University of Siena

Microsoft

TERAFLUX.EU

Exploiting Dataflow Parallelism in Teradevice Computing

THALES

Barcelona
Supercomputing Center

University of Augsburg

CAPS

LABS hp

Roberto Giorgi – University of Siena (coordinator)

Cagliari, Italy – Computing Frontiers

16/05/2012

University of Cyprus

FUNDING OPPORTUNITIES from the
FUTURE & EMERGING TECHNOLOGIES scheme

MANCHESTER
1824

University of Manchester

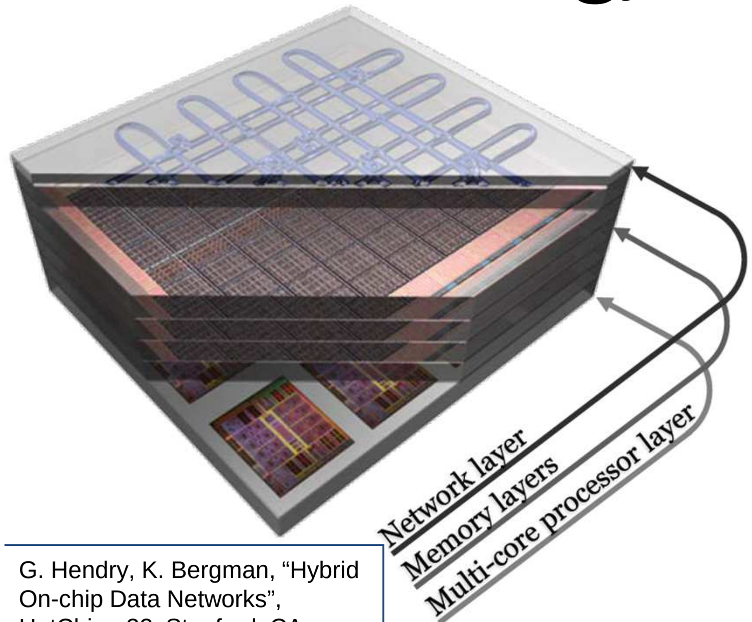
INRIA

What is **TERAFLUX** about

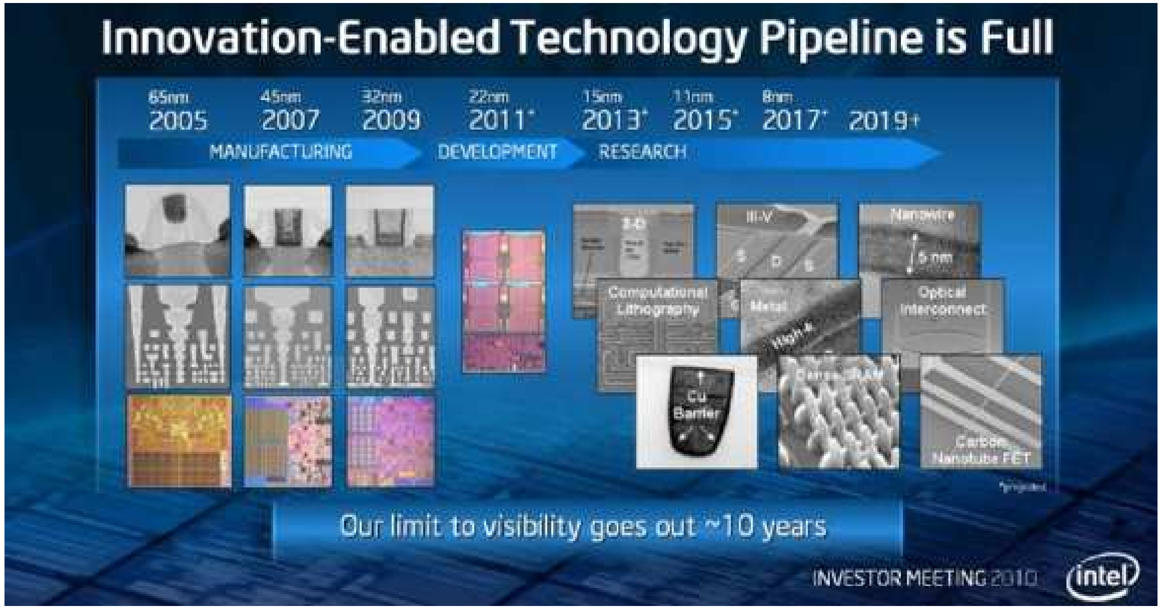
Architecture+Programmability+Reliability
of
Future (single chip)
Many-cores
(targeting 1000+ cores)

Future Scenarios

== 3D stacking, 8nm, 3D transistors, Graphene



G. Hendry, K. Bergman, "Hybrid On-chip Data Networks", HotChips-22, Stanford, CA – Aug. 2010

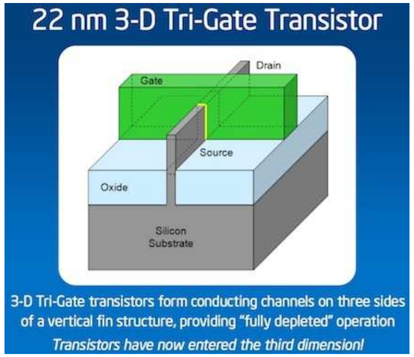


Innovation-Enabled Technology Pipeline is Full

Our limit to visibility goes out ~10 years

INVESTOR MEETING 2010

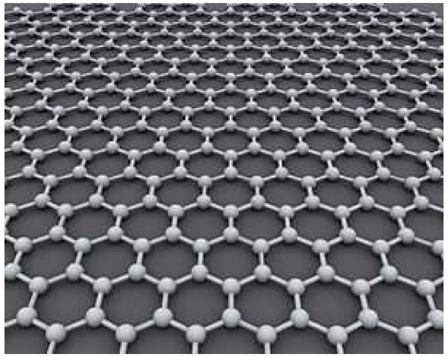
Fab D1X (OR), 42 (AZ) starting the 14nm node in 2013

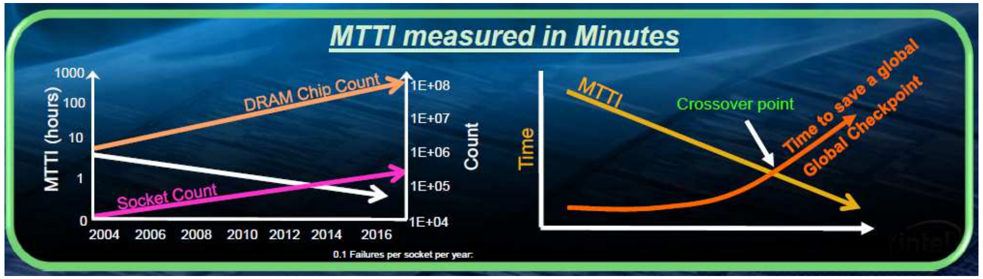


22 nm 3-D Tri-Gate Transistor

3-D Tri-Gate transistors form conducting channels on three sides of a vertical fin structure, providing "fully depleted" operation

Transistors have now entered the third dimension!





MTTI measured in Minutes

Pawloski, May 2011, Exascale Seminar, Ghent

TERA^FLUX

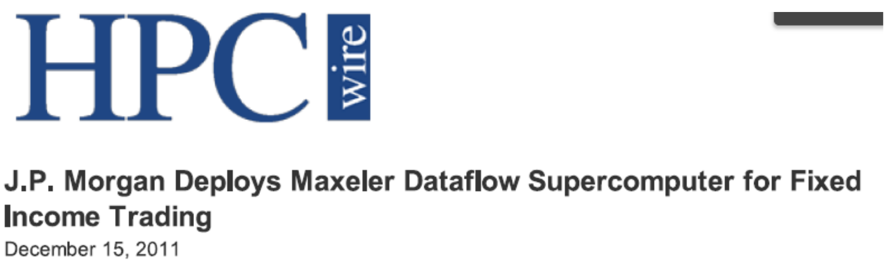
Fundamental approach: DATAFLOW

A Scheme of Computation in which an activity is initiated by presence of the data it needs to perform its function

(Jack Dennis)

Recent Projects/Efforts towards DATAFLOW

- Maxeler (UK) selling "dataflow computer" to J.P. Morgan → about 350x speedup vs. standard x86 cores

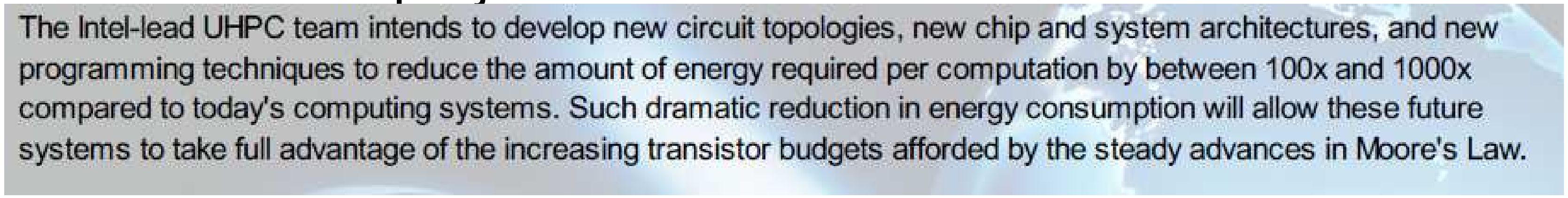


J.P. Morgan Deploys Maxeler Dataflow Supercomputer for Fixed Income Trading

December 15, 2011

- DARPA funding 25M$ for UPHC program, encompassing:
  - Gao's dataflow execution model (codelet based) – SWARM by ETI
  - Intel's Runnamede project

UPHC=Ubiquitus High-Performance Computing

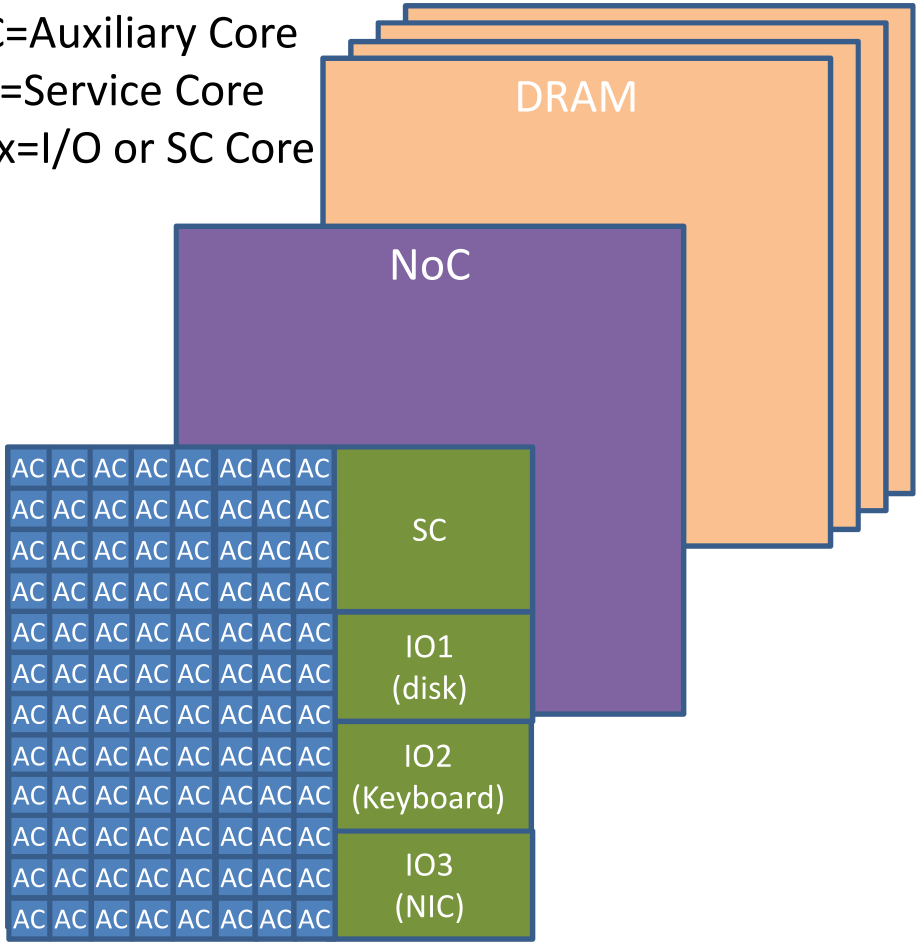


The Intel-lead UHPC team intends to develop new circuit topologies, new chip and system architectures, and new programming techniques to reduce the amount of energy required per computation by between 100x and 1000x compared to today's computing systems. Such dramatic reduction in energy consumption will allow these future systems to take full advantage of the increasing transistor budgets afforded by the steady advances in Moore's Law.

TERAFLUX – Future Many-cores

Key Challenges: Architecture+Programmability+Reliability

AC=Auxiliary Core
SC=Service Core
IOx=I/O or SC Core



TERAFLUX Key Facts:

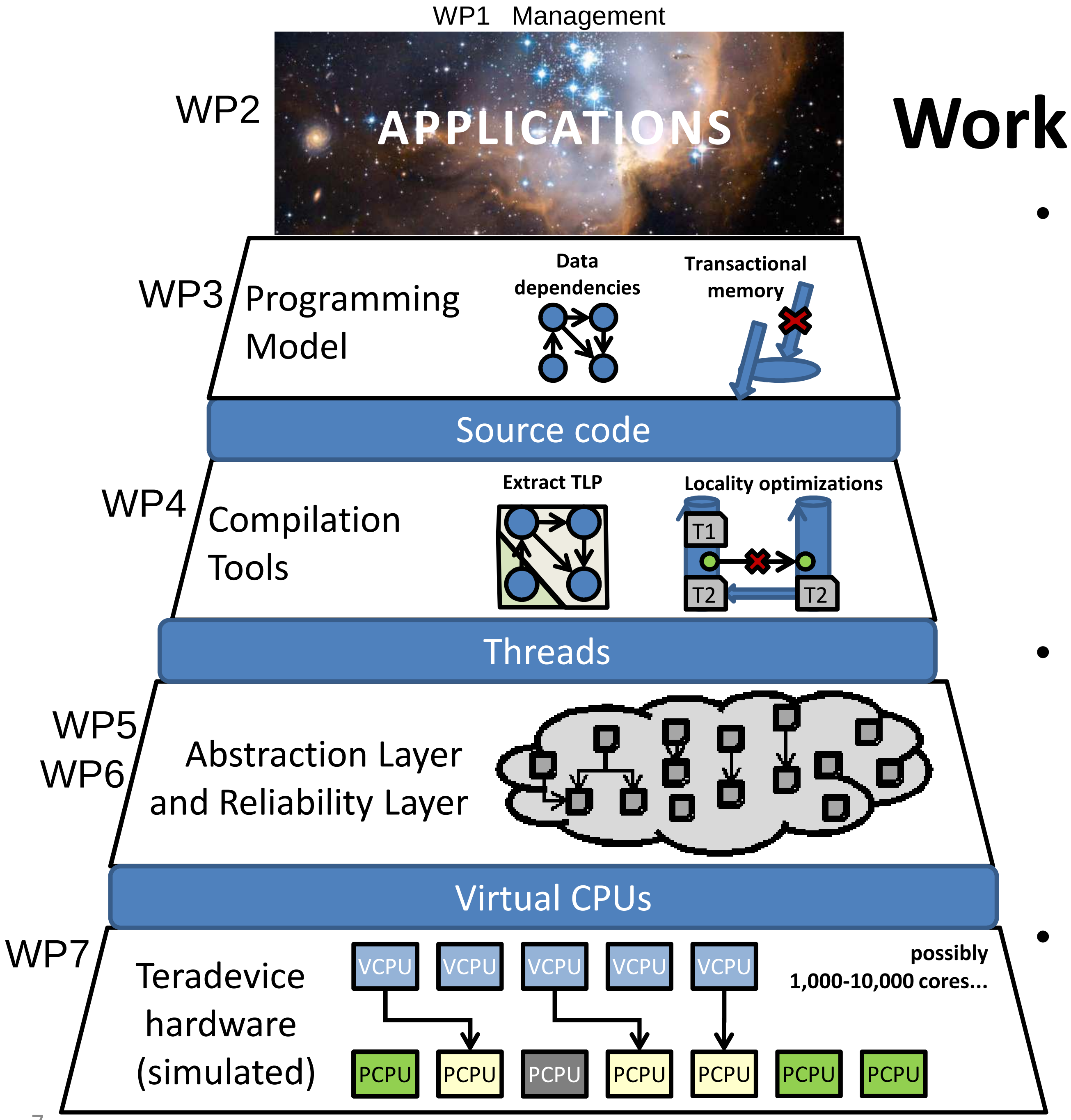
- FET – Integrated Project – 10 Partners – 4 years – 2010-2013
- 7.5 Meuro Total cost (5.7 Meuro EU funded)

TERAFLUX-INCO Key Facts:

- FET – EU Worldwide Cooperation – Extends TERAFLUX to an additional USA partner for 21 months
- 546 Keuro Total cost (420 Keuro EU funded)

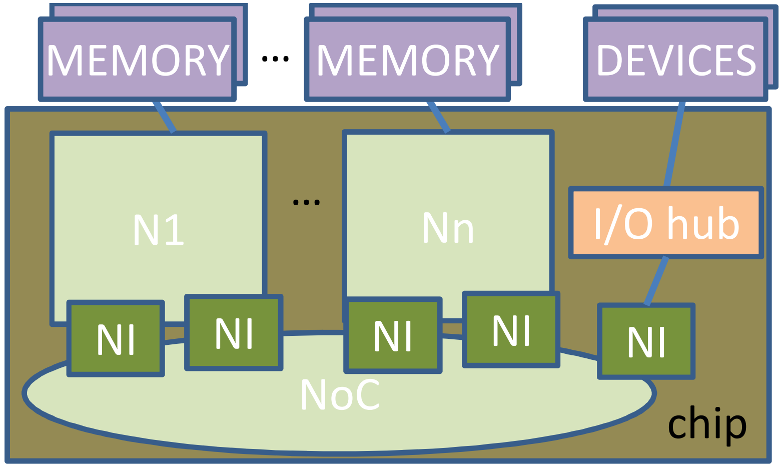
TERAFLUX.EU

Working Hypothesis



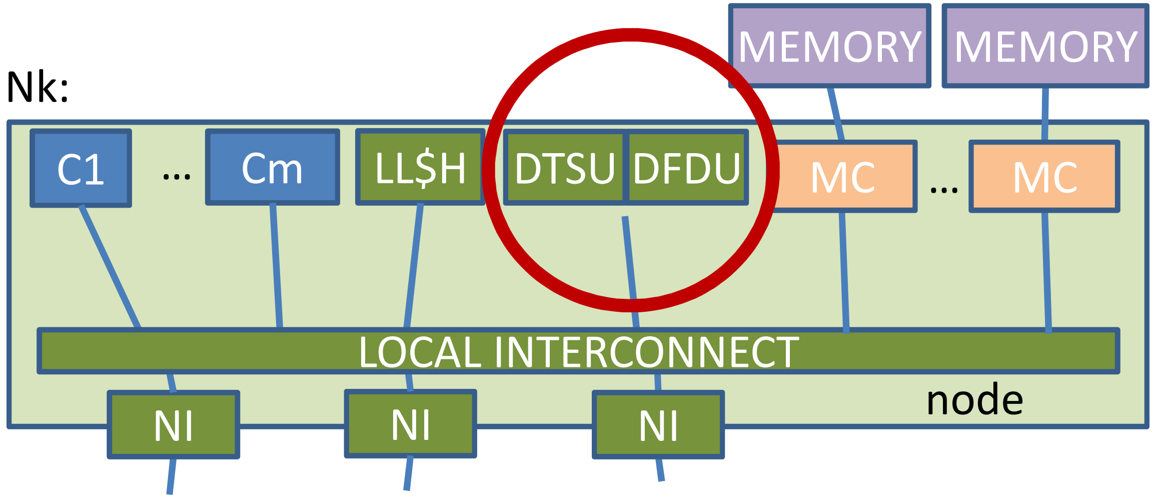
- 1000 Billion- or 1 TERA-device computing platforms pose new challenges:
  - (at least) programmability, complexity of design, reliability
- TERAFLUX context:
  - High performance computing and applications (not necessarily embedded)
- TERAFLUX scope:
  - Exploiting a less exploited path (DATAFLOW) at each level of abstraction

TERAFLUX Architectural template

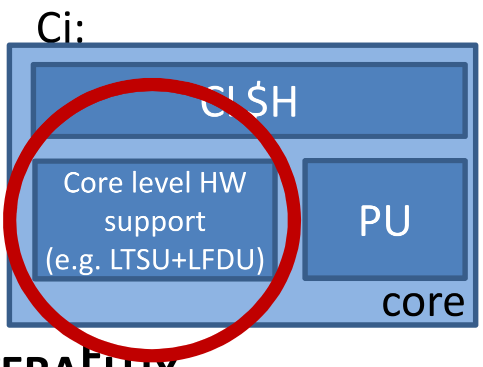


LEGENDA:
n = # of nodes
m = # of cores per node
u= # of DRAM controllers insisting on the Unified Physical Address Space
z = # of I/O Hubs

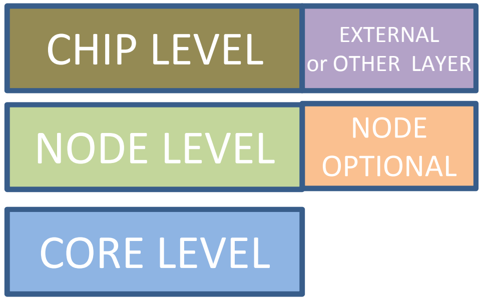
Nk = k-th Node (k=1..n)
NI = Network Interface
NoC = Network on Chip



Cj = j-th core (j=1..m)
MC = Memory Controller
DTSU = Distributed Thread-Scheduler Unit
DFDU = Distributed Fault-Detection Unit
LL$H = Last Level Cache Hierarchy



CL$H = Core Level Cache Hierarchy
PU = Processing Unit
LTSU = Local Thread-Scheduler Unit
LFDU = Local Fault-Detection Unit



TERAFLUX

Our pillars

- FIXED and MOST-USED ISA (**x86**)
- MANYCORE FULL SYSTEM SIMULATOR (**COTSon**)
- REAL WORLD APPLICATIONS (e.g. GROMACS)
- SYNCHRONIZATION: **TRANSACTIONAL MEMORY**
- **GCC** based TOOL-CHAIN
- OFF-THE-SHELF COMPONENTS FOR CORES, OS, NOC,MEMORY HIERARCHY
- **FDU** AND **TSU** (Fault Detection Unit and Thread Scheduling Unit)

A REVIEW OF RECENT MANY-CORES

HotChips 2011

- Hot Chips papers suggest that the rest of the world is moving in a different direction: large numbers of relatively simple CPUs. But the trend is reinforcing a long-appreciated set of questions—as the number of cores grows, **how do you deal scalability with interconnect, memory hierarchy, coherency, and intra-thread synchronization?** Answers to these questions depend on the size of the design, the application space, and the heritage of the design team.

R. Wilson, EETimes, 24/8/2011

SARC Architecture

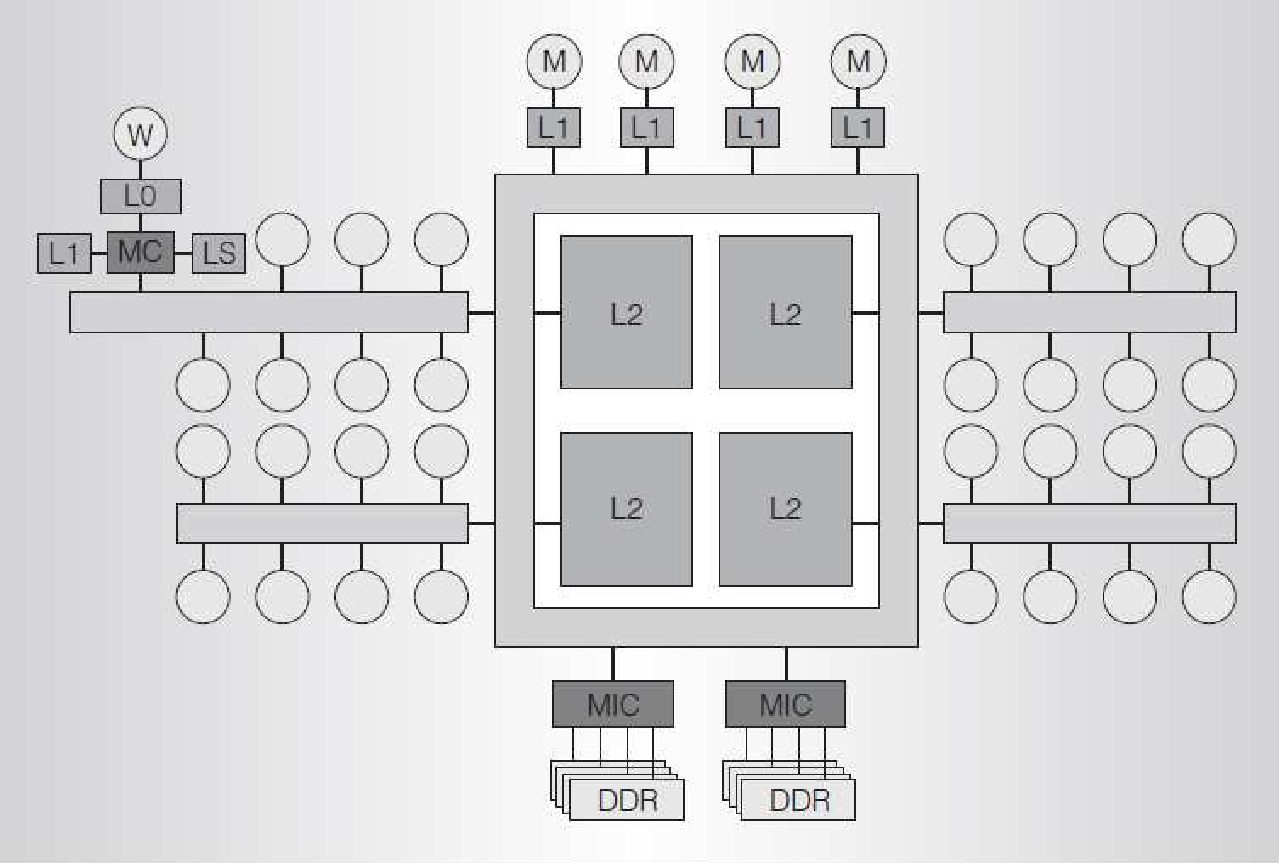


Figure 1. Schematic of the SARC architecture. The number of masters, workers, level-2 (L2) blocks, and memory interface controllers is implementation dependent, as is their on-chip layout.

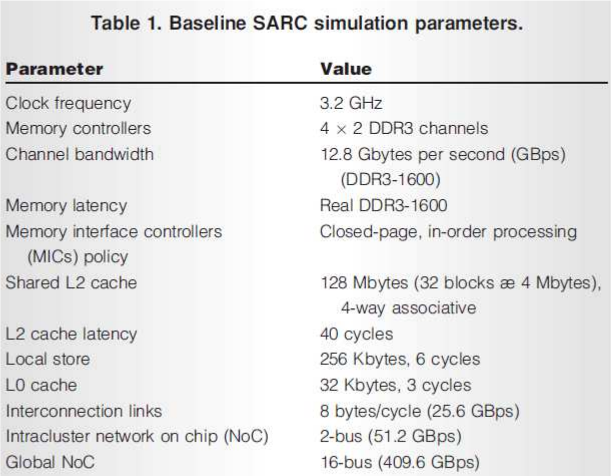


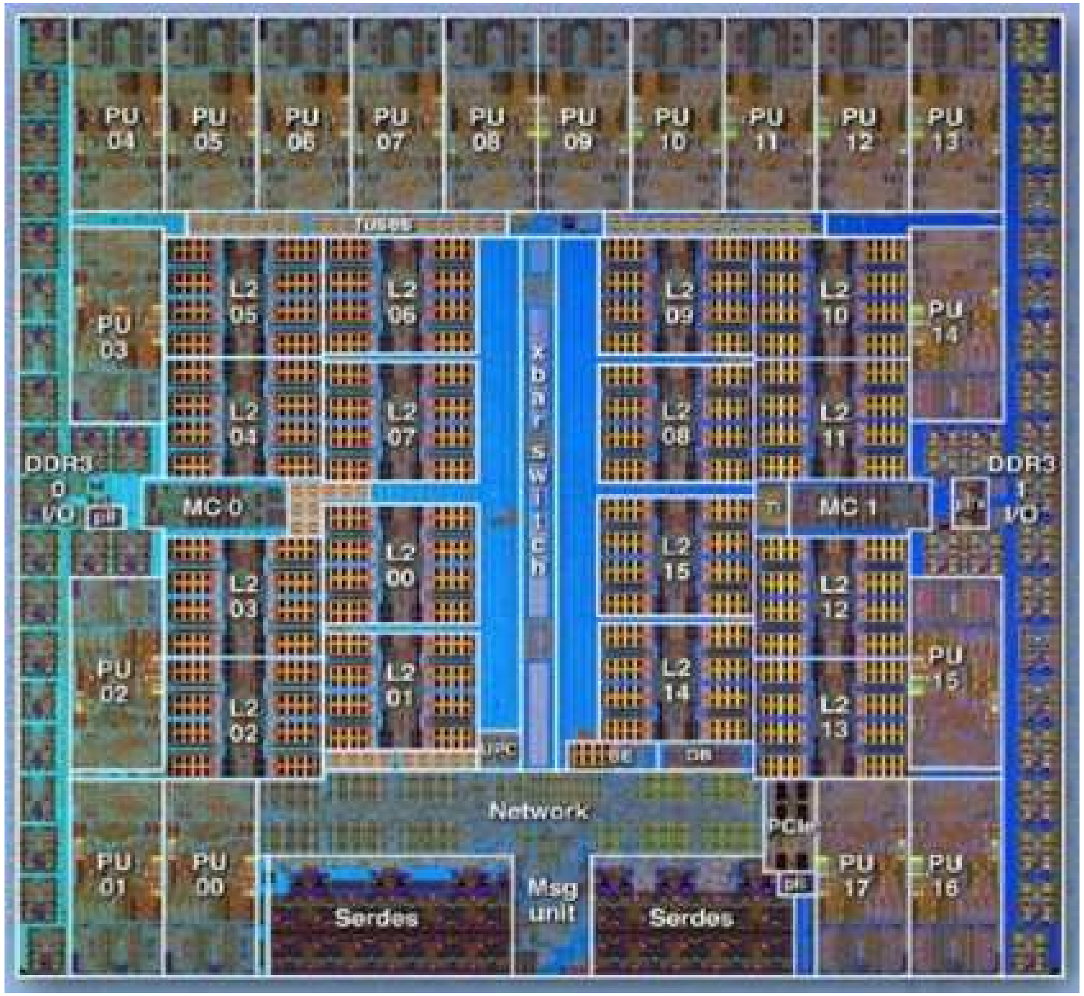
Table 1. Baseline SARC simulation parameters.

Parameter	Value
Clock frequency	3.2 GHz
Memory controllers	4 × 2 DDR3 channels
Channel bandwidth	12.8 Gbytes per second (GBps) (DDR3-1600)
Memory latency	Real DDR3-1600
Memory interface controllers (MICs) policy	Closed-page, in-order processing
Shared L2 cache	128 Mbytes (32 blocks æ 4 Mbytes), 4-way associative
L2 cache latency	40 cycles
Local store	256 Kbytes, 6 cycles
L0 cache	32 Kbytes, 3 cycles
Interconnection links	8 bytes/cycle (25.6 GBps)
Intracluster network on chip (NoC)	2-bus (51.2 GBps)
Global NoC	16-bus (409.6 GBps)

More recently (20110908), Dimitris Nikoloupos confirmed me that there won’t be anymore the LS as it will be integrated in the L2.

The SARC architecture, IEEE micro, Oct. 2010, vol. 30, n. 5, pp. 16-29

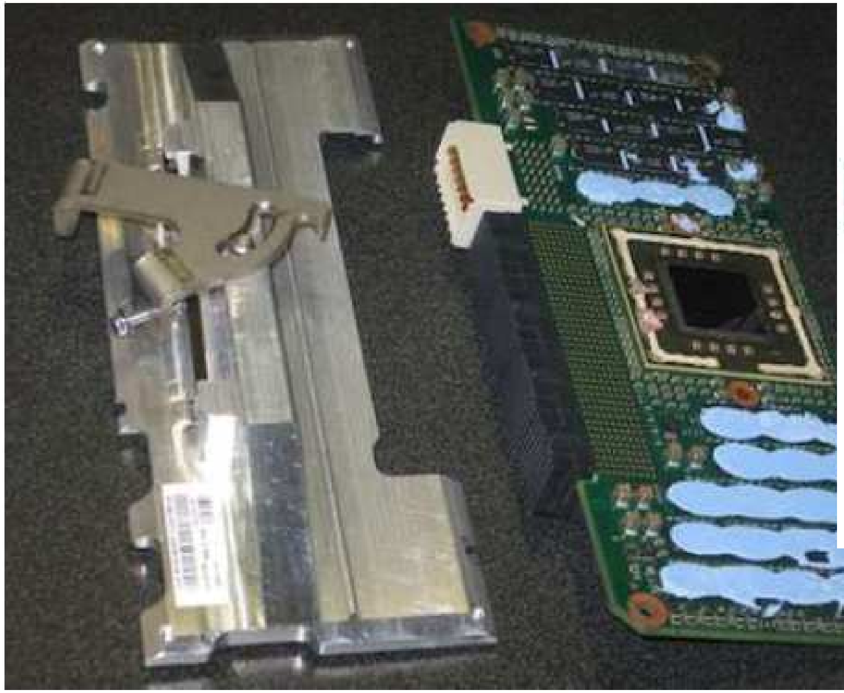
IBM BlueGene/Q

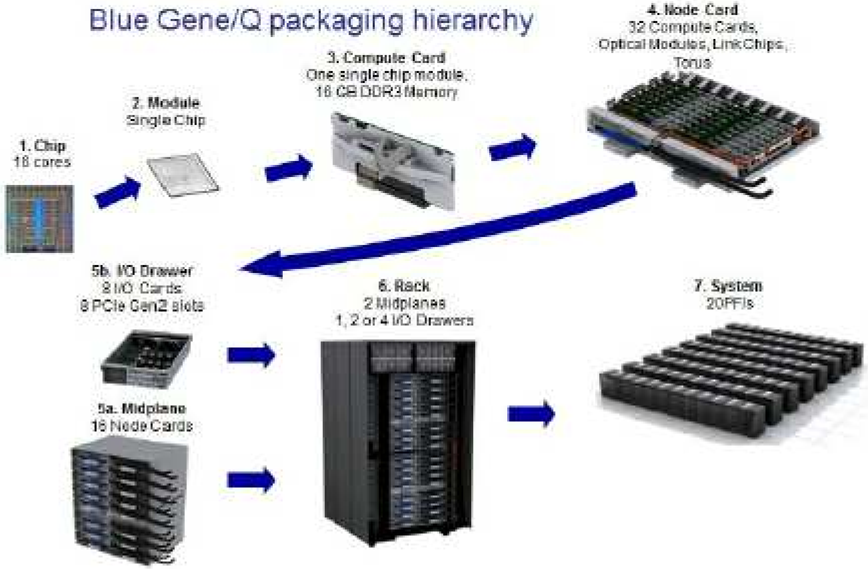


Rather than replicate more than 18 of these large cores, IBM chose to give each core hardware support for four concurrent threads, so under ideal circumstances the chip can behave almost as a 64-CPU system

- Ruud Haring, The IBM Blue Gene/Q Compute chip+SIMD floating-point unit, HotChips Symposium, Aug 2011.

BlueGene/Q module and system



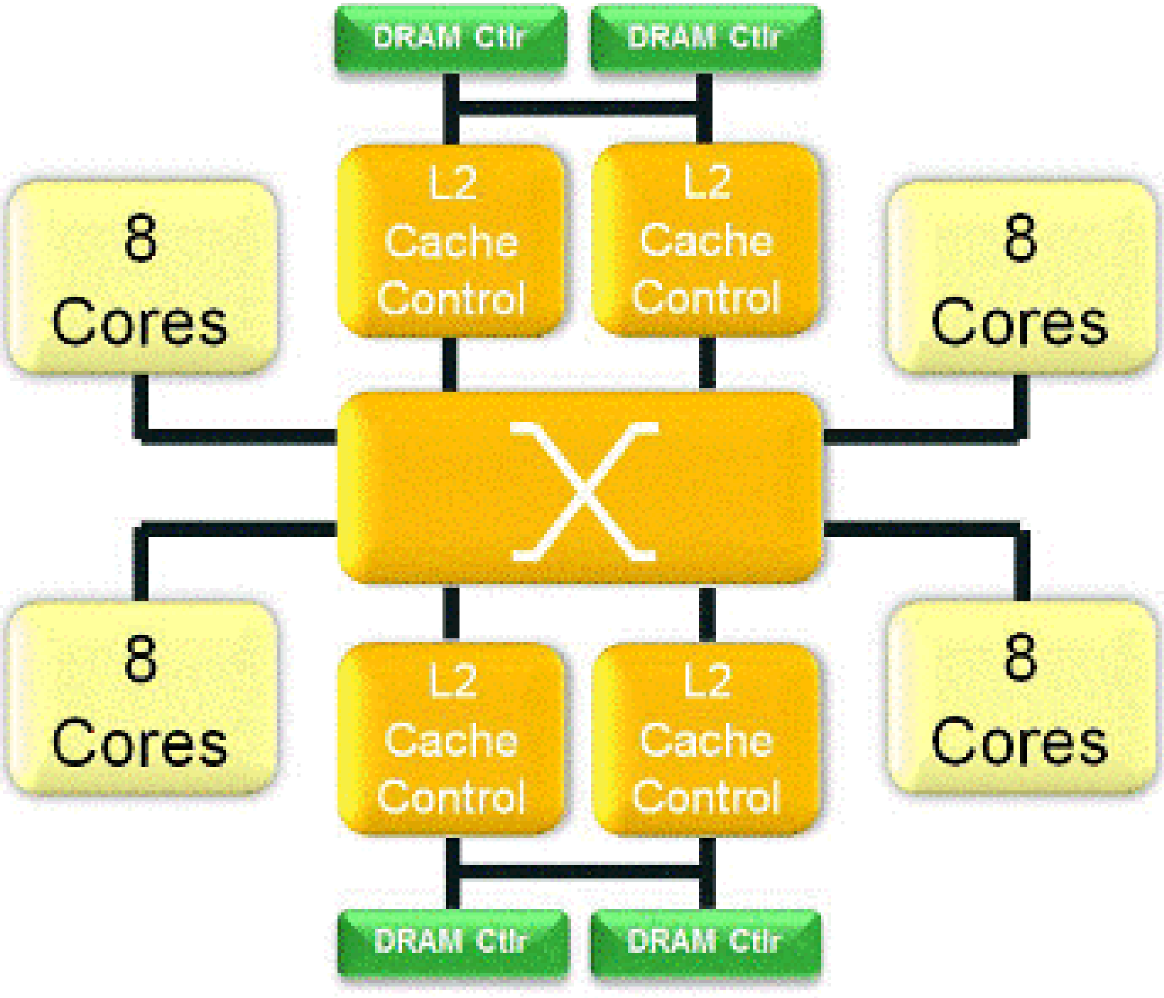


The BlueGene/Q module with DDR3 memory, five links and a water cooling system.

Thanks to the 64-bit support, the modules can now run 8 or 16 GB of DDR3 memory. Five links (2 GB/s per direction) connect each module to its neighbors, making it possible to create different 5D topologies. Half a rack with 8192 BlueGene/Q cores has already proven its capabilities in the Linpack benchmark. With 65.3 Tflops, the test system from the Thomas J. Watson Research Center scored 115th place in the new Top500 list. Its power consumption of 38.8 kW represented a new record value for energy efficiency at close to 1700 Mflops/watt. The Sequoia is supposed to get 96 fully equipped racks, which are supposed to deliver 20 Pflops of theoretical peak performance at the end of 2012.

A. Stiller – translation from the original article from German in c’t by Marcel Sieslack

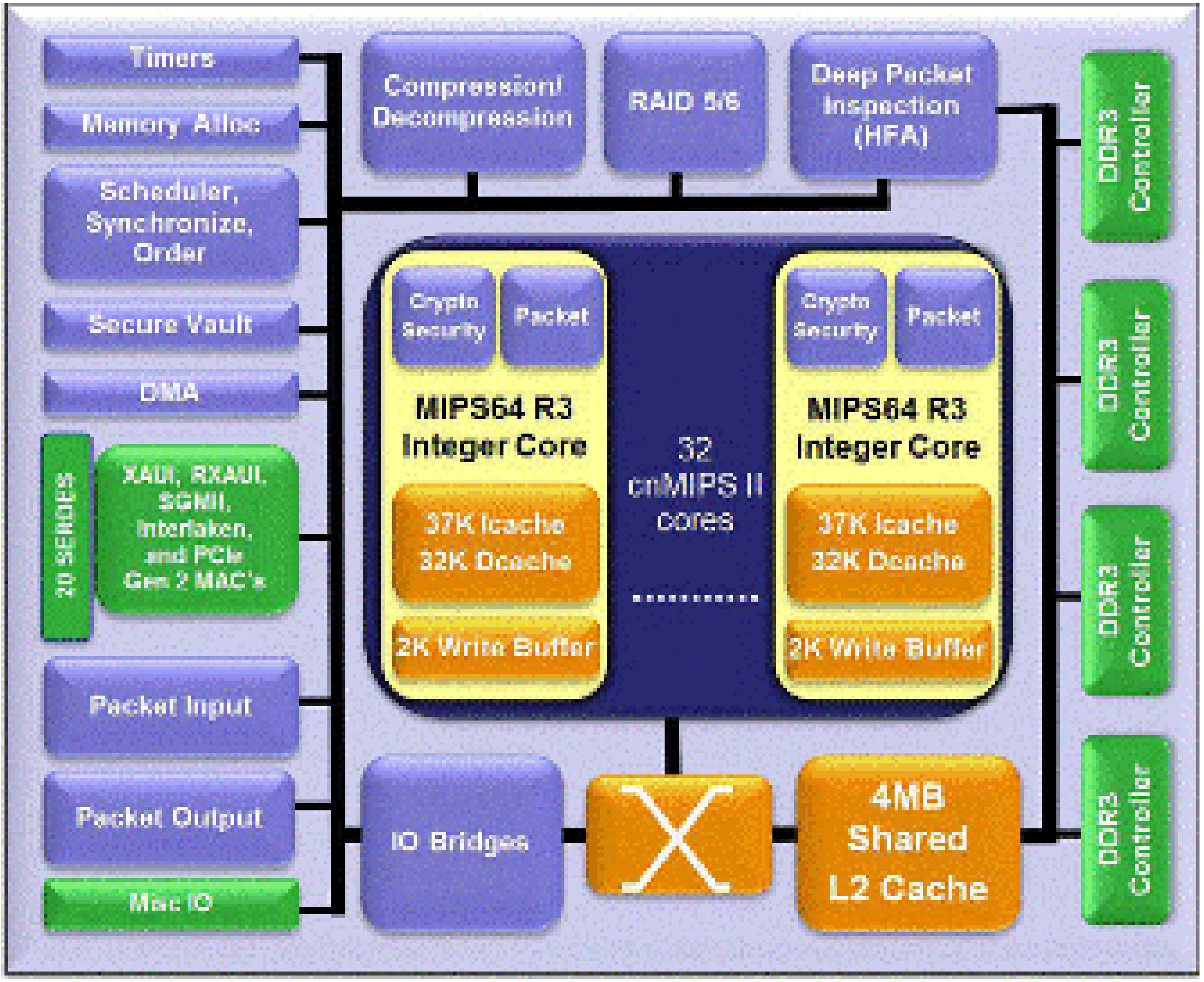
Cavium – Octeon II CN6880



Cavium relies primarily on locks for synchronization, assisted by facilities in the scheduler hardware

Cavium OCTEON II CN6880 Multi-Core MIPS64 Processor, HotChips Symposium, Aug. 2011.

Octeon II (2)


Timers
Memory Alloc
Scheduler, Synchronize, Order
Secure Vault
DMA
20 SERDES
XAUI, RXAUI, SGMII, Interlaken, and PCIe Gen 2 MAC's
Packet Input
Packet Output
Mac IO
Compression/ Decompression
RAID 5/6
Deep Packet Inspection (HFA)
Crypto Security
Packet
MIPS64 R3 Integer Core
37K Icache 32K Dcache
2K Write Buffer
32 cnMIPS II cores
Crypto Security
Packet
MIPS64 R3 Integer Core
37K Icache 32K Dcache
2K Write Buffer
IO Bridges
4MB Shared L2 Cache
DDR3 Controller
DDR3 Controller
DDR3 Controller
DDR3 Controller

Godson-T

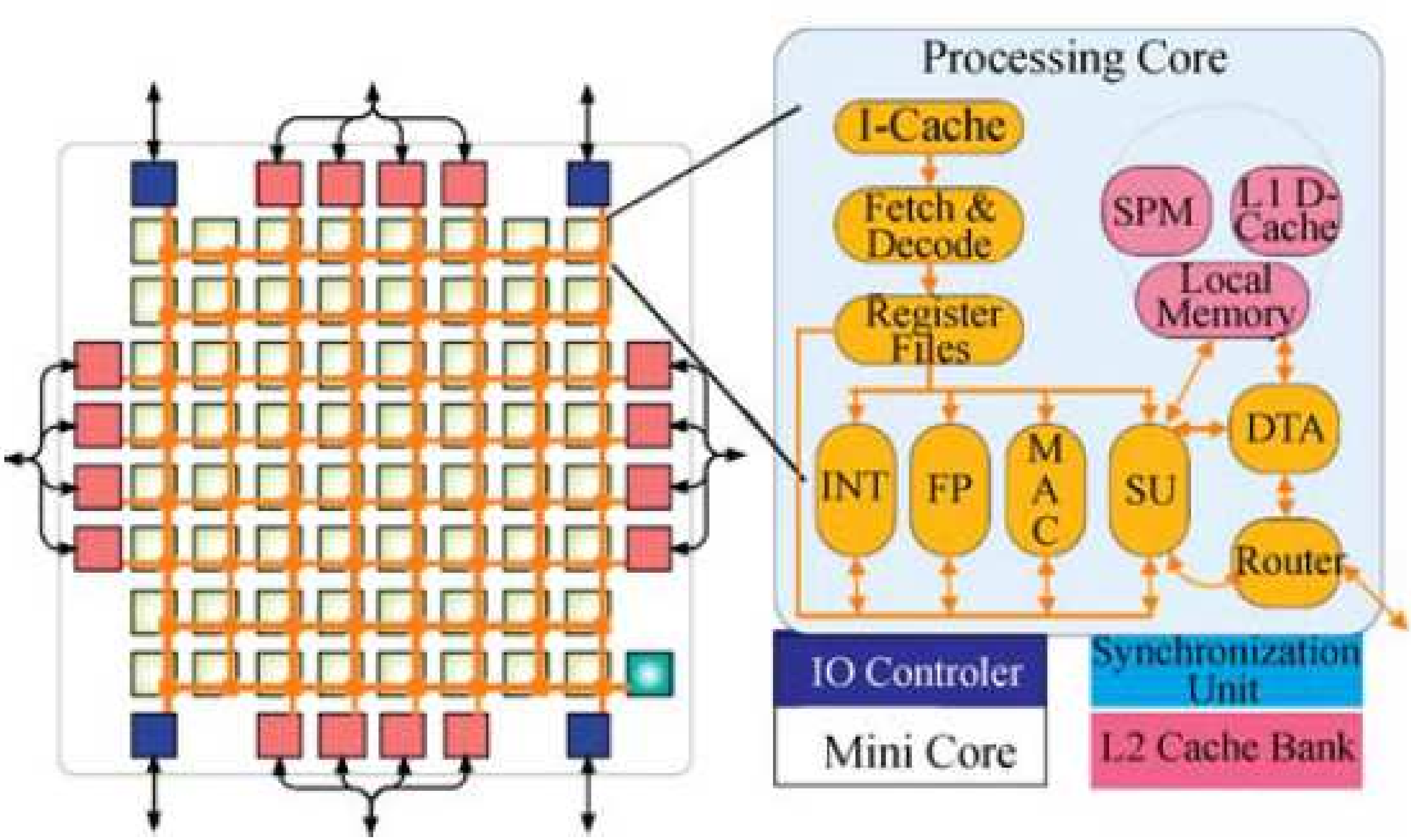
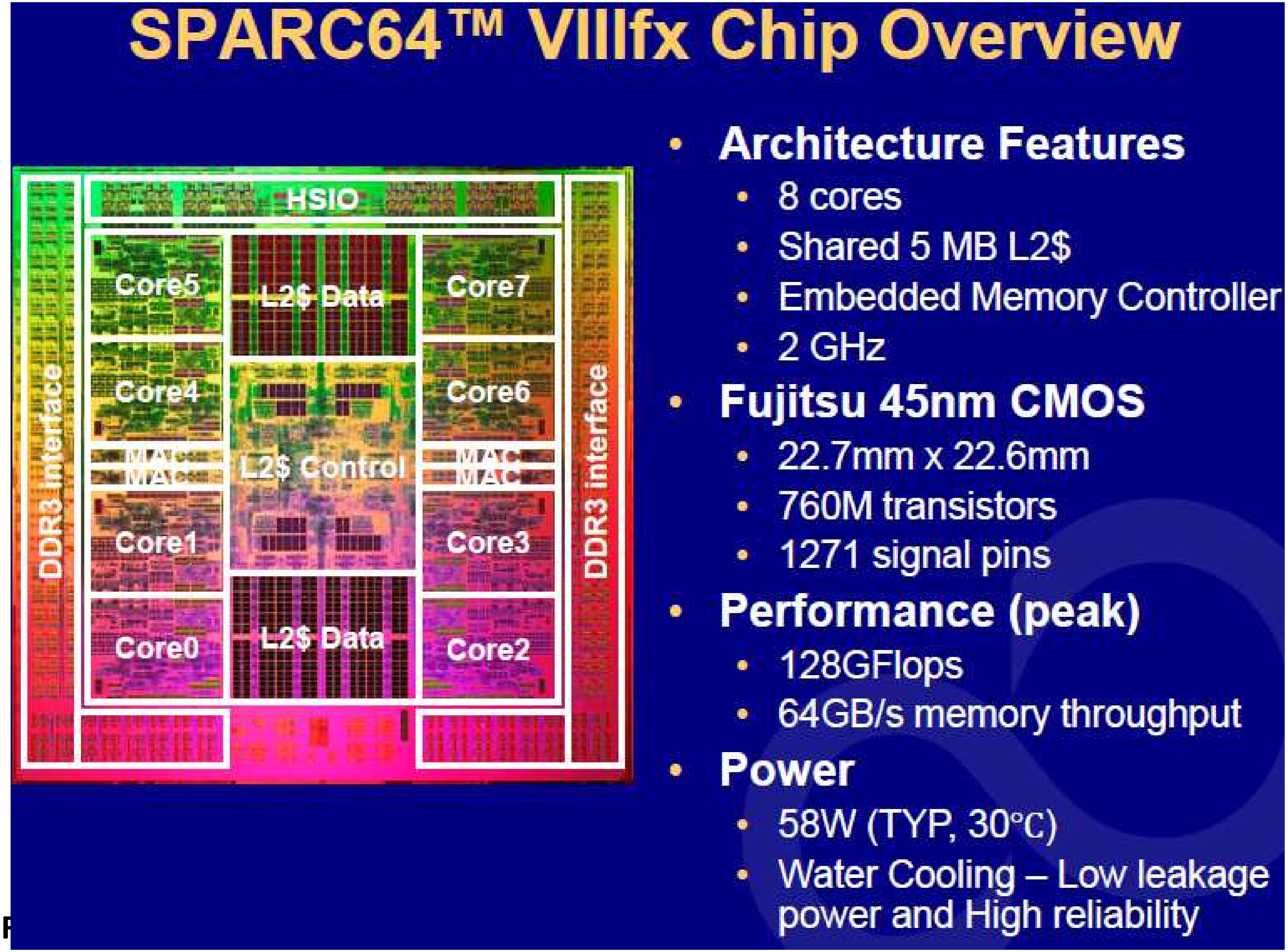


Fig.1. Overview of Godson-T.

Cui HM, Wang L, Fang DR *et al.* Landing stencil code on Godson-T. JOURNAL OF COMPUTER SCIENCE ANDTECHNOLOGY 25(4): 886–894 July 2010.

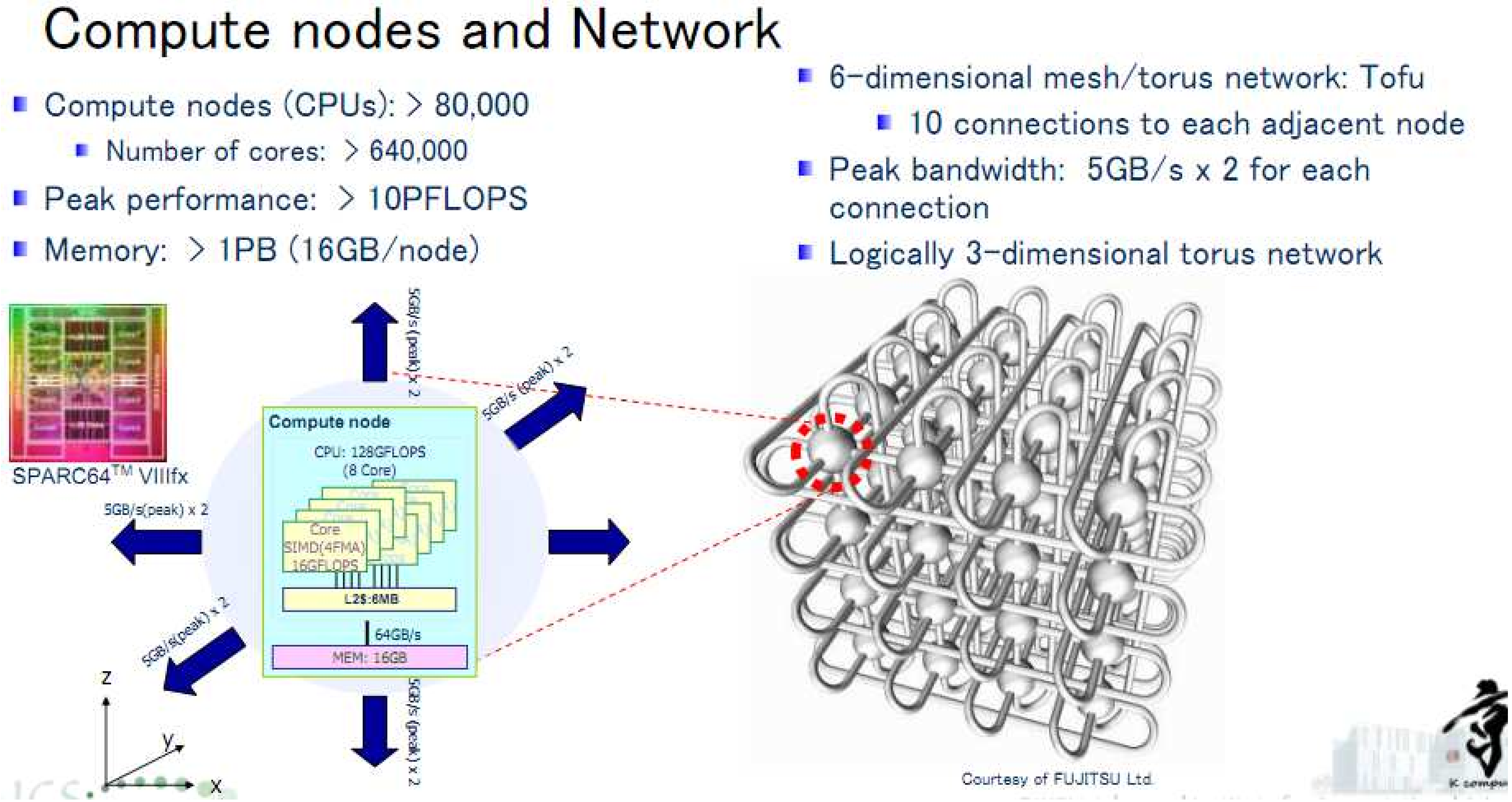


SPARC64™ VIIIfx Chip Overview

- **Architecture Features**
  - 8 cores
  - Shared 5 MB L2$
  - Embedded Memory Controller
  - 2 GHz
- **Fujitsu 45nm CMOS**
  - 22.7mm x 22.6mm
  - 760M transistors
  - 1271 signal pins
- **Performance (peak)**
  - 128GFlops
  - 64GB/s memory throughput
- **Power**
  - 58W (TYP, 30℃)
  - Water Cooling – Low leakage power and High reliability

TER

K-Computer



Compute nodes and Network

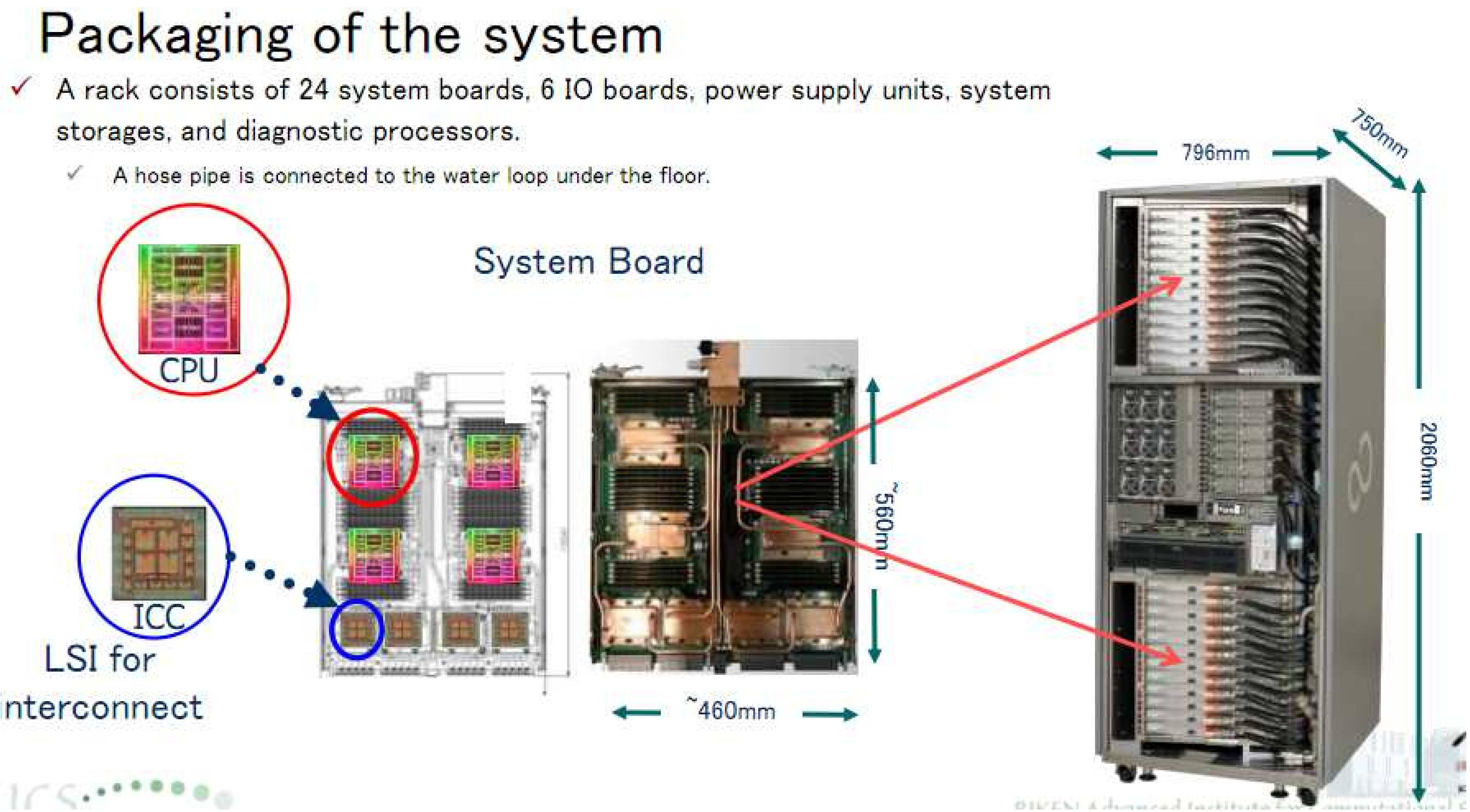
- Compute nodes (CPUs): > 80,000
  - Number of cores: > 640,000
- Peak performance: > 10PFLOPS
- Memory: > 1PB (16GB/node)

- 6-dimensional mesh/torus network: Tofu
  - 10 connections to each adjacent node
- Peak bandwidth: 5GB/s x 2 for each connection
- Logically 3-dimensional torus network

Courtesy of FUJITSU Ltd.

Mitsuo Yokokawa, Fumiyoshi Shoji, Atsuya Uno, Motoyoshi Kurokawa, and Tadashi Watanabe. 2011. *The K computer*: Japanese next-generation supercomputer development project. In*Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design* (ISLPED '11).

K-Computer (2)



Packaging of the system

- ✓ A rack consists of 24 system boards, 6 IO boards, power supply units, system storages, and diagnostic processors.
  - ✓ A hose pipe is connected to the water loop under the floor.

- S. Fumiyoshi, The K Computer: Project Overview

IBM Cyclops-64

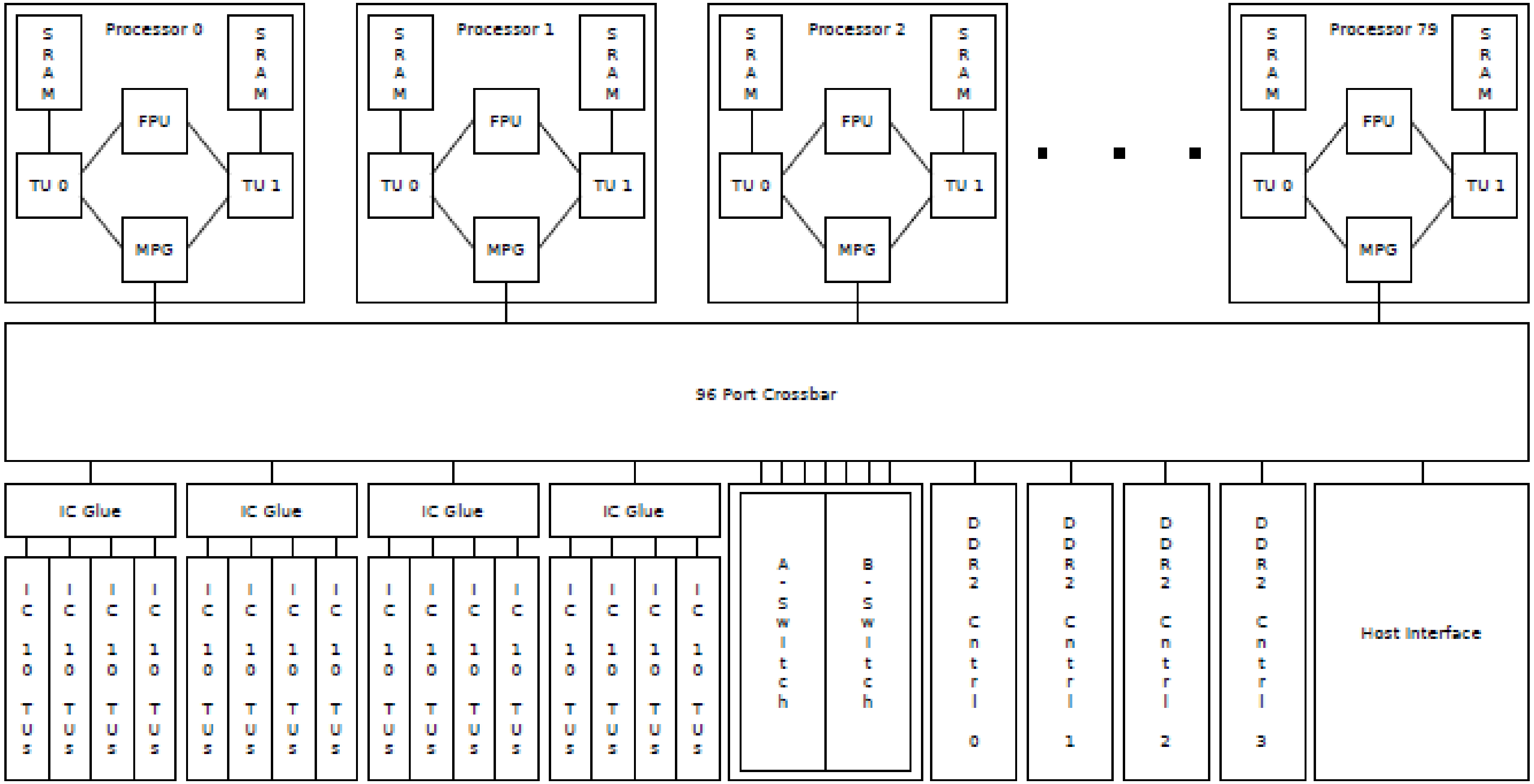


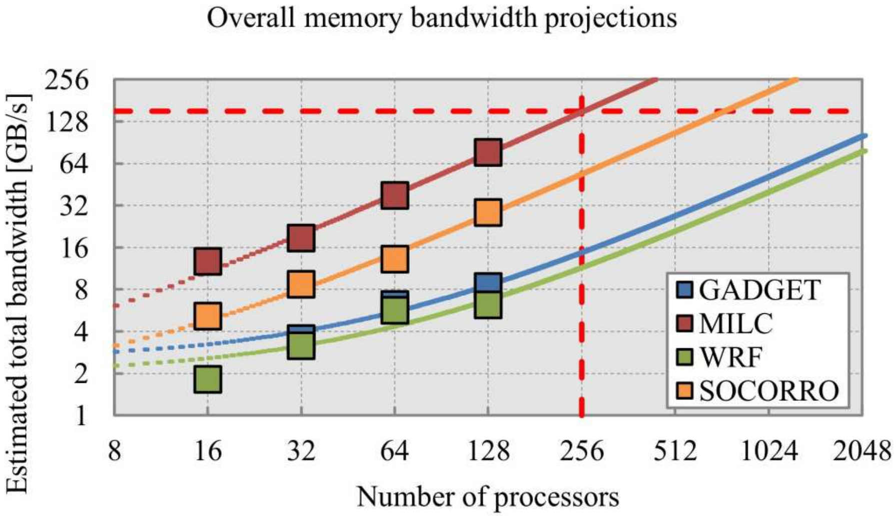
Figure 1: IBM Cyclops-64 (C64) Many-Core Architecture: The architecture consists of 80 processors (Processor 0 -79). Each processor has two Thread Units (TUs) called TU 0 and TU 1. Both share one Floating-Point Unit (FPU) and one crossbar port (MPG). Each TU is connected to a SRAM bank, which can be accessed by all other TUs via the crossbar. Ten TUs share one Instruction Cache (IC). The system has four on-chip DDR2 memory controllers to access off-chip memory. The A-Switch is used to connect to the six surrounding neighbors in a 3D-mesh network.

Juergen Ributzka, Yuhei Hayashi, Joseph B. Manzano, and Guang R. Gao. 2011. The elephant and the mice: the role of non-strict fine-grain synchronization for modern many-core architectures. In *Proceedings of the international conference on Supercomputing* (ICS '11). ACM, New York, NY, USA, 338-347.

TERAFLUX RESEARCH OVERVIEW

WP2 - APPLICATIONS

- MPI applications
- Measures with executions with 16, 32, 64, 128 procs.
    - Linear regression projections
- Off-chip Memory Bandwidth
- – 1K cores will need more than 256GB/s sustained bandwidth
- – 3x than current DDR3


Overall memory bandwidth projections

- Total memory footprint for MPI applications
- – Total memory footprint increases with the number of processors
- – Manycores with more than 100 cores will require a few dozens GBs of main memory
- Alternative programming models are required to deal with memory requirements for scalability

M. Pavlovic, et al. "On the Memory System Requirements of Scientific Applications: Four Case Studies", In IEEE Intl. Symp. On Workload Characterization (IISWC), Nov 2011.

WP3 – PROGRAMMING MODEL

Transactions and Dataflow additions to Scala

- Modified to the Scala compiler to include transactional constructs – surveyed other possibilities using closures
- Runtime STM support
- Statically Typed Dataflow Library
- Reimplementation of the Scala parallel collection using dataflow plus transactions
- Analysis for Lee-TM of benefits of Dataflow plus transactions

http://apt.cs.man.ac.uk/projects/TERAFLUX/MUTS/

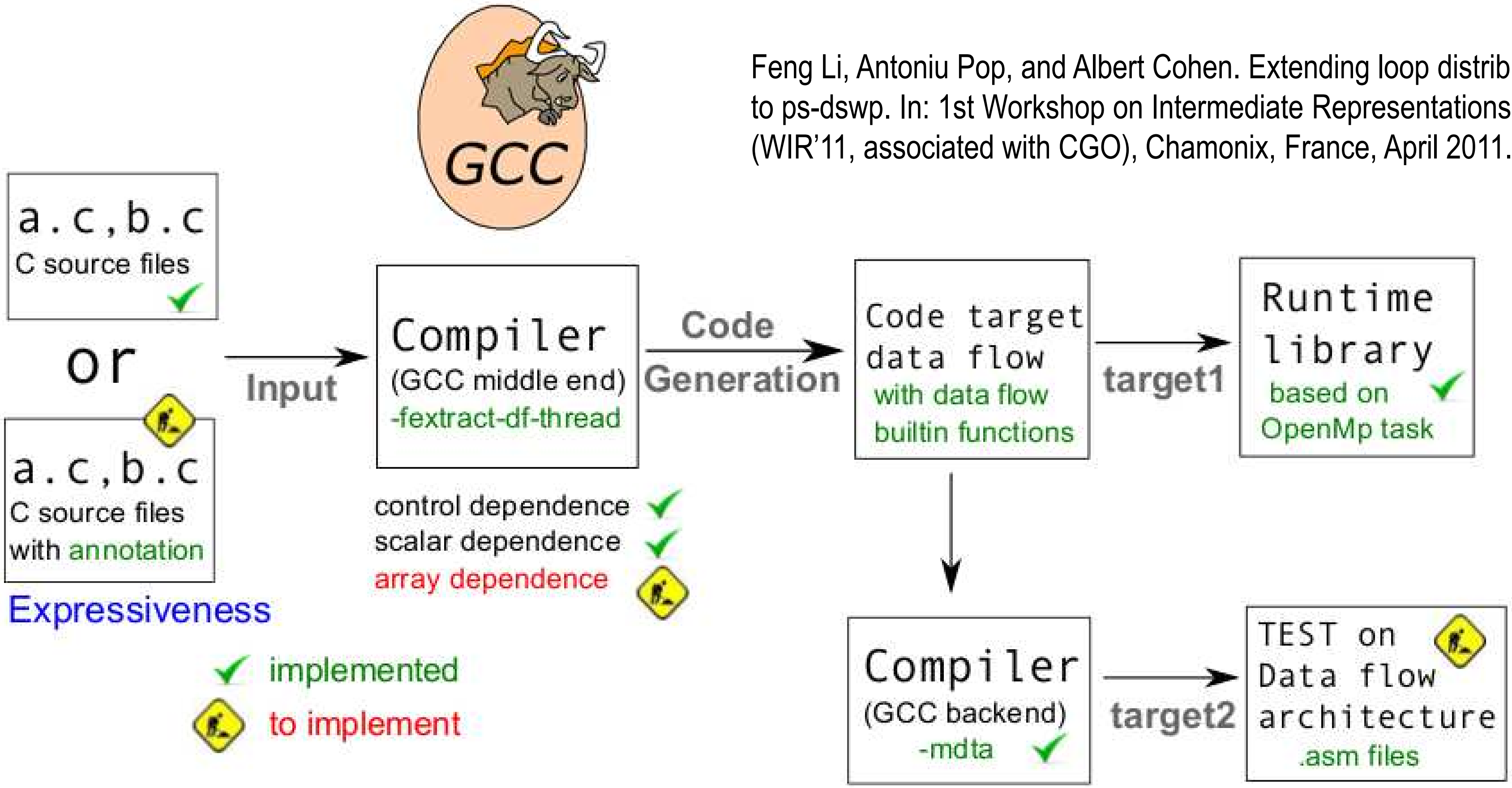
Daniel Goodman, Behram Khan, Salman Khan, Chris Kirkham, Mikel Lujan and Ian Watson. MUTS: Native Scala Constructs for Software Transactional Memory. In: Scala Days Workshop, Stanford, California, June 2-3, 2011.

WP4 – COMPILATION TOOLS

Compilation for Dataflow Threads

Automatic DF Thread Extraction

Feng Li, Antoniu Pop, and Albert Cohen. Extending loop distribution to ps-dswp. In: 1st Workshop on Intermediate Representations (WIR'11, associated with CGO), Chamonix, France, April 2011.



a.c,b.c
C source files ✓

or

🚧 a.c,b.c
C source files
with annotation

Expressiveness

Input

Compiler
(GCC middle end)
-fextract-df-thread

control dependence ✓
scalar dependence ✓
array dependence 🚧

Code Generation

Code target data flow
with data flow builtin functions

target1

Runtime library
based on ✓
OpenMp task

Compiler
(GCC backend)
-mdta ✓

target2

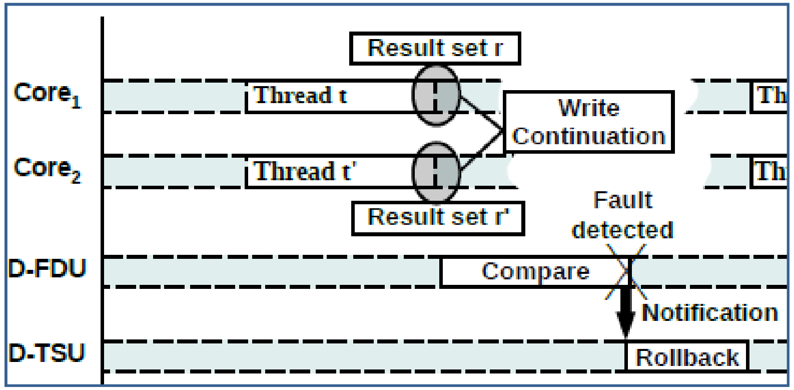
TEST on 🚧
Data flow architecture
.asm files

✓ implemented

🚧 to implement

WP5 - RELIABILITY

- DOUBLE EXECUTION detects control flow AND data errors
- Runs each thread twice, once as a leading thread ***t*** and second time as a trailing thread ***t'***
- The duplicated threads can run on the same core or on different cores of the same node/cluster



- Each execution generates signature of output results
- At completion compare the two signatures, if consistent, the D-TSU writes its results to subsequent thread frames
- If not, no commitment and recovery

Sebastian Weis, Arne Garbade, Julian Wolf, Bernhard Fechner, Avi Mendelson, Roberto Giorgi, and Theo Ungerer. A Fault Detection and Recovery Architecture for a Teradevice Dataflow System. In Data-Flow Execution Models for Extreme Scale Computing (DFM) 2011 Workshop Proceedings. IEEE Computer Society, 2011

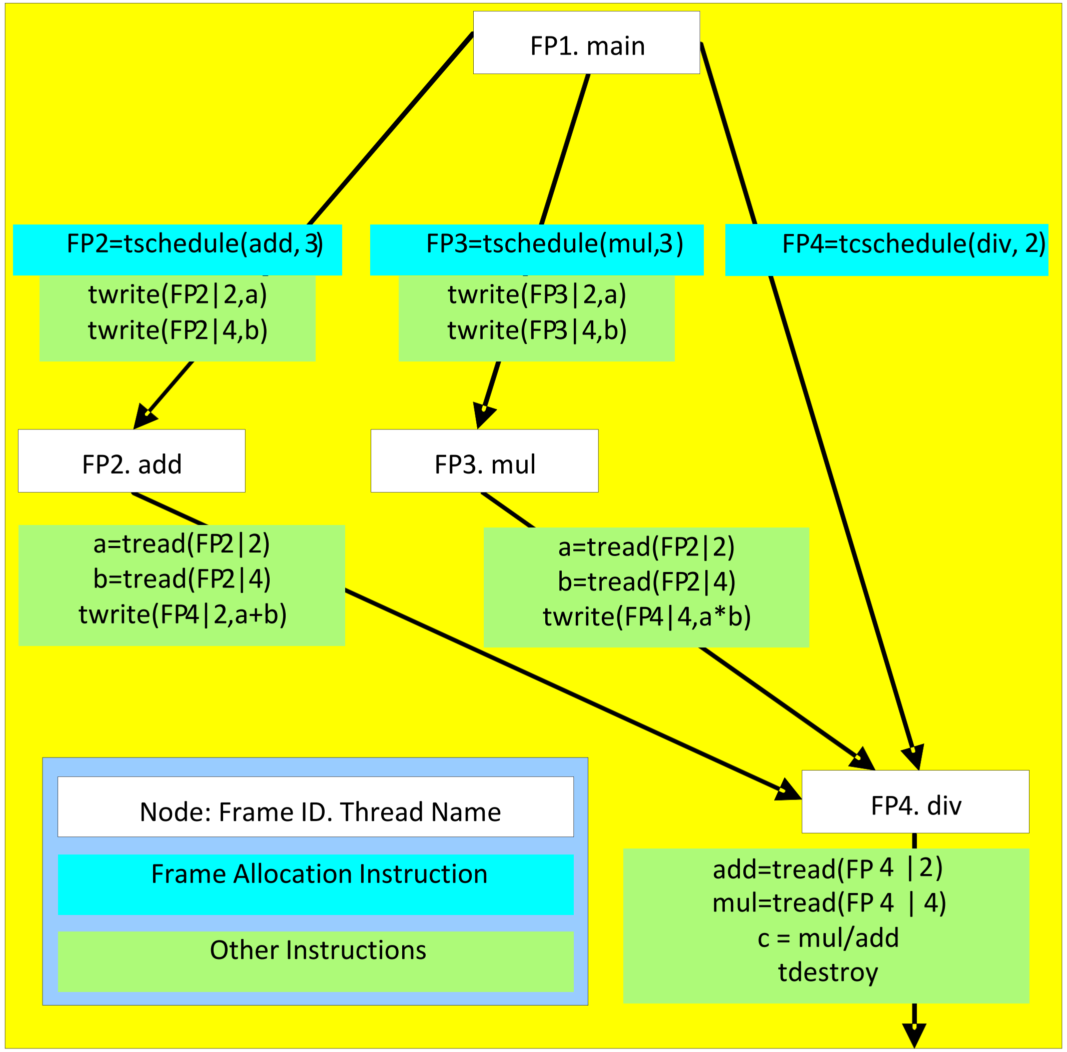
WP6 - ARCHITECTURE

.c (sequential)

```
void main() {
  int a = 4;
  int b = 4;
  int add, mul, c;

  add = a + b;
  mul = a * b;
  c = mul/add;
}
```

.x86 assembly (parallel)



FP1. main

FP2=tschedule(add, 3)
twrite(FP2|2,a)
twrite(FP2|4,b)

FP3=tschedule(mul,3)
twrite(FP3|2,a)
twrite(FP3|4,b)

FP4=tcschedule(div, 2)

FP2. add

a=tread(FP2|2)
b=tread(FP2|4)
twrite(FP4|2,a+b)

FP3. mul

a=tread(FP2|2)
b=tread(FP2|4)
twrite(FP4|4,a*b)

FP4. div

add=tread(FP 4 |2)
mul=tread(FP 4 | 4)
c = mul/add
tdestroy

Node: Frame ID. Thread Name

Frame Allocation Instruction

Other Instructions

.x86 assembly (parallel)

```
main:   movq        $4,     %R8
        movq        $4,     %R9
        movq        $1,     %RAX
        cmpq        $1,     %RAX
        TSCHEDULE   $add,   $3,     %R10
        TSCHEDULE   $mult,  $3,     %R11
        TSCHEDULE   $div,   $2,     %R12
        TWRITE      %R8,    %R10,   $2
        TWRITE      %R9,    %R10,   $3
        TWRITE      %R12,   %R10,   $4
        TWRITE      %R8,    %R11,   $2
        TWRITE      %R9,    %R11,   $3
        TWRITE      %R12,   %R11,   $4
        TDESTROY
```

```
add:    TREAD    $2,     %R8
        TREAD    $3,     %R9
        TREAD    $4,     %R10
        movq     %R8,    %R11
        addq     %R9,    %R11
        TWRITE   %R11,   %R10,   $2
        TDESTROY
```

```
mult:   TREAD    $2,     %R8
        TREAD    $3,     %R9
        TREAD    $4,     %R10
        movq     %R8,    %RAX
        mulq     %R9
        movq     %RAX,   %R11
        TWRITE   %R11,   %R10,   $3
        TDESTROY
```

```
div:    TREAD    $2,     %R8
        TREAD    $3,     %R9
        movq     $0,     %RDX
        movq     %R9,    %RAX
        divq     %R8
        movq     %RAX,   %R10
        TDESTROY
```

T* (or T86) ISE: TSCHEDULE/TDESTROY

	T* INSTRUCTIONS	**IMPLIED COMPILER TARGET**
Synopsis	**TSCHEDULE RS1, RS2, RD**	**TSCHEDULE(<IP>, <SC>, &<frame_pointer>)**
Description	This instruction allocates the resources (a DF-frame of size RS2 words and a corresponding entry in the Distributed Thread Scheduler – or DTS) for a new DF-thread and returns its Frame Pointer (FP) in RD. RS1 specifies the Instruction Pointer (IP) of the first instruction of the code of this DF-thread and RS2 specifies the Synchronization Count (SC). Finally the zero flag is logically negated.	
Notes	The allocated DF-thread is not executed until its SC reaches 0. The TSCHEDULE can be conditional or non-conditional based on the value stored in the zero flag. If the zero flag is set to 1 then the TSCHEDULE will take effect, otherwise it is ignored. Two subsequent TSCHEDULE implement an if-then-else.	
Synopsis	**TDESTROY**	**TDESTROY**
Description	The thread that invokes TDESTROY finishes and its DF-frame is freed, (the corresponding entry in the Thread Scheduling Unit is also freed).	
Notes	-	

Antoni Portero, Zhibin Yu, Roberto Giorgi, "T-Star (T*): An x86-64 ISA Extension to support thread execution on many cores", *HiPEAC ACACES-2011*, ISBN:978 90 382 17987, Fiuggi, Italy, July 2011, pp. 277-280.

T* (or T86) ISE: TWRITE/TREAD

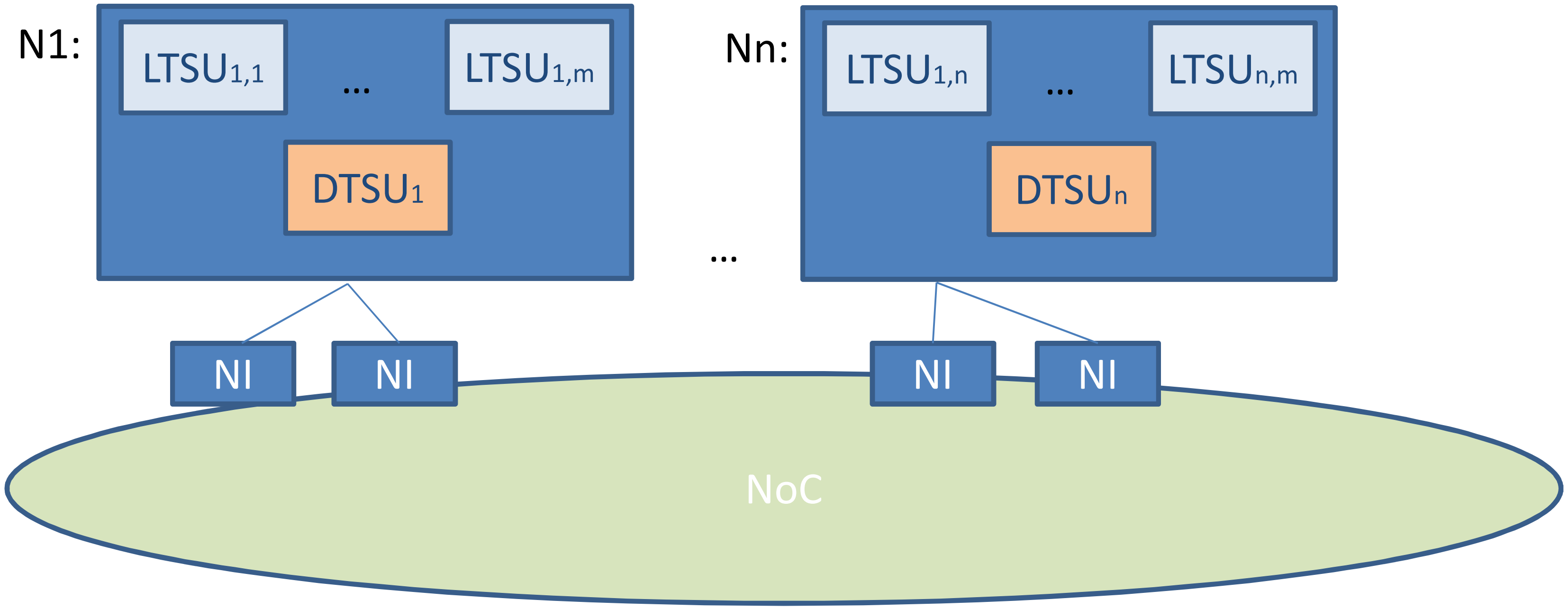
	T* INSTRUCTIONS	IMPLIED COMPILER TARGET
Synopsis	**TWRITE RS, RD, offset**	***(<frame_pointer> + <offset>) = (<source_register>)**
Description	The data in RS is stored into the DF-frame pointed to by RD at the specified offset.	
Notes	Side Effect: The Distributed Thread Scheduler decrements the SC of the corresponding DF-thread entry (located through the FP): $SC_{FP} = SC_{FP}-1$	
Synopsis	**TREAD offset, RD**	**(<destination_register>) = *(<self_frame_pointer> + <offset>)**
Description	Loads the data indexed by 'offset' from the self (current thread) DF-frame into RD.	
Notes	Assumption: the DTS has to load into the register implicitly used by TREAD the value <self_frame_pointer>. In a x86-64 implementation, we can reserve RAX for this purpose.	

T* (or T86) ISE: TALLOC/TFREE

	T* INSTRUCTIONS	IMPLIED COMPILER TARGET
Synopsis	**TALLOC RS1, RS2, RD**	**<pointer> = TALLOC (<size>, <type>)**
Description	Allocates a block of memory of RS1 words. The pointer to it is stored in RD. RS2 specifies the special purpose memory type.	
Notes	The Distributed Thread Scheduler tracks the memory allocated. An implementation can code <type> in the 2 LSB of <size>	
Synopsis	**TFREE (RS)**	**TFREE(<pointer>)**
Description	Frees memory pointed to by RS.	
Notes	The Thread Scheduling Unit tracks the memory deallocated.	

DTS – Distributed Thread Scheduler

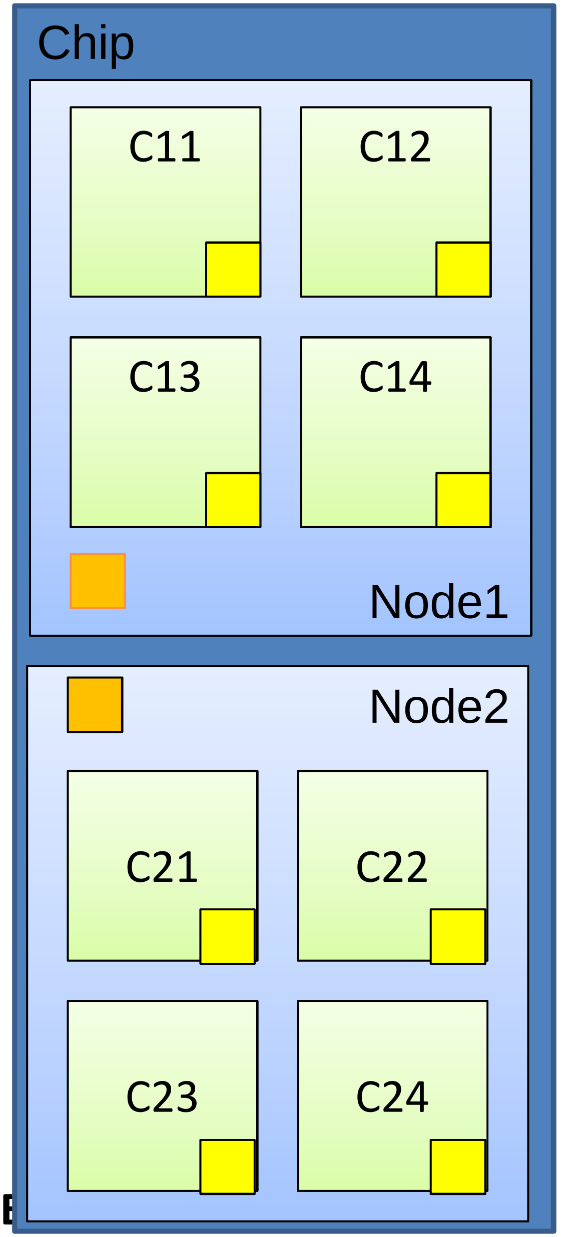
(formerly called TSU)



# of cores = n x m   n = # of nodes   m = # of cores per node

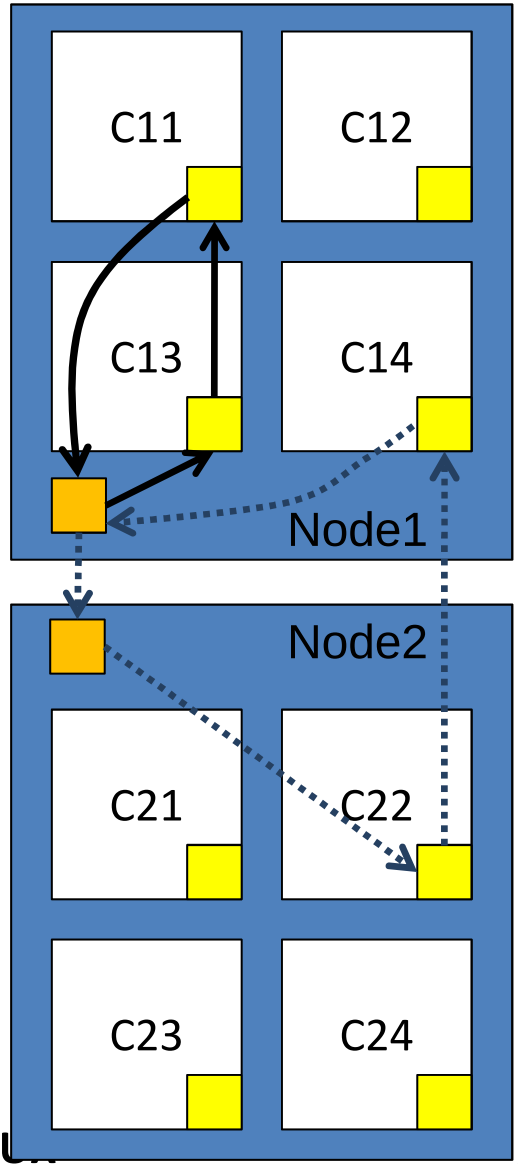
TERAFLUX

Distributed Thread Scheduler
A 2-node x 4-core example



- Core (Cjk)
  - Off-the-shelf cores (may include L1, L2slice)
  - Minimal ISA extension
- Local Thread Scheduling Unit (LTSU)
  - Manages threads on this Core 🟨
- Distributed Thread Scheduling Unit (DTSU) 🟧
  - Distributes threads among Nodes
- Node
  - Groups Cores+Resources

Scheduling Example



- LTSU11 → DTSU1:
  - I need a new frame
- DTSU1 → LTSU13:
  - You're available, give one frame to LTSU11
- LTSU13 → LTSU11:
  - Here is a frame you can use

Fast communication → low overhead

What if every PE in the cluster is busy?

- LTSU14 → DTSU1:
  - "I need a new frame"
- DTSU1 → DTSU2:
  - LSE14 needs a frame, I don't have it
- DTSU2 → LTSU22:
  - Give a frame to LSE14

– LTSU22 → LTSU14:
  - Here is a frame you can use

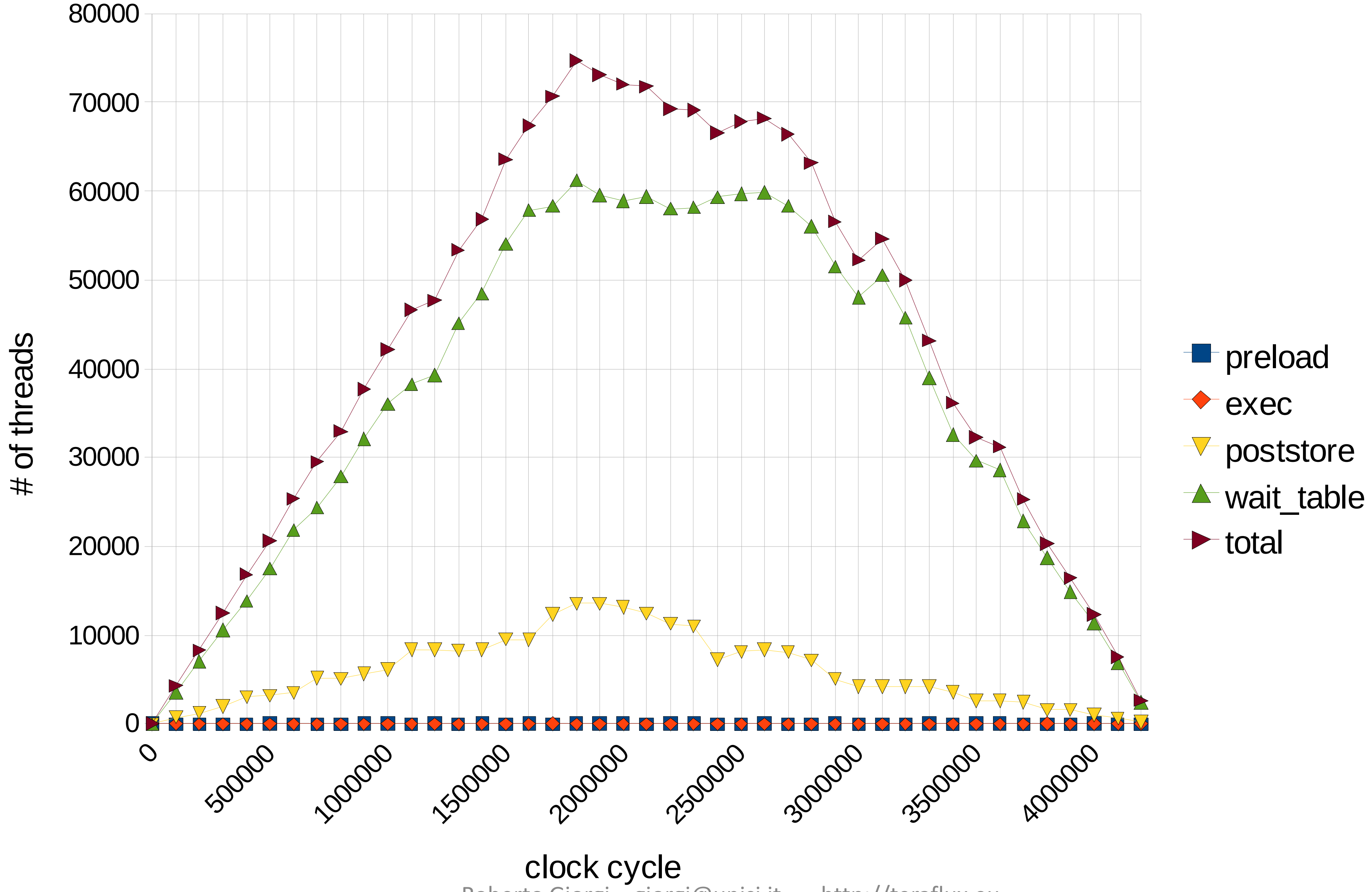
Fine-Grain Thread Scheduling

	Plurality	**CUDA**	**TFlux**	**SSI**	**DTA**
HW/SW	HW + SW	HW + SW	SW	HW	HW
Prog. Model	Custom C language extensions	Custom C language extensions	Custom C preprocessor macros (#pragma)	Standard thread-oriented C libraries (pthreads)	Standard thread-oriented C libraries (pthreads)
Exec. Model	- Pool of RISC processors - Uniform shared memory - Hardware scheduler , synchronizer and load balancer	- Each thread block is split into warps (thread block. - Each thread block is executed by only one multiprocessor. - A multi-processor can execute several blocks concurrently.	High-level threads are mapped to OS threads, using the standard OS programming interfaces as backend.	Subset static interleaved scheduling of fine-grained / coarse-grained threads performed by a hardware Task Scheduling Unit (TSU)	- Decoupling memory and execution activity of non-blocking threads. - Threads are synchronized and communicate each other in a producer-consumer fashion.
Architecture	Complex hardware subsystem: synchronizer, scheduler, load balancer	SIMD, on-chip shared memory	Everything is implemented in software: can run on any general purpose architecture	Hardware scheduler (TSU) + extensions to a VLIW prototype	Thread management and frame memory management implemented in hardware

SSI=Subset Static Interleaved

DTA=Decoupled Threaded Architecture

fib(21): number of threads

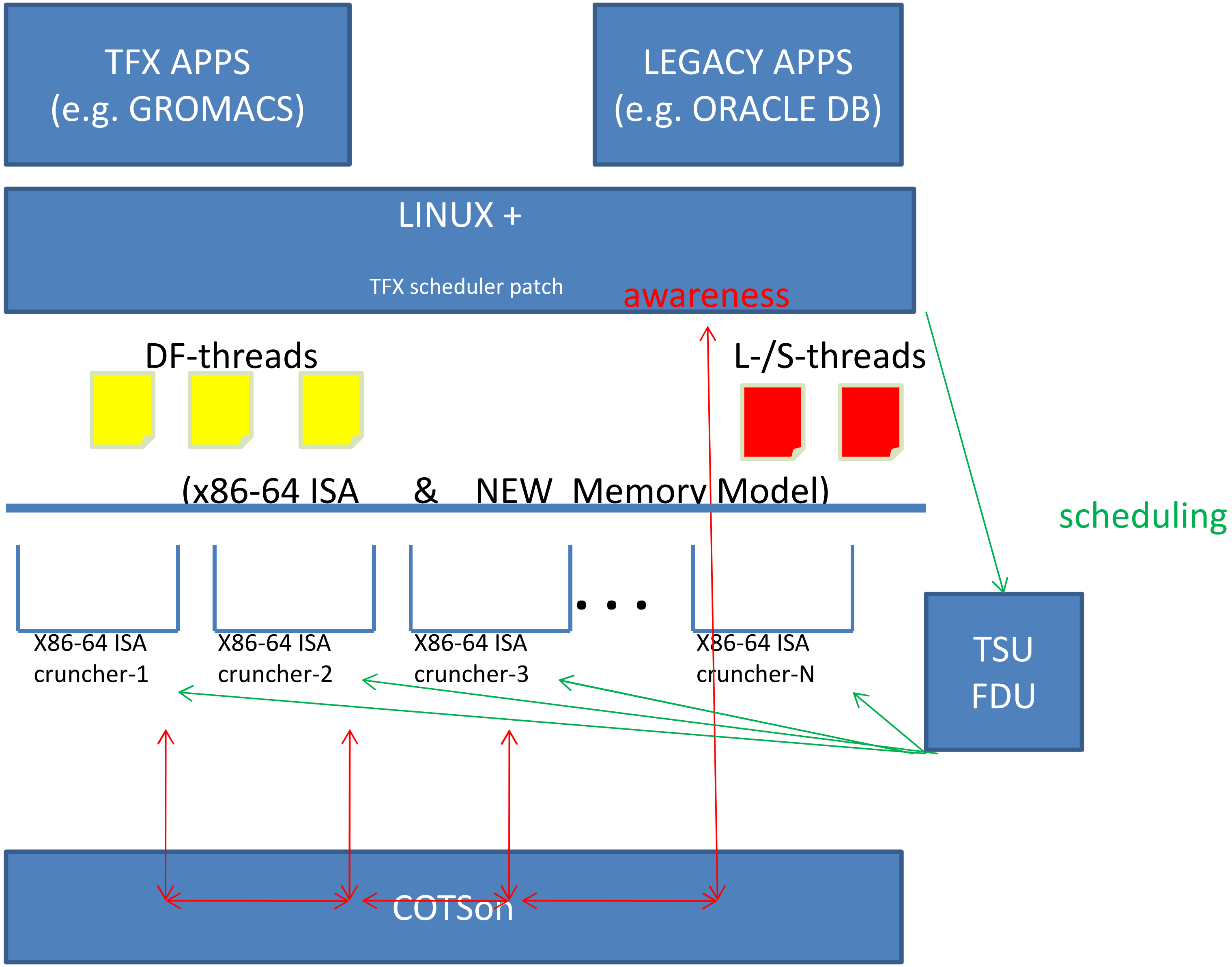


.

WP7: Evaluating a MANY-CORE chip of the future (2020), i.e., 1000+ cores on a chip

Simulator
“illusion”
(SW should
only assume
what seen at
this level)

Simulator
infrastructure



TFX APPS
(e.g. GROMACS)

LEGACY APPS
(e.g. ORACLE DB)

LINUX +

TFX scheduler patch

awareness

DF-threads

L-/S-threads

(x86-64 ISA & NEW Memory Model)

scheduling

X86-64 ISA
cruncher-1

X86-64 ISA
cruncher-2

X86-64 ISA
cruncher-3

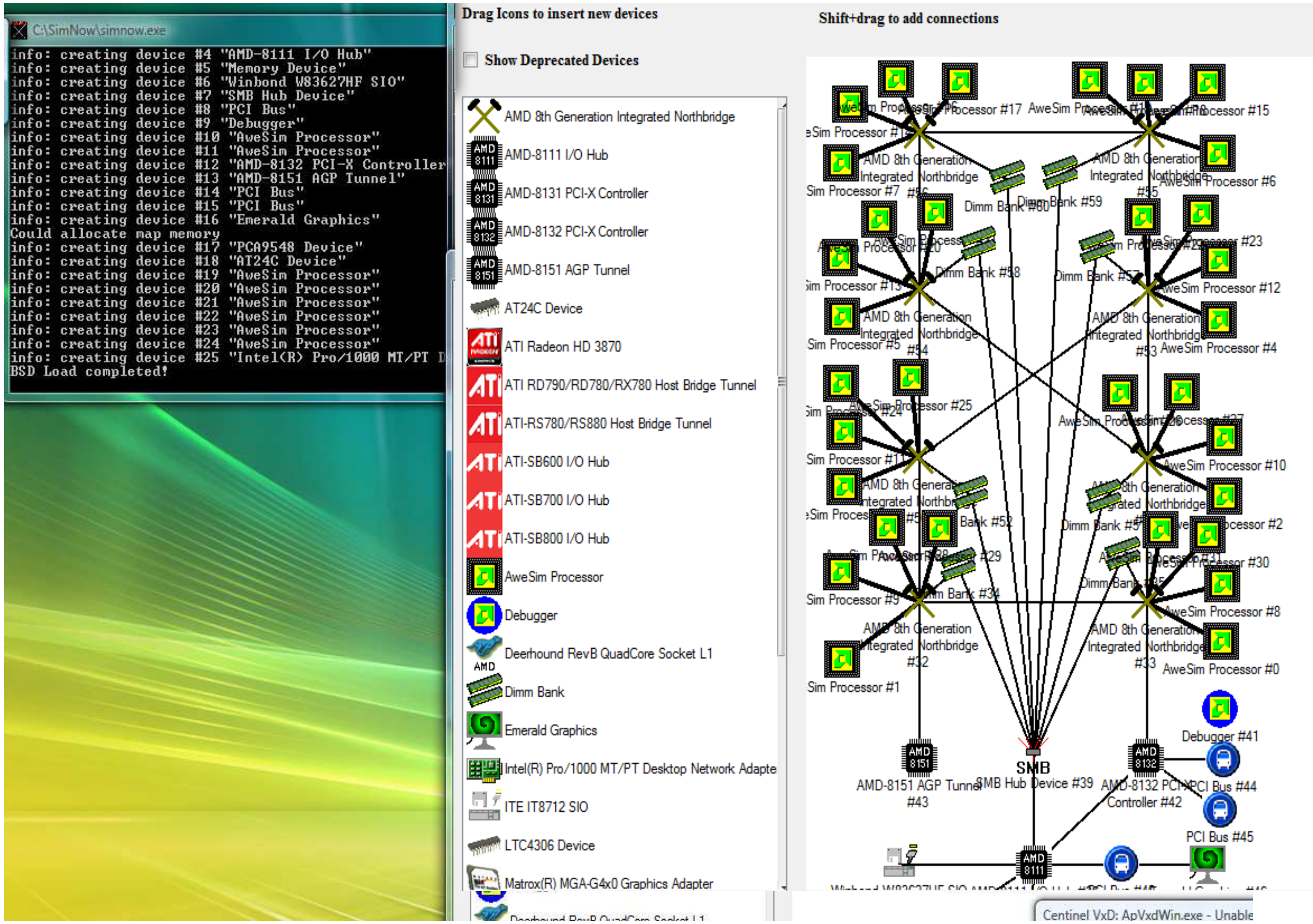
. . .

X86-64 ISA
cruncher-N

TSU
FDU

COTSon

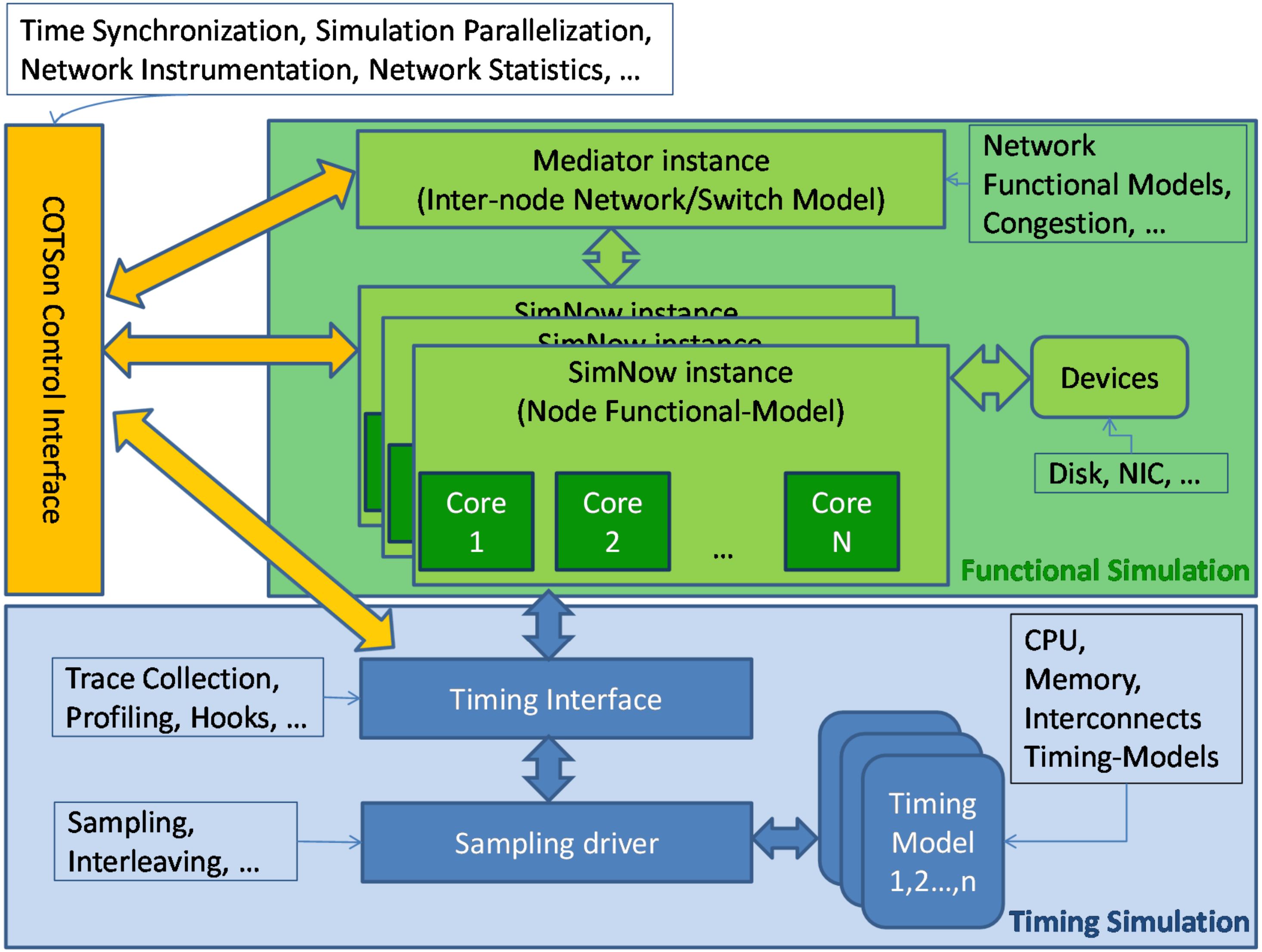
AMD SIMnow and COTSon



C:\SimNow\simnow.exe

```
info: creating device #4 "AMD-8111 I/O Hub"
info: creating device #5 "Memory Device"
info: creating device #6 "Winbond W83627HF SIO"
info: creating device #7 "SMB Hub Device"
info: creating device #8 "PCI Bus"
info: creating device #9 "Debugger"
info: creating device #10 "AweSim Processor"
info: creating device #11 "AweSim Processor"
info: creating device #12 "AMD-8132 PCI-X Controller
info: creating device #13 "AMD-8151 AGP Tunnel"
info: creating device #14 "PCI Bus"
info: creating device #15 "PCI Bus"
info: creating device #16 "Emerald Graphics"
Could allocate map memory
info: creating device #17 "PCA9548 Device"
info: creating device #18 "AT24C Device"
info: creating device #19 "AweSim Processor"
info: creating device #20 "AweSim Processor"
info: creating device #21 "AweSim Processor"
info: creating device #22 "AweSim Processor"
info: creating device #23 "AweSim Processor"
info: creating device #24 "AweSim Processor"
info: creating device #25 "Intel(R) Pro/1000 MT/PT D
BSD Load completed!
```

COTSon Overview

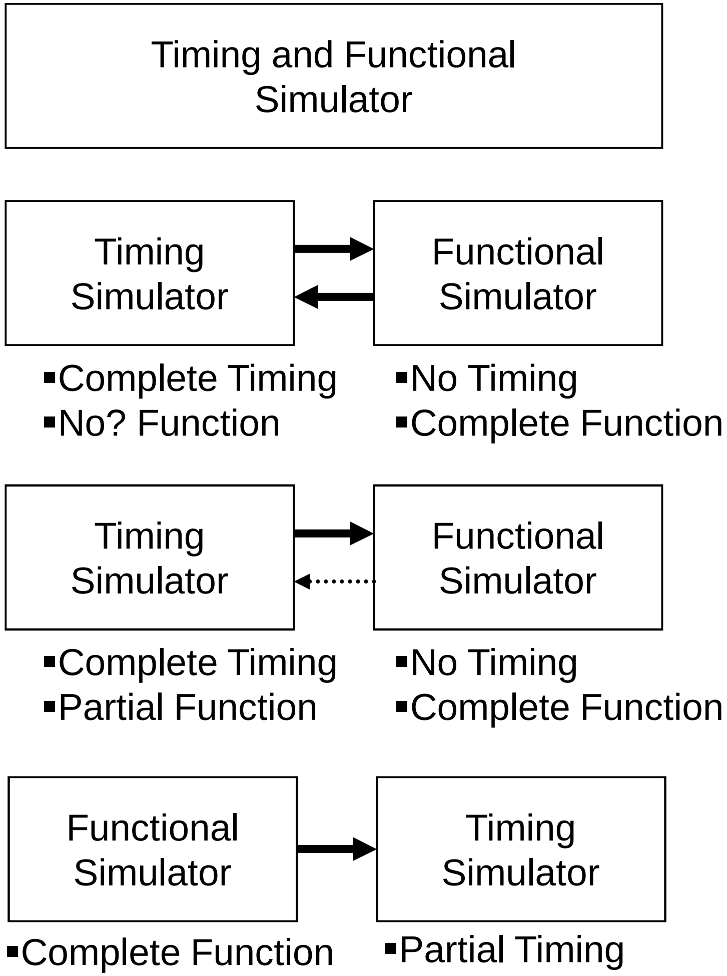

Time Synchronization, Simulation Parallelization, Network Instrumentation, Network Statistics, …
COTSon Control Interface
Mediator instance
(Inter-node Network/Switch Model)
Network Functional Models, Congestion, …
SimNow instance
SimNow instance
SimNow instance
(Node Functional-Model)
Core 1
Core 2
…
Core N
Devices
Disk, NIC, …
Functional Simulation
Trace Collection, Profiling, Hooks, …
Timing Interface
Sampling, Interleaving, …
Sampling driver
Timing Model 1,2…,n
CPU, Memory, Interconnects Timing-Models
Timing Simulation

The ambition of TERAFLUX is however to be able of *changing* such machine in a flexible way, while tackling research challenges on programmability, architectural design and reliability. Therefore, we have the need to stress the COTSon platform, in order to being able to simulate 1000 cores.

Comparison among different approaches for doing research related to 1000-core computing system (Information revised from data of the RAMP project)

	SMP	Cluster	FPGA	Emulator	Simulator
Scalability (1K cores)	C	A	A	A	A
Cost (1K cores)	F(€40M)	C	B(€0.1-0.2M)	A+ (€0.01M)	A+(€0.01M)
Power/Space (Kw, racks)	D (120 kw, 12 racks)	D (120 kw, 12 racks)	A (1.5 kw, 0.3 racks)	A+ (0.1 kw, 0.1racks)	A+ (0.1 kw, 0.1 racks)
Observability	D	C	A+	A+	A+
Reproducibility	B	D	A+	A+	A+
Reconfigurability	D	C	A+	A+	A+
Credibility	A+	A+	B+/A-	F/D	C
Development time	B	B	C	A+	A+
Performance (clock)	A (2GHz)	A(3GHz)	C (0.1 GHz)	B(≈ 0.9 of original)	C(1/10 to 1/1000 SMP)
x86-64 ISA	A+	A+	F	A+	A+
Modifiable	F	F	B	A	A
GPA	D	D	B+/A-	B	A

FUNCTIONAL/TIMING SIMULATION



Timing and Functional Simulator

Integrated (SimOS)

- Complex, no reuse, very slow

Timing Simulator ⇄ Functional Simulator

- ▪Complete Timing
- ▪No? Function

- ▪No Timing
- ▪Complete Function

Timing-Directed (Exec-driven)

+ Timing feedback
- Tight Coupling
- Very slow

Timing Simulator ⇄ Functional Simulator

- ▪Complete Timing
- ▪Partial Function

- ▪No Timing
- ▪Complete Function

Timing-First (Multifacet)

+ Timing feedback
+ Using existing simulators
+ Software development advantages
- Slow

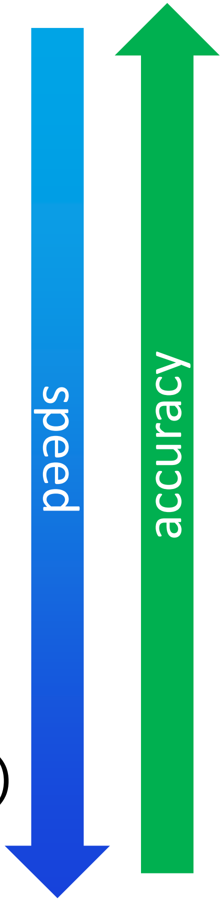
Functional Simulator → Timing Simulator

- ▪Complete Function

- ▪Partial Timing

Functional-First (Trace-driven)

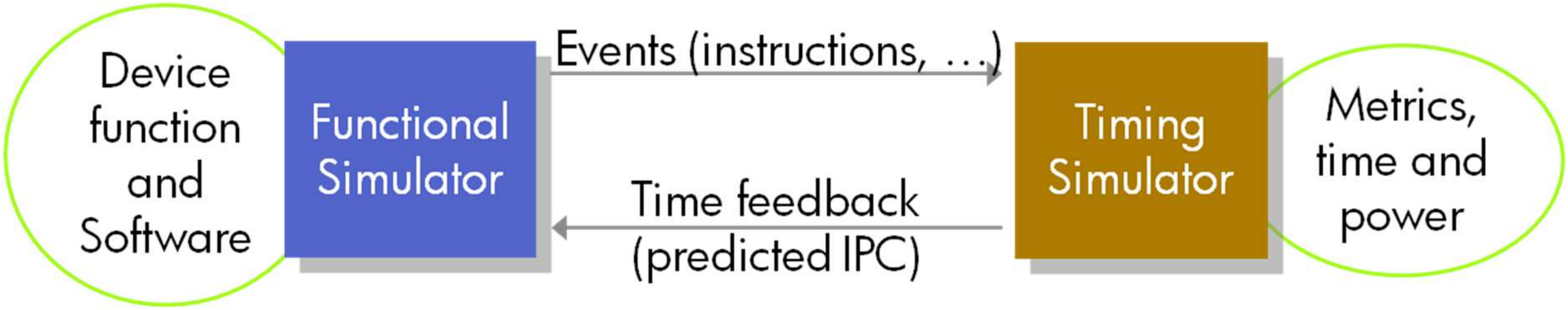
+Fast
- No timing feedback



Source: Multifacet Project (www.cs.wisc.edu/multifacet) - [Mauer02-sigmetrics-Full_System Timing_First Simulation]

COTSon: FUNCTIONAL-DIRECTED

- A variant of “functional first”
  - Adds timing feedback at coarse granularity (100s – 1000s of instructions)
- Applications see an approximation of time
  - May miss some fine-grain timing interaction
- Compatible with fast (caching) emulators and samplers

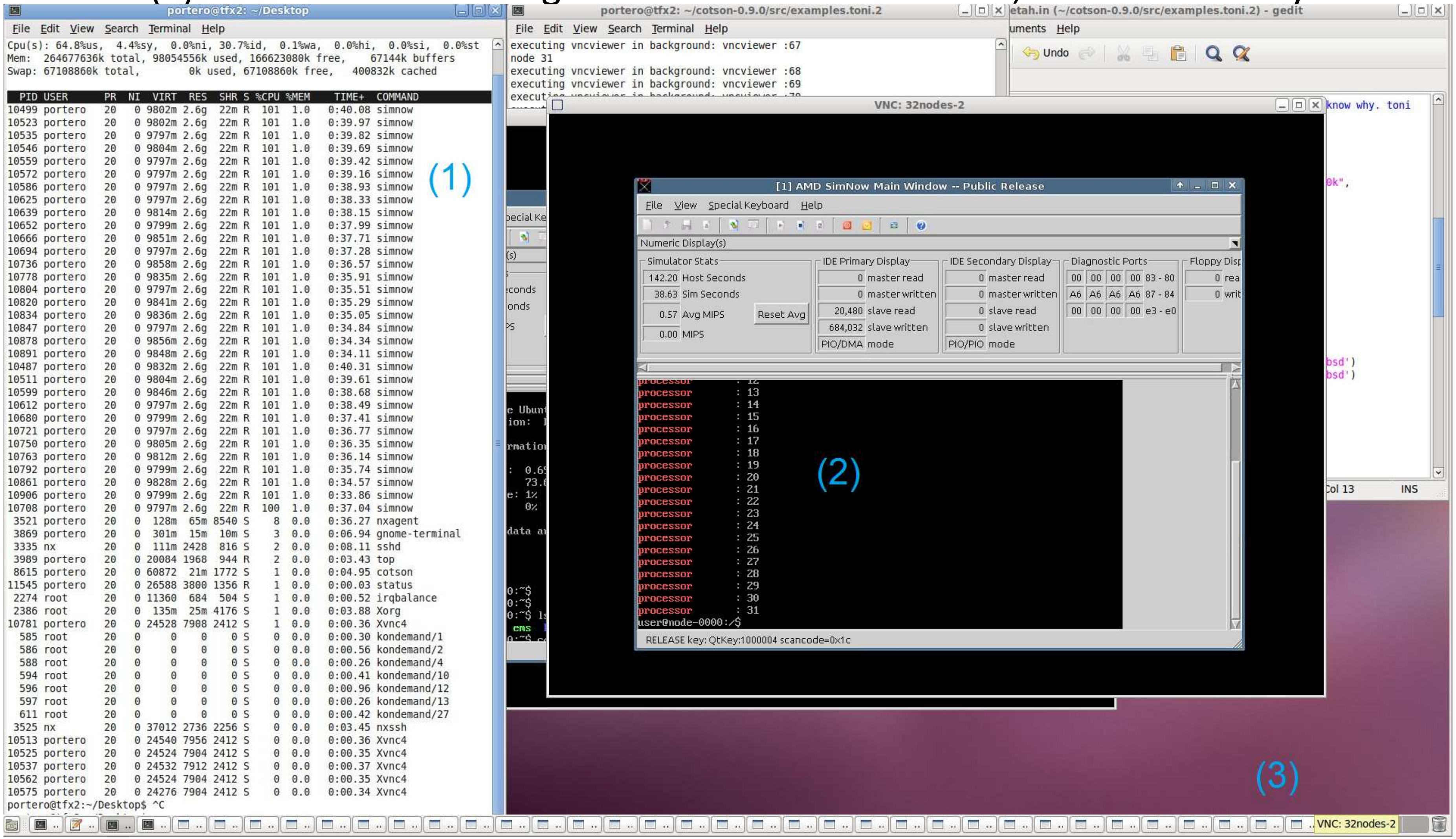


OTHER RECENT X64 SIMULATORS

	Sniper	Graphite	Gem5	COTSon	MARSSx86
Timing-directed/integrated			X		
Func-directed	X	X		X	X
User-level	X	X	X	X	X
Full-system			X	X	X
Archs Supported	x64	x64	x64 Alpha SPARC	x64	x64
Parallel (in-node)	X	X		Multi-node	
Shared caches	X		X	X	X

Heirman120401-ISPASS Tutorial
The SNIPER multi-core simulator

Simulation booting up 1024 cores. (1) COTSon execution of 32 SimNow instances. (2) Each instance manages 32 cores. Host: 48 cores, 256 GB memory



portero@tfx2: ~/Desktop
File Edit View Search Terminal Help
(1)
portero@tfx2: ~/cotson-0.9.0/src/examples.toni.2
File Edit View Search Terminal Help
etah.in (~/cotson-0.9.0/src/examples.toni.2) - gedit
uments Help
Undo
VNC: 32nodes-2
[1] AMD SimNow Main Window -- Public Release
File View Special Keyboard Help
Numeric Display(s)
(2)
(3)
VNC: 32nodes-2

```
Cpu(s): 64.8%us,  4.4%sy,  0.0%ni, 30.7%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  264677636k total, 98054556k used, 166623080k free,    67144k buffers
Swap: 67108860k total,        0k used, 67108860k free,   400832k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10499 portero   20   0 9802m 2.6g  22m R  101  1.0   0:40.08 simnow
10523 portero   20   0 9802m 2.6g  22m R  101  1.0   0:39.97 simnow
10535 portero   20   0 9797m 2.6g  22m R  101  1.0   0:39.82 simnow
10546 portero   20   0 9804m 2.6g  22m R  101  1.0   0:39.69 simnow
10559 portero   20   0 9797m 2.6g  22m R  101  1.0   0:39.42 simnow
10572 portero   20   0 9797m 2.6g  22m R  101  1.0   0:39.16 simnow
10586 portero   20   0 9797m 2.6g  22m R  101  1.0   0:38.93 simnow
10625 portero   20   0 9797m 2.6g  22m R  101  1.0   0:38.33 simnow
10639 portero   20   0 9814m 2.6g  22m R  101  1.0   0:38.15 simnow
10652 portero   20   0 9799m 2.6g  22m R  101  1.0   0:37.99 simnow
10666 portero   20   0 9851m 2.6g  22m R  101  1.0   0:37.71 simnow
10694 portero   20   0 9797m 2.6g  22m R  101  1.0   0:37.28 simnow
10736 portero   20   0 9858m 2.6g  22m R  101  1.0   0:36.57 simnow
10778 portero   20   0 9835m 2.6g  22m R  101  1.0   0:35.91 simnow
10804 portero   20   0 9797m 2.6g  22m R  101  1.0   0:35.51 simnow
10820 portero   20   0 9841m 2.6g  22m R  101  1.0   0:35.29 simnow
10834 portero   20   0 9836m 2.6g  22m R  101  1.0   0:35.05 simnow
10847 portero   20   0 9797m 2.6g  22m R  101  1.0   0:34.84 simnow
10878 portero   20   0 9856m 2.6g  22m R  101  1.0   0:34.34 simnow
10891 portero   20   0 9848m 2.6g  22m R  101  1.0   0:34.11 simnow
10487 portero   20   0 9832m 2.6g  22m R  101  1.0   0:40.31 simnow
10511 portero   20   0 9804m 2.6g  22m R  101  1.0   0:39.61 simnow
10599 portero   20   0 9846m 2.6g  22m R  101  1.0   0:38.68 simnow
10612 portero   20   0 9797m 2.6g  22m R  101  1.0   0:38.49 simnow
10680 portero   20   0 9799m 2.6g  22m R  101  1.0   0:37.41 simnow
10721 portero   20   0 9797m 2.6g  22m R  101  1.0   0:36.77 simnow
10750 portero   20   0 9805m 2.6g  22m R  101  1.0   0:36.35 simnow
10763 portero   20   0 9812m 2.6g  22m R  101  1.0   0:36.14 simnow
10792 portero   20   0 9799m 2.6g  22m R  101  1.0   0:35.74 simnow
10861 portero   20   0 9828m 2.6g  22m R  101  1.0   0:34.57 simnow
10906 portero   20   0 9799m 2.6g  22m R  101  1.0   0:33.86 simnow
10708 portero   20   0 9797m 2.6g  22m R  100  1.0   0:37.04 simnow
 3521 portero   20   0  128m  65m 8540 S    8  0.0   0:36.27 nxagent
 3869 portero   20   0  301m  15m  10m S    3  0.0   0:06.94 gnome-terminal
 3335 nx        20   0  111m 2428  816 S    2  0.0   0:08.11 sshd
 3989 portero   20   0 20084 1968  944 R    2  0.0   0:03.43 top
 8615 portero   20   0 60872  21m 1772 S    1  0.0   0:04.95 cotson
11545 portero   20   0 26588 3800 1356 R    1  0.0   0:00.03 status
 2274 root      20   0 11360  684  504 S    1  0.0   0:00.52 irqbalance
 2386 root      20   0  135m  25m 4176 S    1  0.0   0:03.88 Xorg
10781 portero   20   0 24528 7908 2412 S    1  0.0   0:00.36 Xvnc4
  585 root      20   0     0    0    0 S    0  0.0   0:00.30 kondemand/1
  586 root      20   0     0    0    0 S    0  0.0   0:00.56 kondemand/2
  588 root      20   0     0    0    0 S    0  0.0   0:00.26 kondemand/4
  594 root      20   0     0    0    0 S    0  0.0   0:00.41 kondemand/10
  596 root      20   0     0    0    0 S    0  0.0   0:00.96 kondemand/12
  597 root      20   0     0    0    0 S    0  0.0   0:00.26 kondemand/13
  611 root      20   0     0    0    0 S    0  0.0   0:00.42 kondemand/27
 3525 nx        20   0 37012 2736 2256 S    0  0.0   0:03.45 nxssh
10513 portero   20   0 24540 7956 2412 S    0  0.0   0:00.36 Xvnc4
10525 portero   20   0 24524 7904 2412 S    0  0.0   0:00.35 Xvnc4
10537 portero   20   0 24532 7912 2412 S    0  0.0   0:00.37 Xvnc4
10562 portero   20   0 24524 7904 2412 S    0  0.0   0:00.35 Xvnc4
10575 portero   20   0 24276 7904 2412 S    0  0.0   0:00.34 Xvnc4
portero@tfx2:~/Desktop$ ^C
```

```
executing vncviewer in background: vncviewer :67
node 31
executing vncviewer in background: vncviewer :68
executing vncviewer in background: vncviewer :69
```

Simulator Stats		IDE Primary Display		IDE Secondary Display		Diagnostic Ports					Floppy Disp
142.20	Host Seconds	0	master read	0	master read	00	00	00	00	83 - 80	0 rea
38.63	Sim Seconds	0	master written	0	master written	A6	A6	A6	A6	87 - 84	0 writ
0.57	Avg MIPS	20,480	slave read	0	slave read	00	00	00	00	e3 - e0	
0.00	MIPS	684,032	slave written	0	slave written						
		PIO/DMA	mode	PIO/PIO	mode						

Reset Avg

```
processor       : 13
processor       : 14
processor       : 15
processor       : 16
processor       : 17
processor       : 18
processor       : 19
processor       : 20
processor       : 21
processor       : 22
processor       : 23
processor       : 24
processor       : 25
processor       : 26
processor       : 27
processor       : 28
processor       : 29
processor       : 30
processor       : 31
user@node-0000:/$
```

RELEASE key: QtKey:1000004 scancode=0x1c

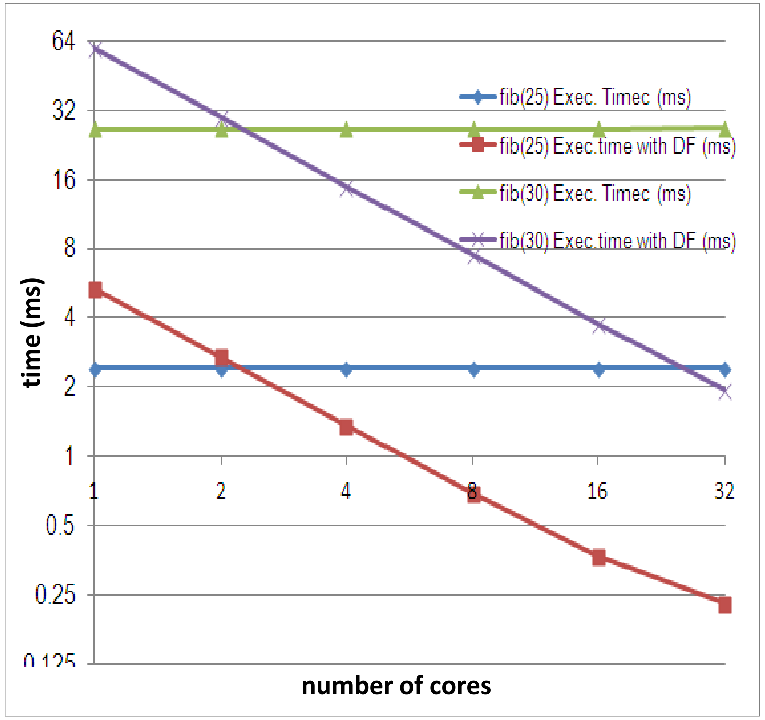
```
know why. toni
0k",
bsd')
bsd')
```

Col 13 INS

Note: the simulation is PARALLEL at GUEST NODE-LEVEL and it's also possible to distribute the simulation on several HOST NODES

INITIAL EXPERIMENTAL RESULTS

The proposed simulation framework has been validated running applications and benchmarks on a target machine with up to 1024 cores, operating in accordance with dataflow principle on standard cores



We run several applications and benchmarks based on well established programming models (mainly OpenMP and MPI):

- NAS Parallel Benchmark (NPB)
- Graph500 and HPL 2.0 Linpack
- Sequential Recursive Fibonacci

Major Technical Innovations in TERAFLUX

- Fragmenting the Applications in Finer grained DF-threads:
  - DF-threads allow an easy way to decouple memory accesses, therefore hiding memory latencies, balancing the load, managing fault, temperature information without fine grain intervention of the software.
- Possibility to repeat the execution of a DF-thread in case this thread happened to be on a core later discovered as faulty
- Taking advantage of a “direct” dataflow communication of the data (through what we call DF-frames).
- Synchronizing threads while taking advantage of native dataflow mechanism (e.g. several threads can be synchronized at a barrier)
  - DF-threads allow (atomic) Transactional semantics (DF meets TM)
- A Thread Scheduling Unit would allow fast thread switching and scheduling, besides the OS scheduler; scalable and distributed
- A Fault Detection Unit works in conjunction with TSU

TERAFLUX SIMULATOR (COTSon)

http://cotson.sf.net

HP-Labs COTSon is **OPEN-SOURCE**

FUTURE AND
EMERGING
TECHNOLOGIES
PROJECT N. 249013

COOPERATION

SEVENTH FRAMEWORK
PROGRAMME THEME
FET proactive 1 (ICT-2009.8.1)
Concurrent Tera-Device Computing

TERA^{F}LUX

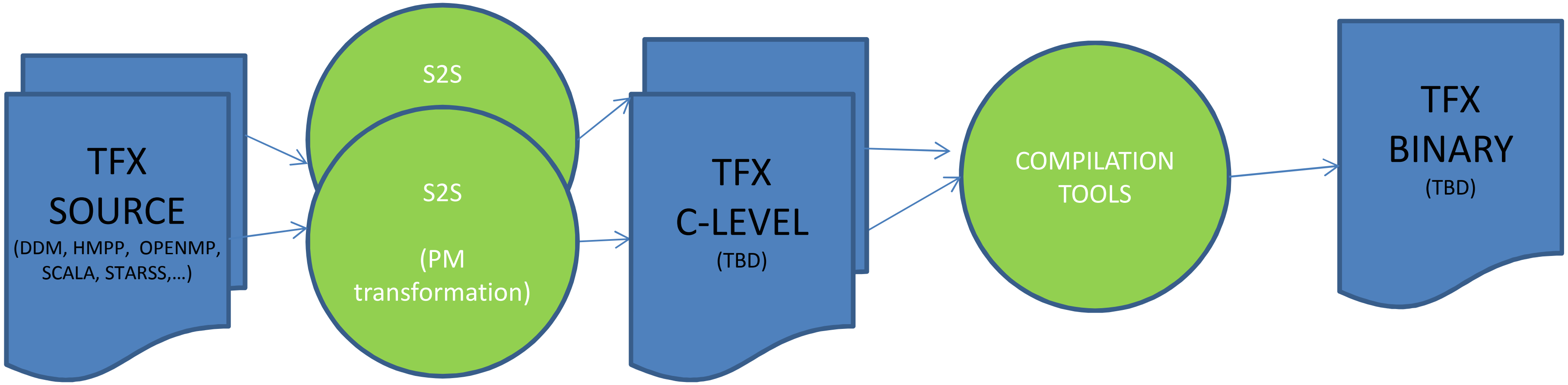
Exploiting dataflow parallelism in Teradevice Computing

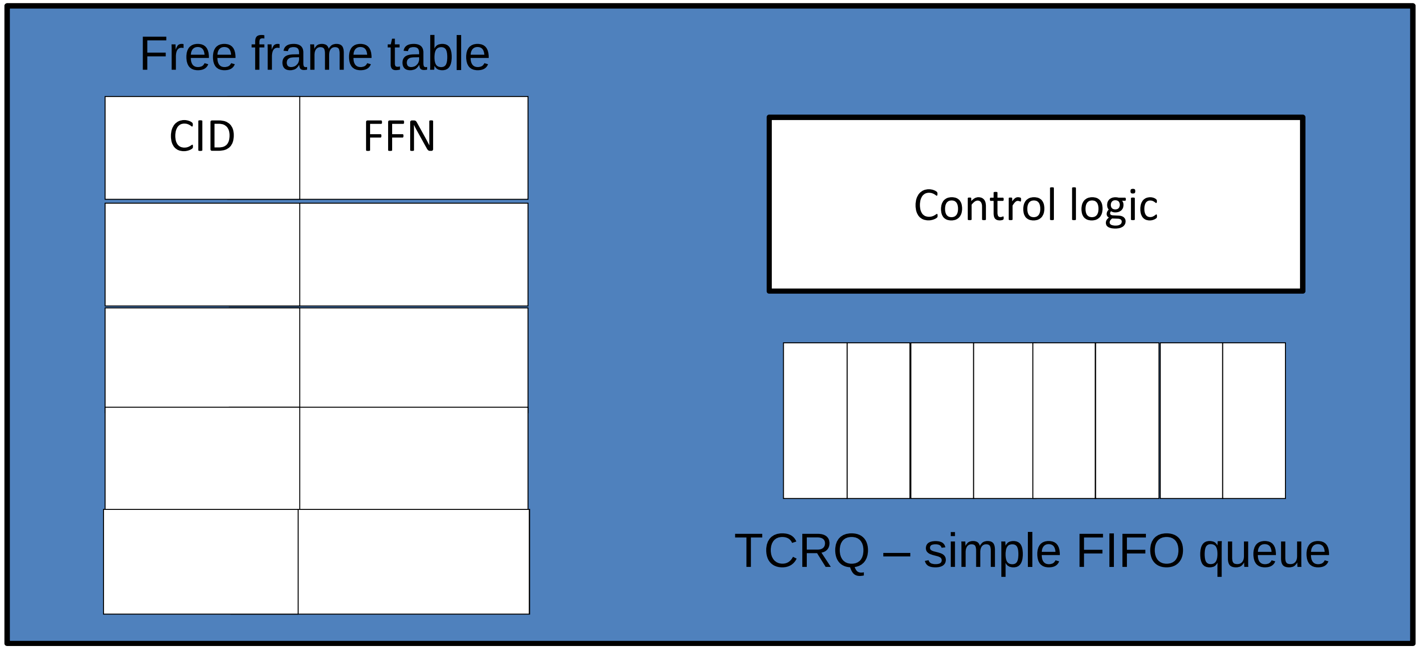
PROJECT NUMBER: 249013

http://teraflux.eu

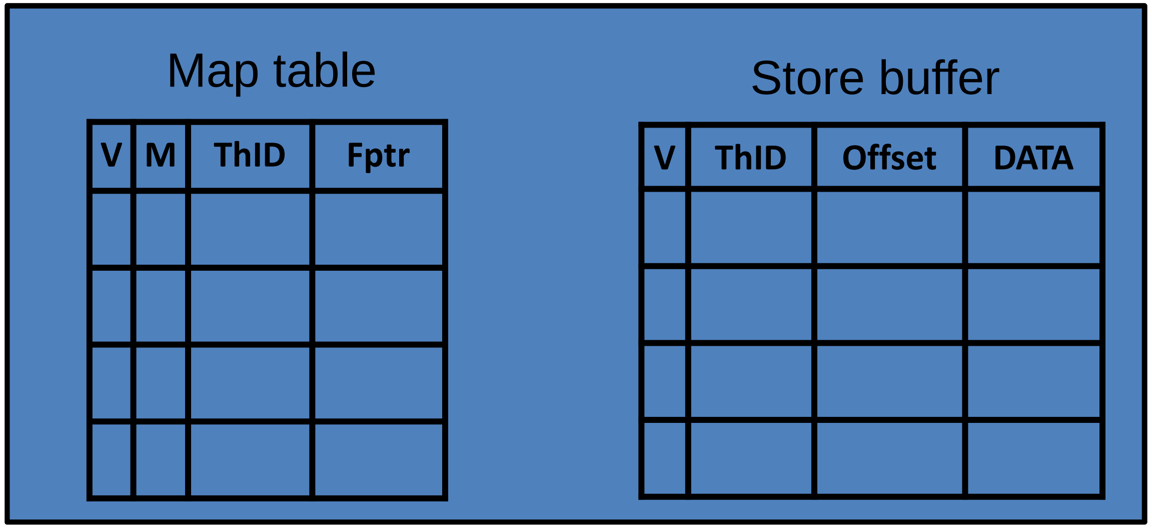
BACKUP SLIDES

TERAFLUX TOOLCHAIN (Jan. 2011)





- FreeFrameTable (CID – Core ID, FFN – number of free frames)
  - Keeps track of the occupancy of processors inside a cluster
  - Updated on each TDestroy and accepted TSchedule
- TCRQ – ThreadScheduleRequestQueue
  - Holds unserved ThreadScheduleRequest messages
  - Message is pushed into queue when there are no free local resources
  - Message is popped from the queue when either TDestroyor BroadcastResponse arrives
- Control logic
  - Responsible for both inter and intra node communication and updating the messages inside a scheduler



Map table

V	M	ThID	Fptr

Store buffer

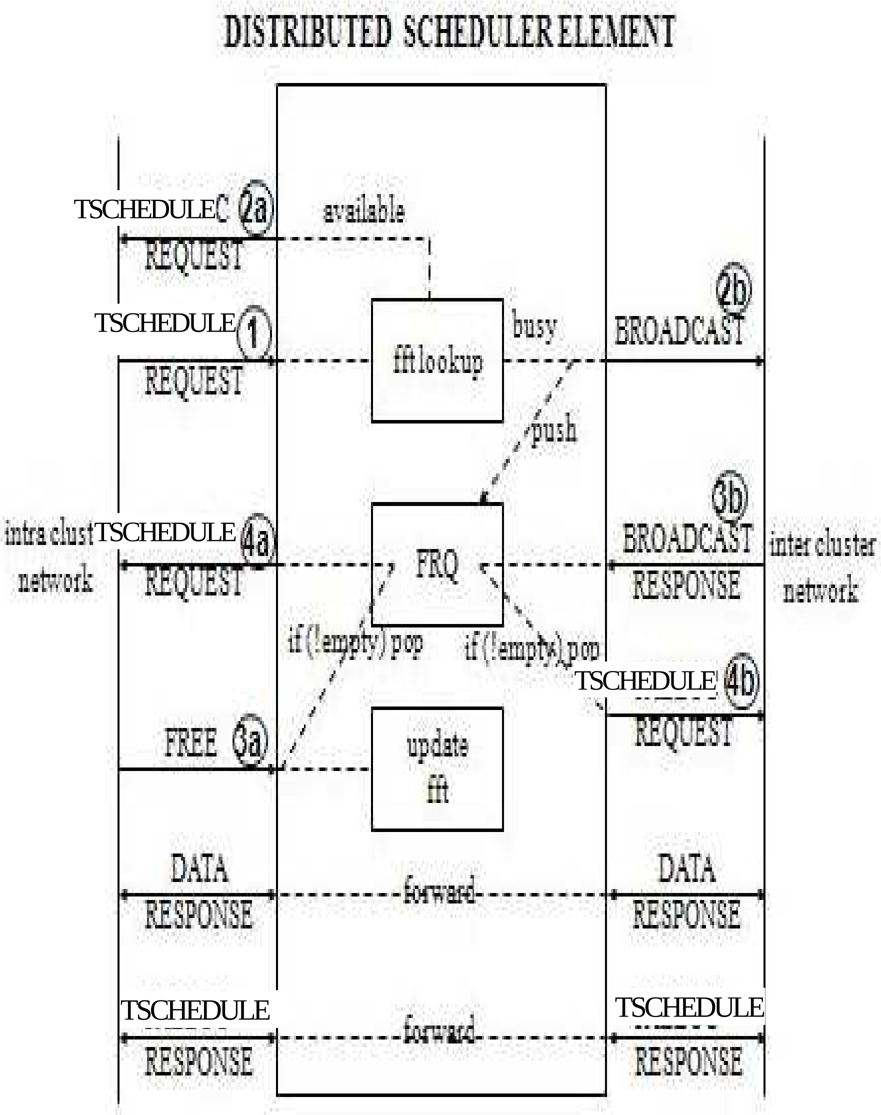
V	ThID	Offset	DATA

- Map table (V – Valid, M – Mapped, ThID – thread ID, Fptr – Frame pointer)
  - Keeps track of the issued resource requests for the execution of new threads
  - a ThID is assigned to a thread when new Tschedule instruction occurs; it is used just inside that core
  - a Fptr is assigned when a TScheduleResponse message arrives; it is unique globally in the system
  - Can be cleared on the thread completion

- Store buffer (V – valid, ThID – thread ID, offset – for storing in a frame, DATA – data to store)
  - Keeps track of the TStores issued for the threads that didn’t receive a TScheduleResponse yet (those kept in Map table and still not mapped)
  - On each TStore for the new thread that still doesn’t have resources assigned, a new entry is created
  - When TScheduleResponse arrives, all entries are checked and TStore messages are sent (entry invalidated) if there is any matching
  - If TStore occurs for a thread that already has its resources assigned, there is no need to use the buffer

Distributed Thread Scheduler Unit

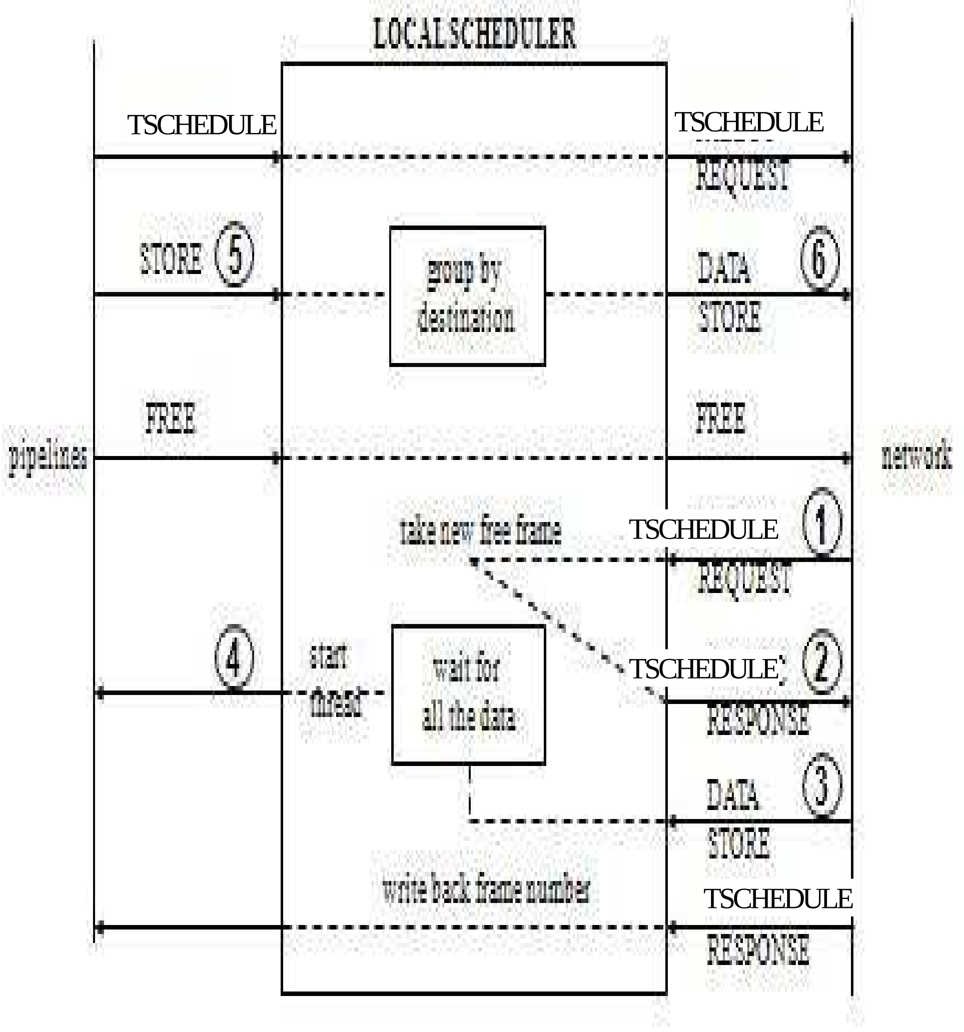
- On new TScheduleRequestMessage checks the availability in local node
  - If yes – forwards it
  - If no – put the message in FRQ and send broadcast
- Message is removed from FRQ when FfreeMessage or BroadcastResponse arrive
- Other messages are just forwarded



TERA^F^LUX

Local Thread Scheduling Unit

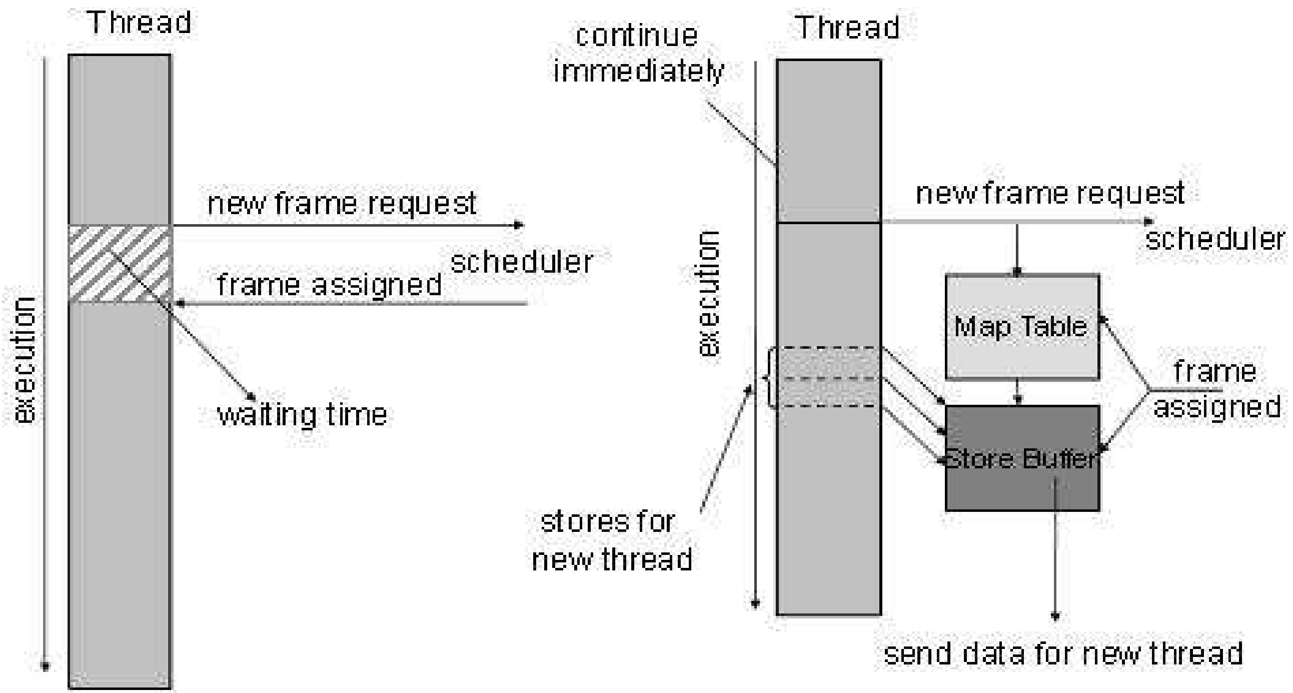
- On TScheduleRequestMessage
  - Choose a free frame for execution of the new thread
  - Send TScheduleResponseMessage to the issuing processor
- On TScheduleResponseMessage simply update the continuation with the frame identifier
- On store send DataStore message (group them if the destination is the same)



TERAFLUX

Non-blocking resource assignment (1)

- Avoid waiting from the distributed scheduler by introducing Virtual frame pointers
- Two additional structures – map table and store buffer

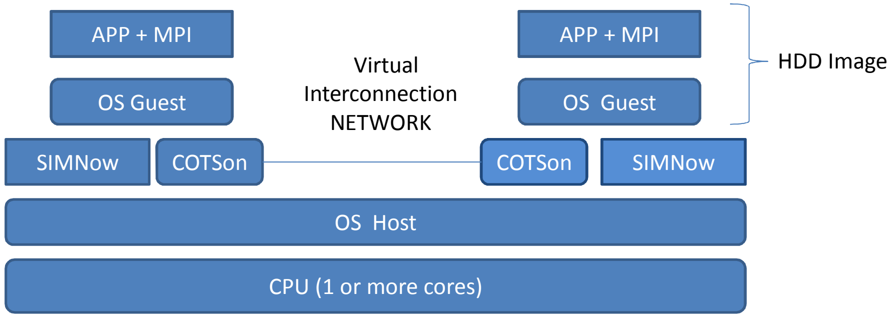


Blocking frame assignment

Non-blocking frame assignment

Even if we don’t speed-up the starting time of new threads, execution time is shorter and processor becomes free earlier

One Physical Machine running two Virtual Machine instances that communicate through the Virtual Network (Mediator).



ADVANTAGES

This setup can run both on a real machines (at least at smal scale for tests) AND on the COTSon simulator

It allows us to modify system parameters like e.g. number o. cores in each simulated instance.

It allows for a parallelization of the simulation (the several instances are running in parallel on the available cores – load balancing automatically provided by the Host OS scheduler).

Possible to avoid copying buffers among instances because they reside in the Host Shared Memory Network

Possibility to take advantage of RVI/VT-x virtualization mechanisms across different Physical Machines (under development).

The communication and synchronization among the simulation instances adds up to the Application traffic, but could bypass TCP/IP and avoid using the Physical Interconnection Network.

No need to use the Physical Network.

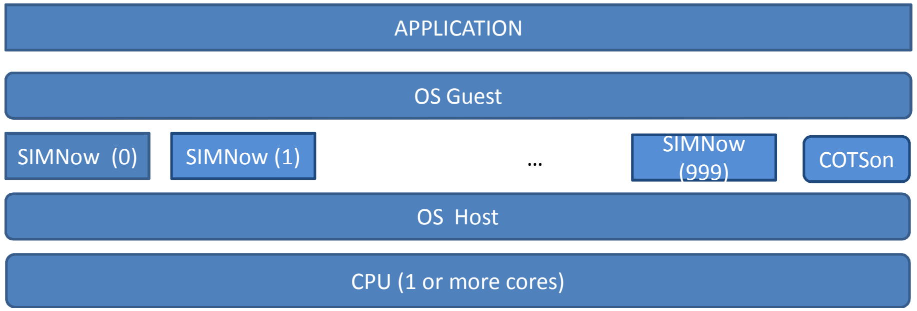
DISADVANTAGES

- Taking into account that we aim to flexibly change the programming model and architecture (e.g. the dataflow based execution model and architecture), this setup may end up in poor performance when N (number of nodes) increases.
- Tightens the Application to the Machine, which is exactly the opposite direction that we follow globally in TERAFLUX: we aim to decouple the Application (WP2) from the Machine with appropriate Programming Models (WP3), Compilation Tools (WP4) and Execution Models (WP6).
- The MPI run-time is constantly involved to appropriately schedule the ready tasks/threads on the available nodes.
- The Physical architecture that is more natural to model is a Distributed Machine not like the general one we aim in TERAFLUX.

VM instances governed by a Single Source Image (SSI) OS

ADVANTAGES

- Allows us to run Shared Memory applications like OpenMP ones (can still run MPI as if it was a single big node).
- Can run both on a real machines (at least at small scale for tests) and on the COTSon simulator.
- It allows us to modify system parameters like e.g. number of cores in each simulated instance.
- It allows for a parallelization of the simulation (the several instances are running in parallel on the available cores – load balancing automatically provided by the Host OS scheduler).
- Possible to avoid copying buffers among instances because they reside in the Host Shared Memory Network
- Possibility to take advantage of RVI/VT-x virtualization mechanisms across different Physical Machines (under development).
- The communication and synchronization among the simulation instances adds up to the Application traffic, but could bypass TCP/IP and avoid using the Physical Interconnection Network.
- Load Balancing for the Application is managed by the Guest OS
- No need to use the Physical Network.



DISADVANTAGES

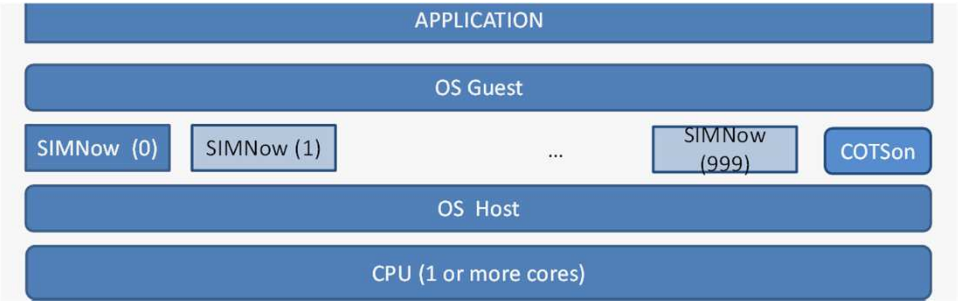
This setup requires the use of a Distributed OS as Guest OS (like e.g., Kerrighed [KERRIGHED10], which offers the view of a unique SMP machine on top of a cluster) or in general a SSI (Single System Image) OS.

Relatively poor performance when N (number of nodes) increases;

Partially tightens the Application to the Machine, which is in the opposite direction in respect to what we follow globally in TERAFLUX: we aim to decouple the Application (WP2) from the Machine with appropriate Programming Models (WP3), Compilation Tools (WP4) and Execution Models (WP6).

The underlying Guest Architecture is a “cluster”, which is then more naturally mapped to a physical Distributed Machine not a generic one like we aim in TERAFLUX.

One core aware of all the other cores



ADVANTAGES

- Allows us to run Shared Memory applications like OpenMP ones (can still run MPI as if it was a single big node).
- Can run both on a real machines (at least at small scale for tests) AND on the COTSon simulator as provided at the Month-1 of the TERAFLUX project.
- It allows us to modify system parameters like e.g. number of cores in each simulated instance.
- It allows for a parallelization of the simulation (the several instances are running in parallel on the available cores – load balancing automatically provided by the Host OS scheduler).
- Possible to avoid copying buffers among instances because they reside in the Host Shared Memory Network.
- Possibility to take advantage of RVI/VT-x virtualization mechanisms across different Physical Machines (under development).
- The communication and synchronization among the simulation instances adds up to the Application traffic, but could bypass TCP/IP and avoid using the Physical Interconnection Network.
- Load Balancing for the Application is managed by the Guest OS
- No need to use the Physical Network.
- No need to use a very different OS like an SSI OS.
- The underlying Guest Architecture is a shared memory machine, however thanks to the availability of a global address space, there is now full possibility of evolving the machine in a more “general one” like the one we aim to evolve during the TERAFLUX project. The TERAFLUX Execution Model can decouple completely the architecture of the machine.

DISADVANTAGES

- Relatively poor performance when N (number of nodes) increases; however, as other simulator like COREMU [Wang11] already demonstrated a high speed up in simulations even with 255 cores, we have good confidence that we can improve much the simulation speed going in a similar direction.
- Requires some patches to the Linux OS; however we shall need to patch anyway the Memory Manager and the Scheduler in order to properly support the TERAFLUX threads

NAS benchmarks running in COTSon

- Machine 37nodes of 4 cores. One node Master and 36 Slaves
- Two examples from the set:

```
NAS Parallel Benchmarks 3.3 -- BT Benchmark
 No input file inputbt.data. Using compiled default
Class A: Size:   64x  64x  64
Iterations:  200    dt:   0.0008000
Number of active processes:    36
```

	Time in seconds	Mop/s total	Mop/s/process
BT 1-4	398.93	421.84	11.72
BT 2-4	422.08	398.7	11.08
BT 4-4	398.17	422.65	11.74

```
NAS Parallel Benchmarks 3.3 – CG
     Size:      14000
     Iterations:    15
     Number of active processes:    32
     Number of nonzeroes per row:       11
     Eigenvalue shift: .200E+02
```

	Time in seconds	Mop/s total	Mop/s/process
CG 1-4	46.26	32.35	1.01
CG 2-4	48.45	30.89	0.97
CG 4-4	46.65	32.08	1

NAS benchmarks running in COTSon (cont.)

```
NAS Parallel Benchmarks 3.3 -- EP Benchmark
 Number of random numbers generated:       536870912
 Number of active processes:                      32
EP Benchmark Results: CPU Time =    5.5777, N = 2^   28
________________________________________
 NAS Parallel Benchmarks 3.3 -- FT Benchmark
 No input file inputft.data. Using compiled defaults
 Size               :  256x 256x 128 (Class A)
 Iterations         :              6,  Number of processes :       32
 Processor array     :         1x  32, Layout type         :     1D
_____________________________________
 NAS Parallel Benchmarks 3.3 -- IS Benchmark

 Size:  8388608  (class A), Iterations:   10
 Number of processes:     32, IS Benchmark Completed
 Class           =      A, Size            =     8388608
 Iterations      =           10
_________________________________________
 NAS Parallel Benchmarks 3.3 -- LU Benchmark
 Size:   64x  64x  64 (Class A),Iterations:  250
 Number of processes:    32
_________________________________
 NAS Parallel Benchmarks 3.3 -- MG Benchmark
No input file. Using compiled defaults
Size:  256x 256x 256  (class A)
 Iterations:    4,  Number of processes:     32
```

	Time in seconds	Mop/s total	Mop/s/ process
		410.53	
EP 1-4	4.9301	108.9	3.4
EP 2-4	5.5777	96.25	3.01
EP 4-4	4.9324	108.85	3.4
FT 1 -4	187.25	38.11	1.19
FT 2 -4	185.43	38.49	1.2
FT 4 -4	201.85	35.36	1.1
IS 1-4	2.59	32.39	1.01
IS 2 -4	2.67	31.45	0.98
IS 4 -4	2.57	32.64	1.02
LU 1-4	188.96	631.33	19.73
LU 2 -4	185.15	644.32	20.14
LU 4 -4	183.93	648.6	20.27
MG 1-4	171.15	22.74	0.71
MG 2 -4	176.11	22.1	0.69

Plurality

- Plurality: http://www.plurality.com/profile.html
  - Architecture: general-purpose accelerator for multicore/manycore system-on-chip (SoC)
  - Task-oriented programming model: the programmer has to perform a partitioning of the program into specific tasks (task-map)
  - The body of each task is a traditional sequential code
  - Each core is a RISC processor
  - Scheduler, synchronization and load balancing among cores are done by a complex hardware subsystem that communicates with all the RISC processors
  - Uniform shared memory access

CUDA

- Programming model: extensions to standard C language (CUDA libraries)
- DRAM memory addressing + on-chip shared memory
- However a single process must run spread across multiple disjoint memory spaces (???)
- Recursive functions are not supported (must be converted to loops)
- Bus bandwidth and latency between CPU and GPU may be a bottleneck (acceleratore esterno)

TFlux

- TFlux:
  - Paralell processing system targeted to commodity, Linux-based shared-memory multiprocessor systems
  - Data-Driven multi-threading
  - Programming model: takes as input a C program, argumented with TFlux-specific compiler directives (#pragma's)
  - Everything implemented in software

Subset Static Interleaved (SSI)

- Interleaved threads
  - Advantage: operations latencies become shorter in terms of executed instructions from the same thread
- Combination of blocked multithreading and static interleaved multithreading:
  - A set of background threads + a set of foreground threads. Foreground threads are interleaved until a stall occurs (e.g. cache miss). When a foreground thread stalls and a background thread is ready for execution we exchange them so that the foreground thread becomes a background thread and vice versa
- TSU: task scheduling unit (in hardware)