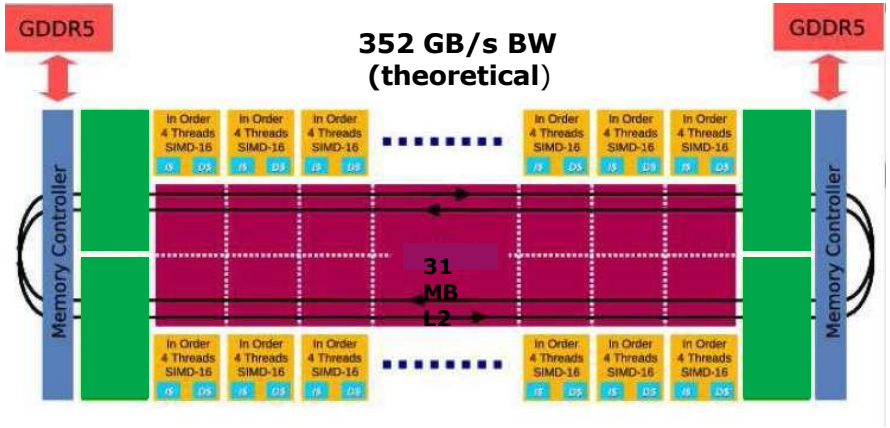


- Performance Monitoring Unit
  - 4 event select register
  - 4 performance counters
  - Shared among the 4 HW threads
  - Programmable via model specific registers (MSR) using RDMSR/WRMSR

Knights Corner Architecture Overview – Cache



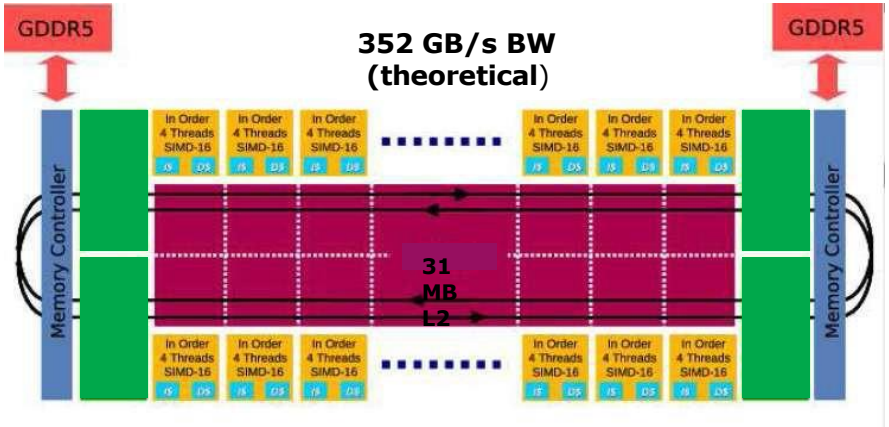
Knights Corner Architecture Overview – Cache



- L1 cache
  - 32K I-cache per core
  - 32K D-cache per core
  - 8 way associative
  - 64B cache line
  - 3 cycle access
  - Up to 8 outstanding requests
  - Fully coherent (MESI)

- L2 cache
  - 512K Unified per core
  - 8 way assoc
  - Inclusive
  - 31M total across 62 cores
  - 11 cycle best access
  - Up to 32 outstanding requests
  - Streaming HW prefetcher
  - Fully coherent

Knights Corner Architecture Overview – Cache



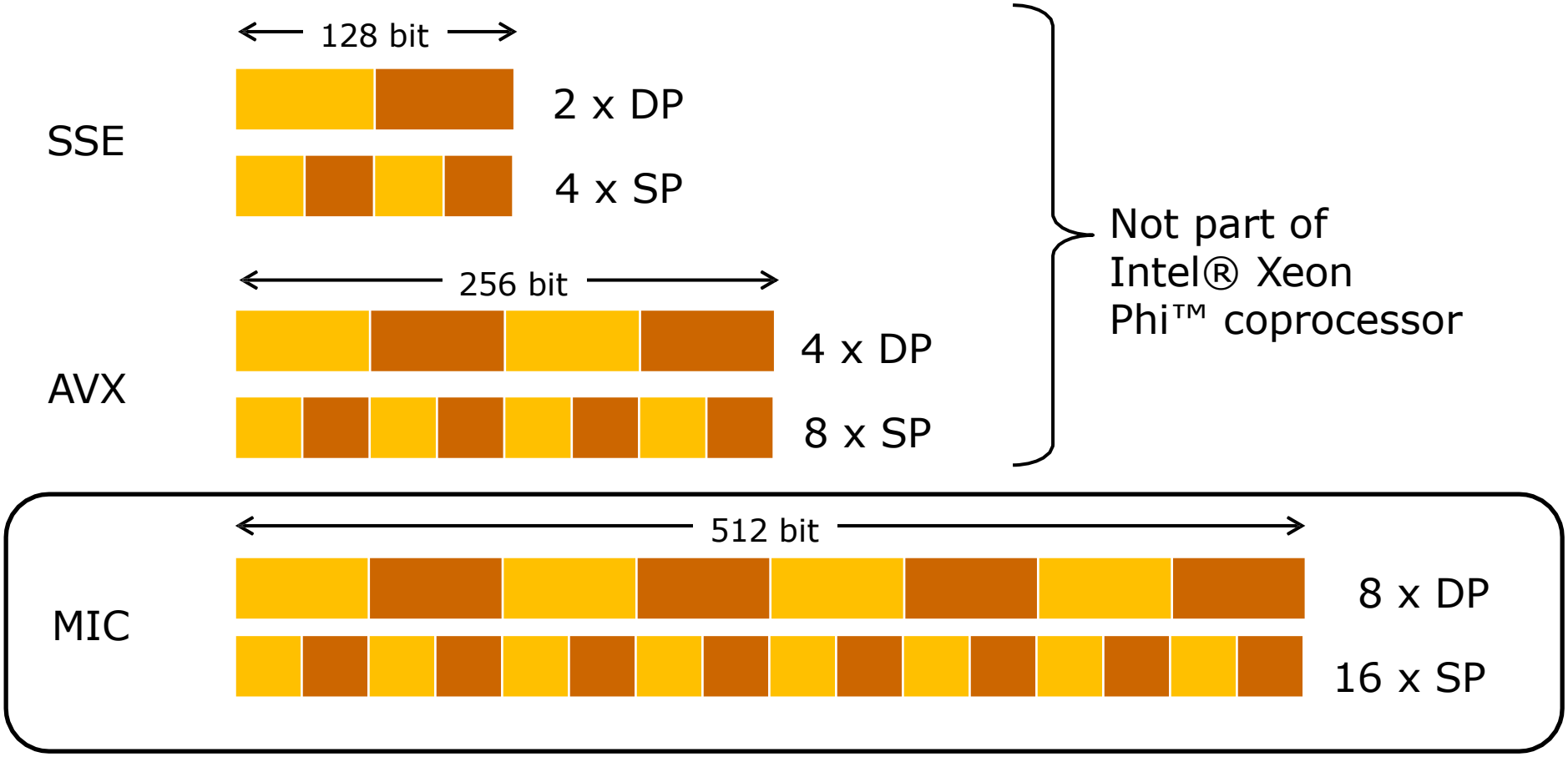
- Alignment
  - Based upon number of elements, element size, and vector load and store instruction
  - 64B alignment for 4B (float) data elements for a 16 to 16 vector load

- Memory
  - 8GB GDDR5
  - 8 Memory controllers, 16 GDDR5 channels, up to 5.5 GT/s
  - 300 ns access
  - Aggregate 352 GB/s peak memory bandwidth
  - ECC
- PCI Express*
  - Gen2 (Client) x16 per direction

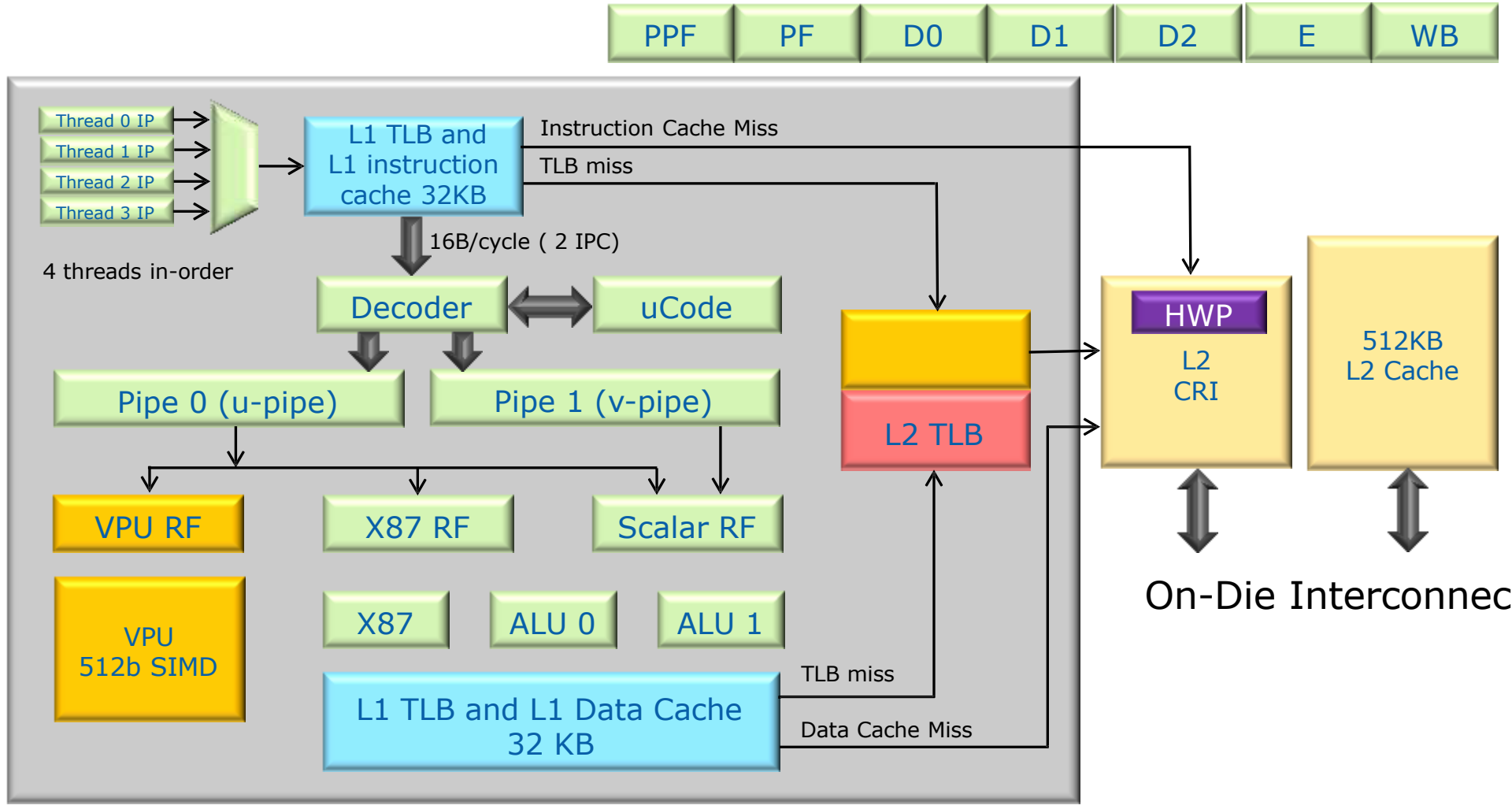
Module Outline

- Intel® Many Integrated Core (MIC) Architecture
- Intel® Xeon Phi™ Coprocessor Overview
- **Core and Vector Processing Unit**
- Setting Expectations
- Performance
- Summary

Knights Corner Architecture Overview
Vector Processing Unit and ISA



Vector Processing Unit Extends the Scalar IA Core



Vector Processing Unit and Intel® Initial Many Core Instructions (Intel® IMCI)

- Vector Processing Unit Execute Intel IMCI
- 512-bit Vector Execution Engine
  - 16 lanes of 32-bit single precision and integer operations
  - 8 lanes of 64-bit double precision and integer operations
  - 32 512-bit general purpose vector registers in 4 thread
  - 8 16-bit mask registers in 4 thread for predicated execution
- Read/Write
  - One vector length (512-bits) per cycle from/to Vector Registers
  - One operand can be from memory
- IEEE 754 Standard Compliance
  - 4 rounding models, even, 0, +∞, -∞
  - Hardware support for SP/DP denormal handling
  - Sets status register VXCSR flags but not hardware traps

Vector Instruction Performance

- VPU contains 16 SP ALUs, 8 DP ALUs
- Most VPU instructions have a latency of 4 cycles and TPT 1 cycle
  - Load/Store/Scatter have 7-cycle latency
  - Convert/Shuffle have 6-cycle latency
- VPU instruction are issued in u-pipe
- Certain instructions can go to v-pipe also
  - Vector Mask, Vector Store, Vector Packstore, Vector Prefetch, Scalar

Module Outline

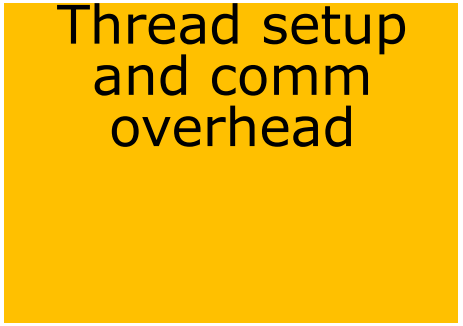
- Intel® Many Integrated Core (MIC) Architecture
- Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Software Ecosystem
- Performance
- Summary

Programming Considerations - Setting Expectations (1)

- Getting full performance from the Intel® MIC architecture requires both a high degree of parallelism *and* vectorization
  - Not all code can be written this way
  - Not all programs make sense on this architecture

- The Intel® Xeon® processor is not an Intel® Xeon® processor
  - It specializes in running highly parallel and vectorized code
  - New vector instruction set and 512-bit wide registers
  - Not optimized for processing serial code

Programming Considerations - Setting Expectations (2)



- Coprocessor comes with 8 GB of memory
  - Only ~7 GB is available to your program
    - The other ~1GB is used for data transfer and is accessible to your coprocessor code as buffers.
- Very short (low-latency) tasks not optimal for offload to the coprocessor
  - Costs that you need to amortize to make it worthwhile:
    - Code and data transfer
    - Process/thread creation
  - Fastest data transfers currently require careful data alignment
- This architecture is not optimized for serial performance

Intel® MIC Programming Considerations – This is not a GPU

- Very different memory architectures
  - The Intel MIC Architecture is not optimized for concurrent out-of-cache random memory accesses by large numbers of threads (GPUs are)
  - The Intel MIC Architecture has a "traditional" coherent-cache architecture
  - GPUs have a memory architecture specialized for localized "shared memory" processing

- "Threads" and "cores" mean something very different - GPU versions of these are limited and lighter-weight

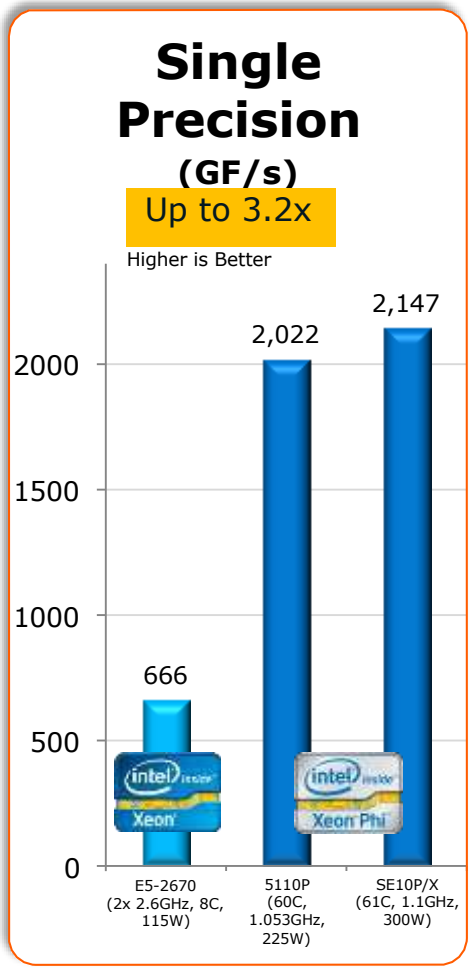
- Each architecture (host CPU, Intel MIC Architecture, or GPU) is really good at some things, and not others
  - Because the Intel MIC Architecture is similar to the Intel® Xeon® processor, probably your best choice for further accelerating highly parallel and vectorized code developed on Intel Xeon processor

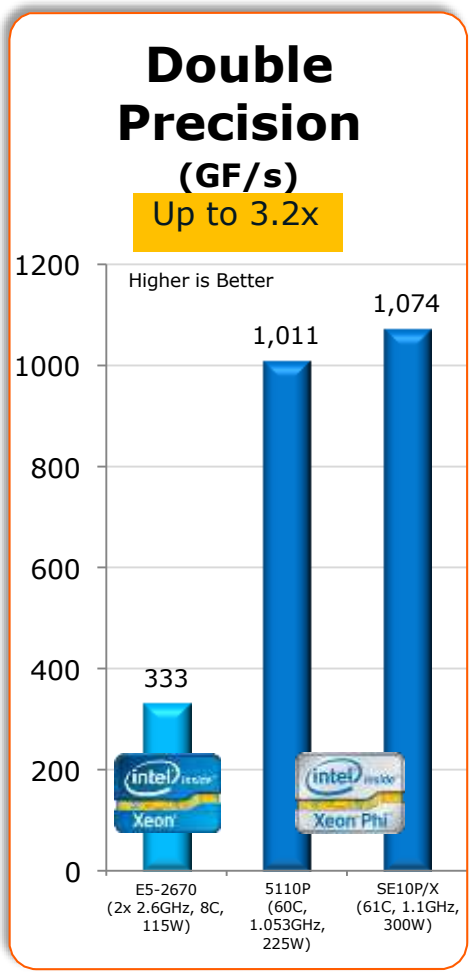
Module Outline

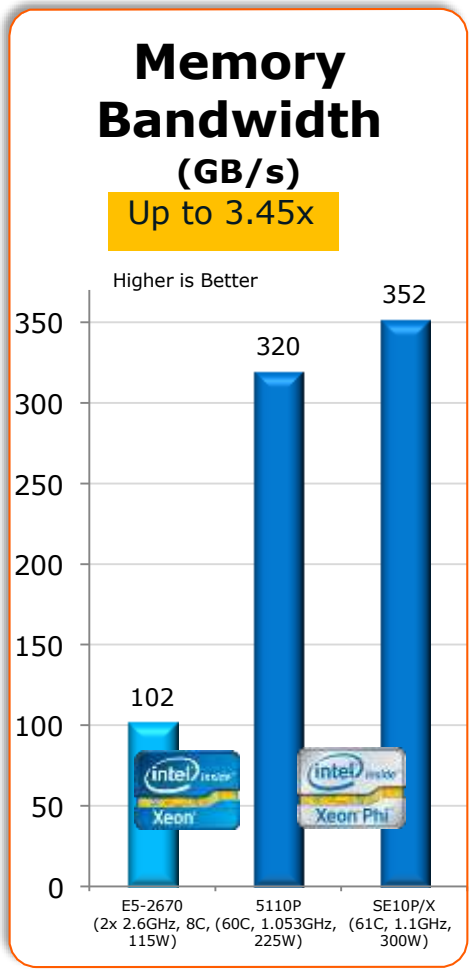
- Intel® Many Integrated Core (MIC) Architecture
- Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Setting Expectations
- **Performance**
- Summary

Theoretical Maximum

(Intel® Xeon® processor E5-2670 vs. Intel® Xeon Phi™ coprocessor 5110P & SE10P/X)

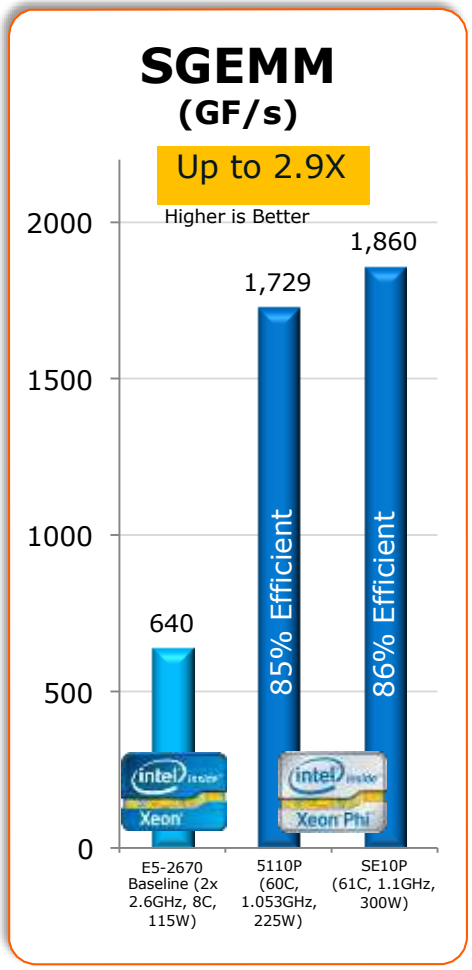


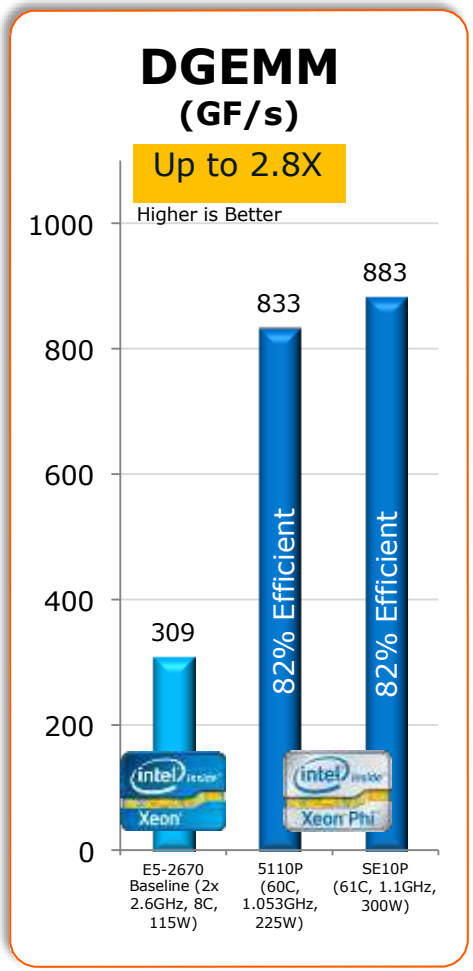


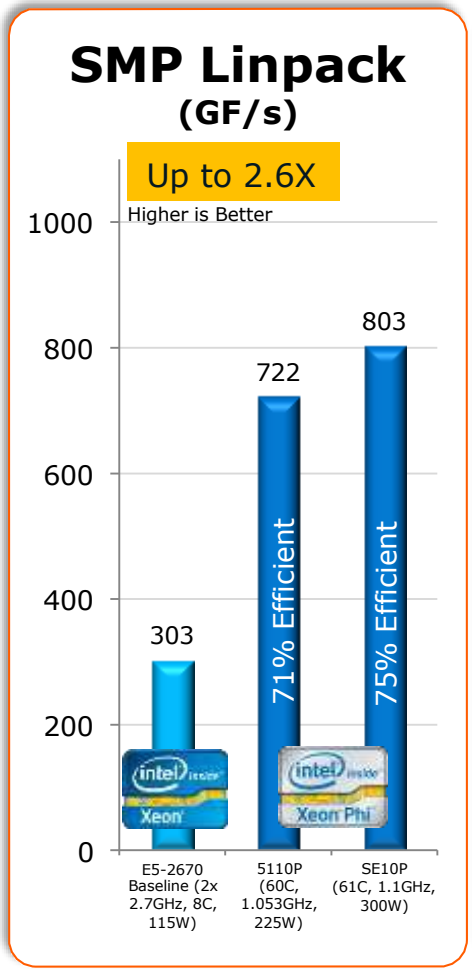


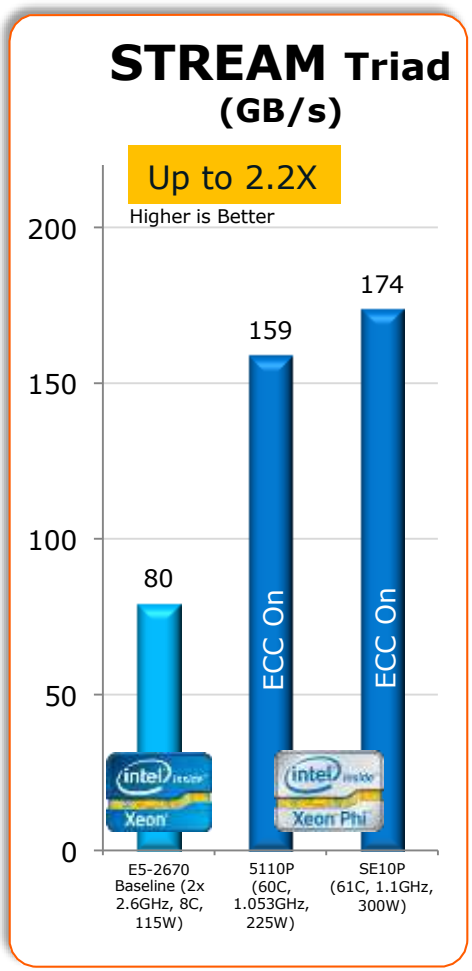
Source: Intel as of October 17, 2012 Configuration Details: Please reference slide speaker notes.
For more information go to http://www.intel.com/performance

Synthetic Benchmark Summary



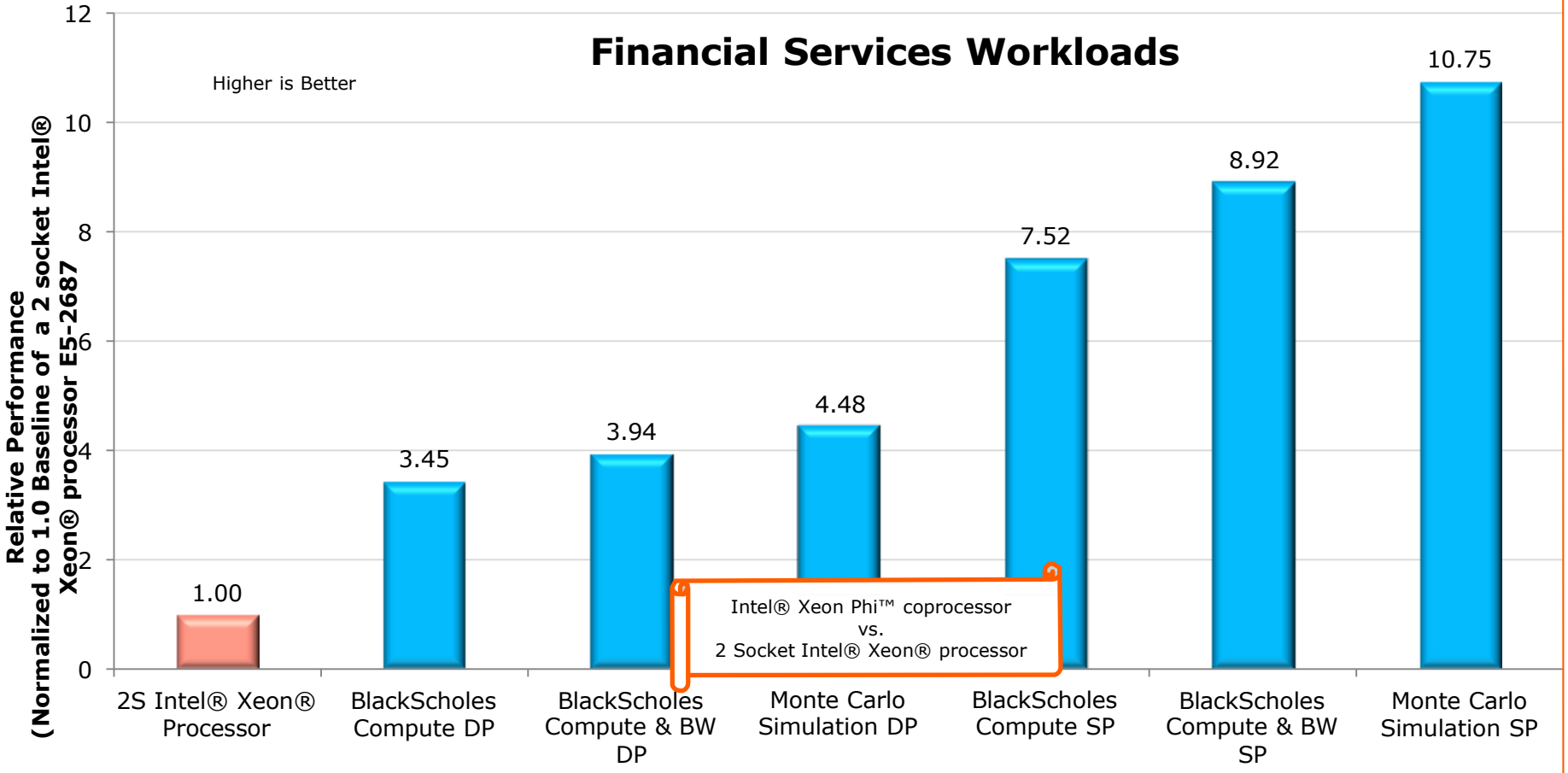






Coprocessor results: Benchmark runs 100% on coprocessor, no help from Intel® Xeon® processor host (aka native). For more information go to http://www.intel.com/performance

Intel® Xeon Phi™ Coprocessor vs. Intel® Xeon® Processor



Coprocessor results: Benchmark runs 100% on coprocessor, no help from Intel® Xeon® processor host (aka native). For more information go to http://www.intel.com/performance

Notes
1. 2 X Intel® Xeon® Processor E5-2670 (2.6GHz, 8C, 115W)
2. Intel® Xeon Phi™ coprocessor SE10 (ECC on) with pre-production SW stack

Higher SP results are due to certain Single Precision transcendental functions in the Intel® Xeon Phi™ coprocessor that are not present in the Intel® Xeon® processor

Module Outline

- Intel® Many Integrated Core (MIC) Architecture
- Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Setting Expectations
- Performance
- **Summary**

Architecture Summary

- Intel® Many Integrated Core (MIC) Architecture
- Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Setting Expectations
- Performance
- Summary