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

Reliability and fault-tolerance in future large-scale CMP architecture -- the Teraflux approach

Avi Mendelson – Microsoft + Technion
Theo Ungerer -- University of Augsburg

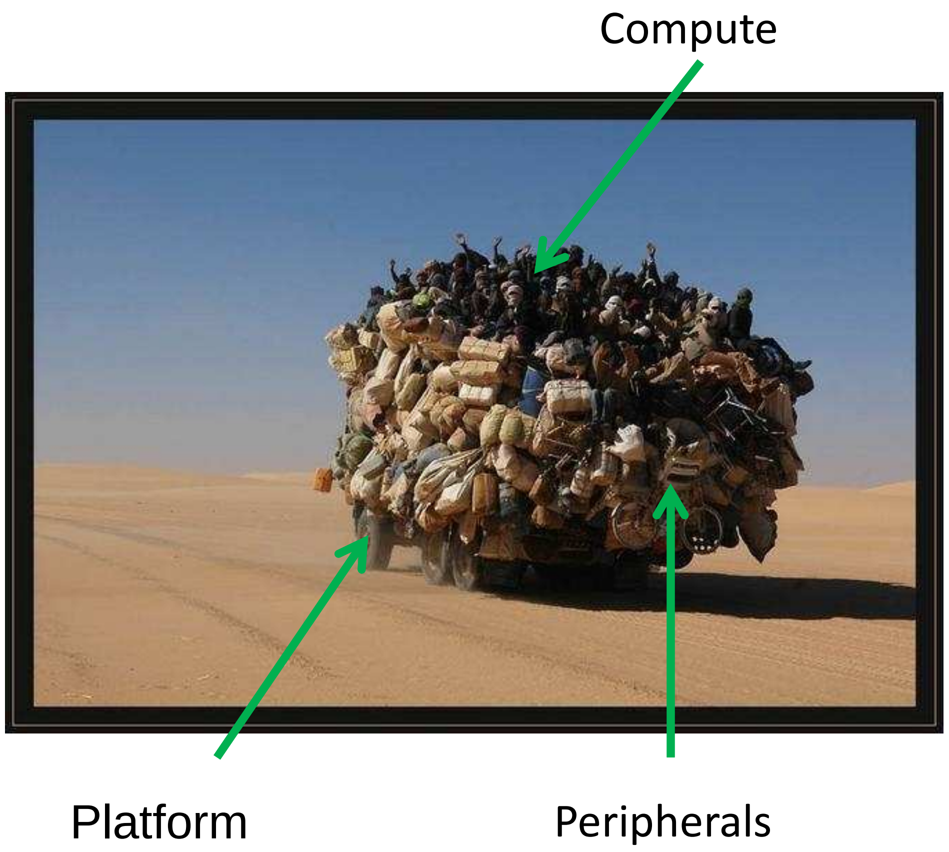
Teraflux in a nutshell

- An EU research project (FET).
- Assums 1000's processors on die
- Connected through a NoC
- No system-wide support for HW coherency
- HW components can become faulty
  - Transient errors
  - Stuck at faults
- SW needs to make sure it works transparency to potential faults
- Resource allocation and scheduling should be distribution

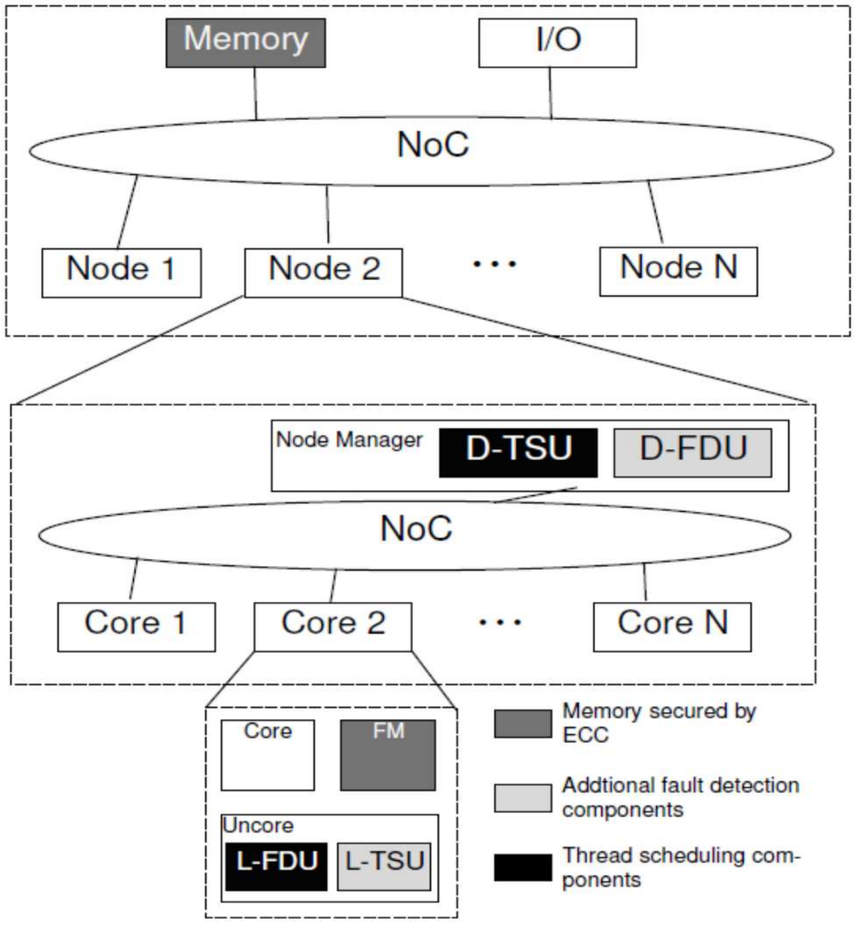
Disclaimer: The project examine different potential solutions, this presentation presents my approach

How to fit 1000 cores on die?

The unstructured option



The Structured (hierarchical) option

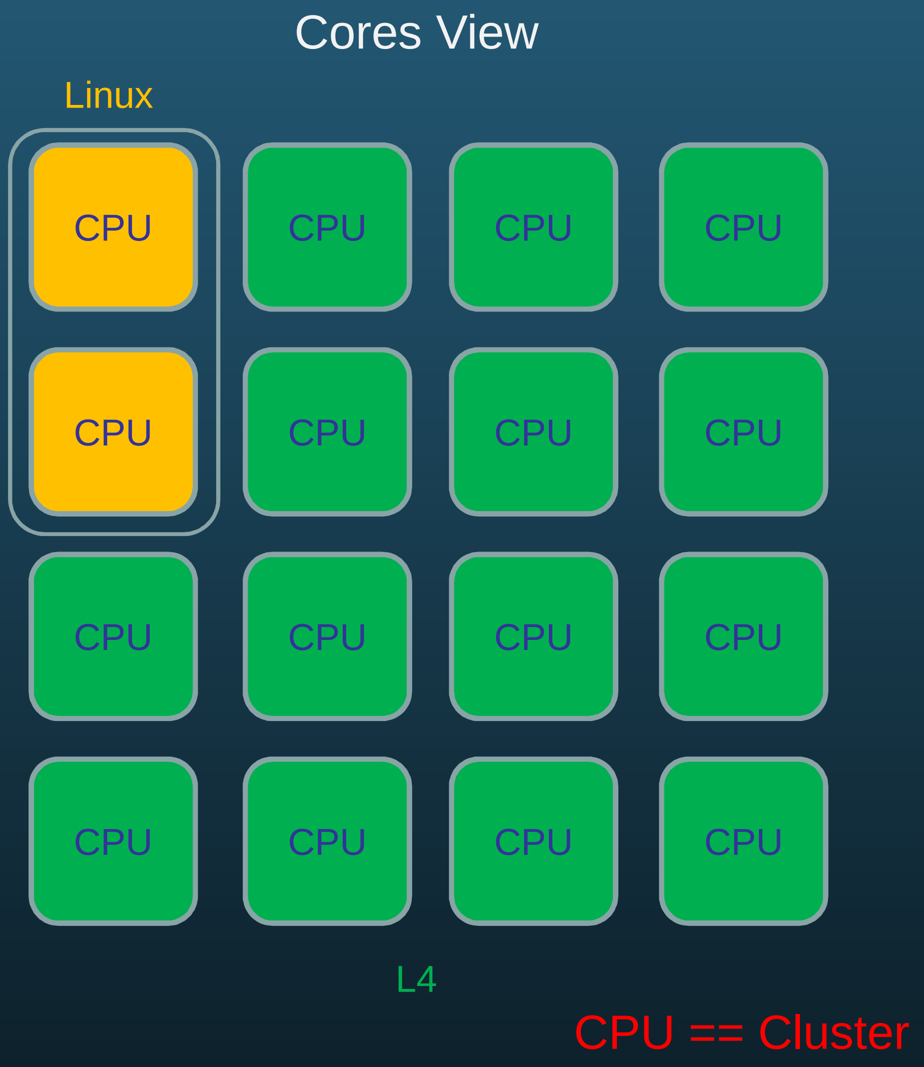


Basic Architecture

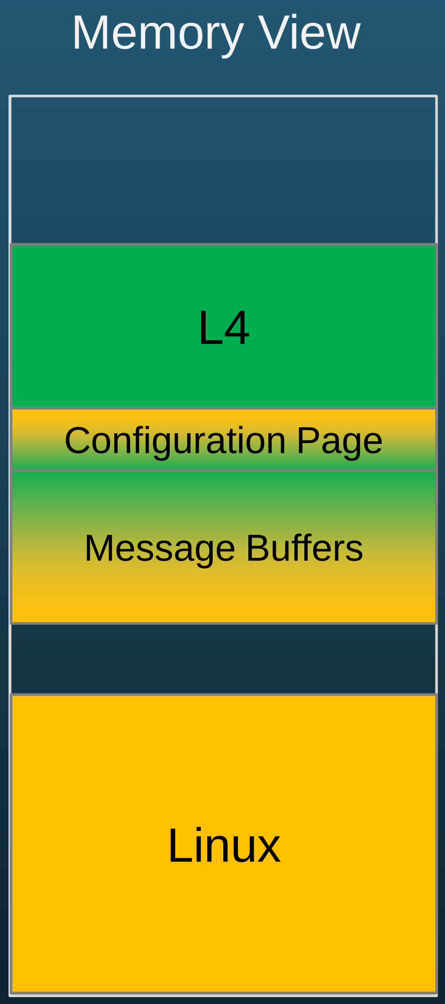
- Clustered architecture
  - Same ISA to all processors
  - HW based coherency within the cluster and no HW based coherency between clusters.
- Clusters can be symmetric or asymmetric;
  1. **Service-cluster(s)**: GP core that runs GP OS such as Linux.
  2. **Auxiliary clusters**: e.g., single issue, power efficient computational cores
- NoC: Supports
  1. Topological connections of resources (cores, memories, accelerators, etc.) within a node (cluster) and among nodes (clusters)
  2. The inner cluster NoC may be different than the external NoC
- Memory hierarchy
  1. Globally addressable physical space to guarantee on-chip global accessibility, possibly with variable latencies (NUMA)
  2. Physical memory may be partitioned into local memory vs. global memory

TERAFLUX

System Overview Target System



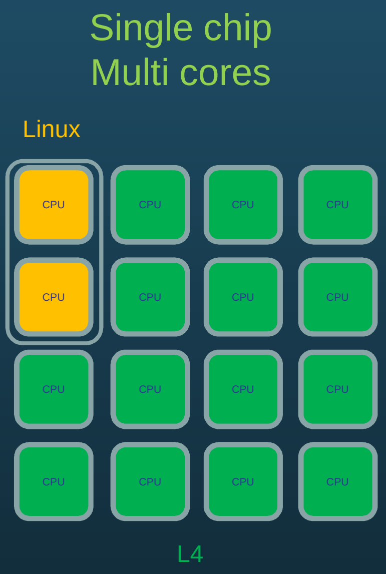
CPU == Cluster



TERAFLUX

Target System OS Requirements

- Manages jobs on uKernel (uK) cores
- Proxies uKs I/O requests
- Remote debug uKs/self
- Runs high level (system) FT managing uK/self faults

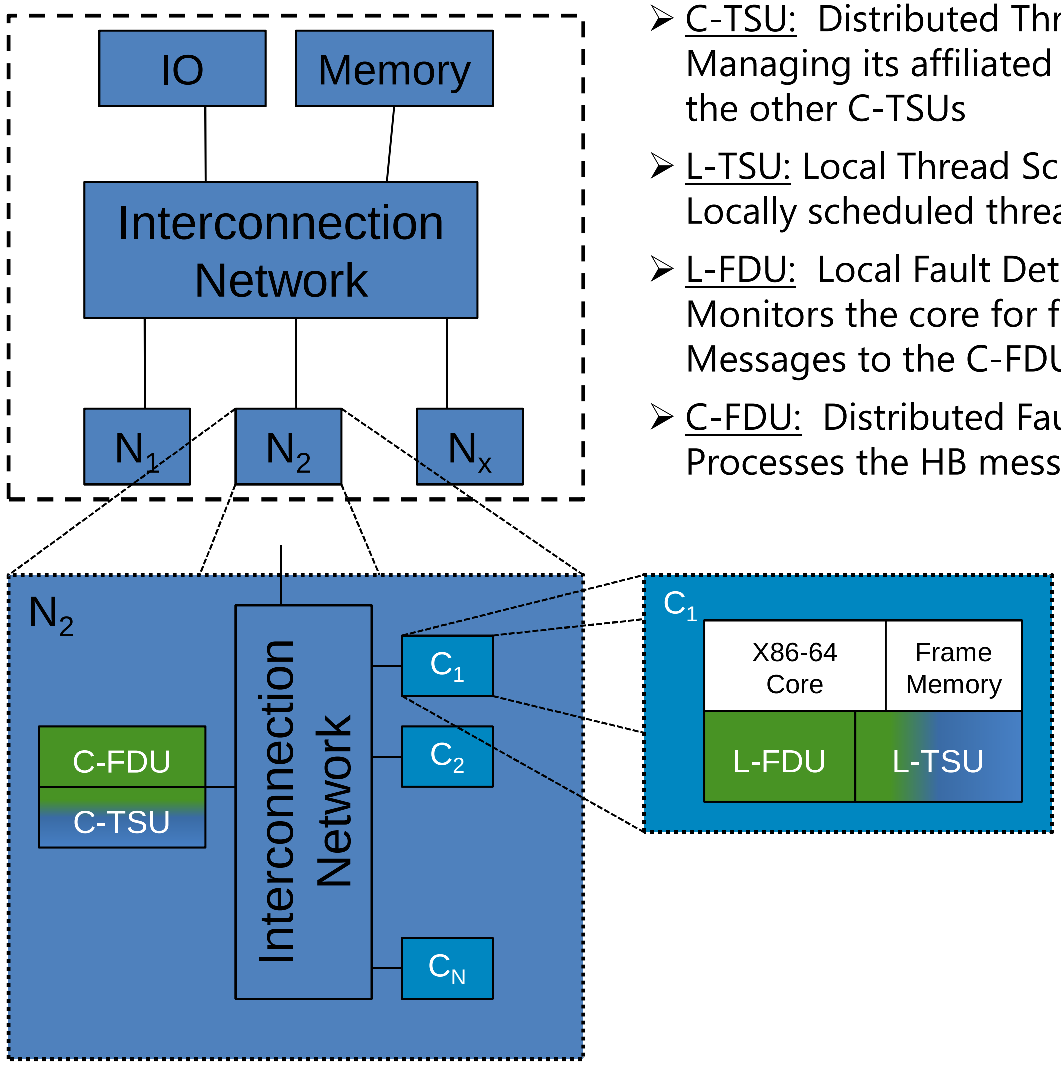


- Each uK runs a Task (or Tasks)
- Tasks sent by full OS (FOS)
- Tasks are DF entities, no side-effects
- Failed task simply restarted
- Runs low level FT, reporting to FOS

Reliability and Fault-Tolerant – high level

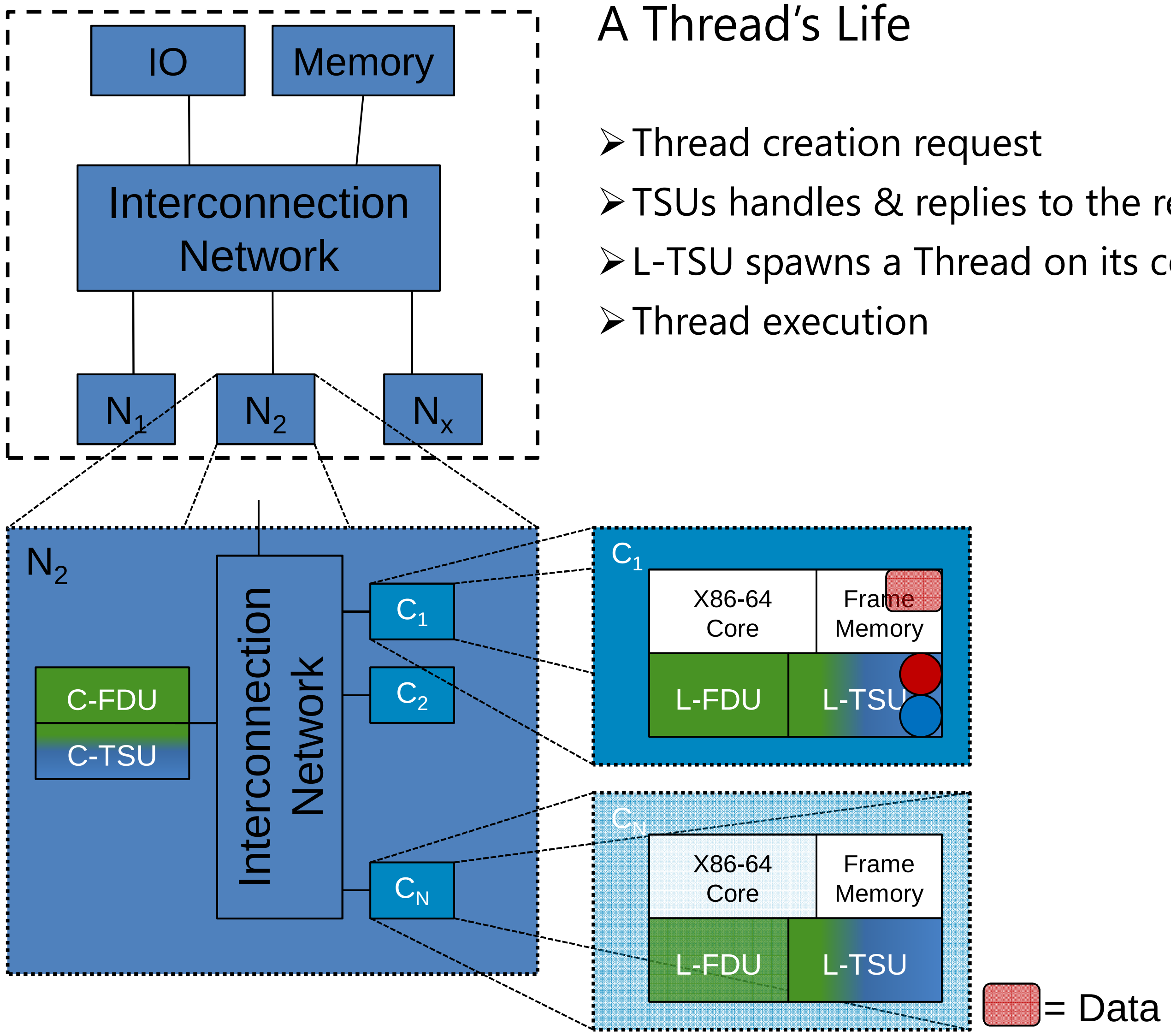
- Is implemented at all different levels of the hierarchy
  - At the Global Level – the Linux OS will management the resource partition, global scheduling, load balance, migration, etc.
  - At the NoC level, an adaptive algorithm is developed to manage failures of links
  - At cluster level we manage and report statistics on failure to upper level in order to balance the execution
  - At core level we assume a detection mechanism to report when the core, or execution of the core is faulty
- We will take advantage of the DF model that supports “all or nothing” execution mode.

Management and Reliability Components



- C-TSU: Distributed Thread Scheduling Unit
  Managing its affiliated L-TSUs and the communications with the other C-TSUs
- L-TSU: Local Thread Scheduling Unit
  Locally scheduled threads and handles communication
- L-FDU: Local Fault Detection Unit
  Monitors the core for fault detection and sends Heartbeat (HB) Messages to the C-FDU
- C-FDU: Distributed Fault Detection Unit
  Processes the HB messages and compares execution signatures

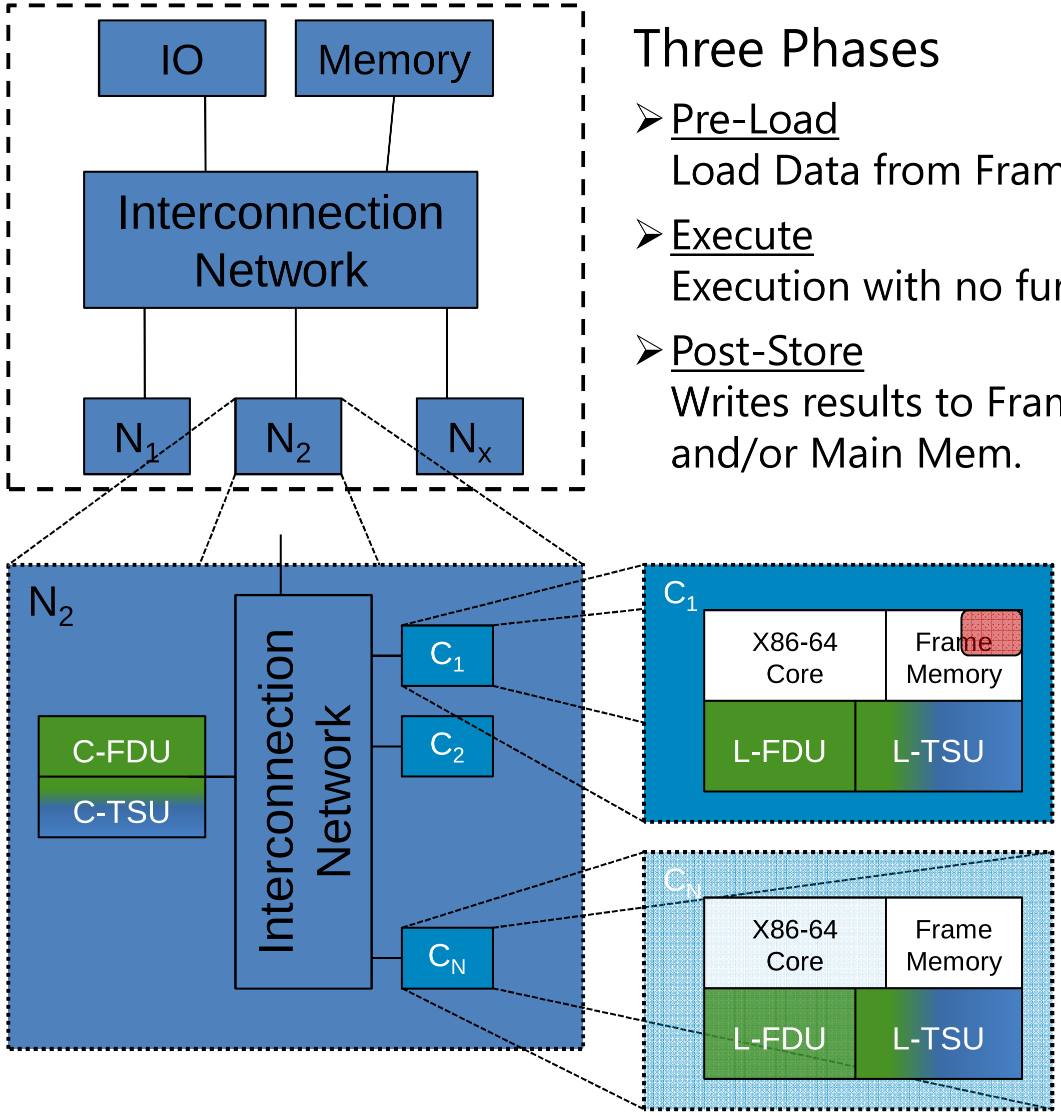
A C-DTA Example Thread



A Thread's Life

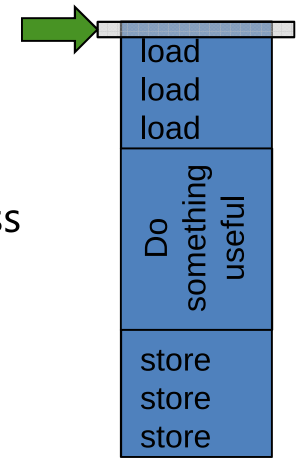
- ➢ Thread creation request
- ➢ TSUs handles & replies to the request
- ➢ L-TSU spawns a Thread on its core
- ➢ Thread execution

A Thread Execution

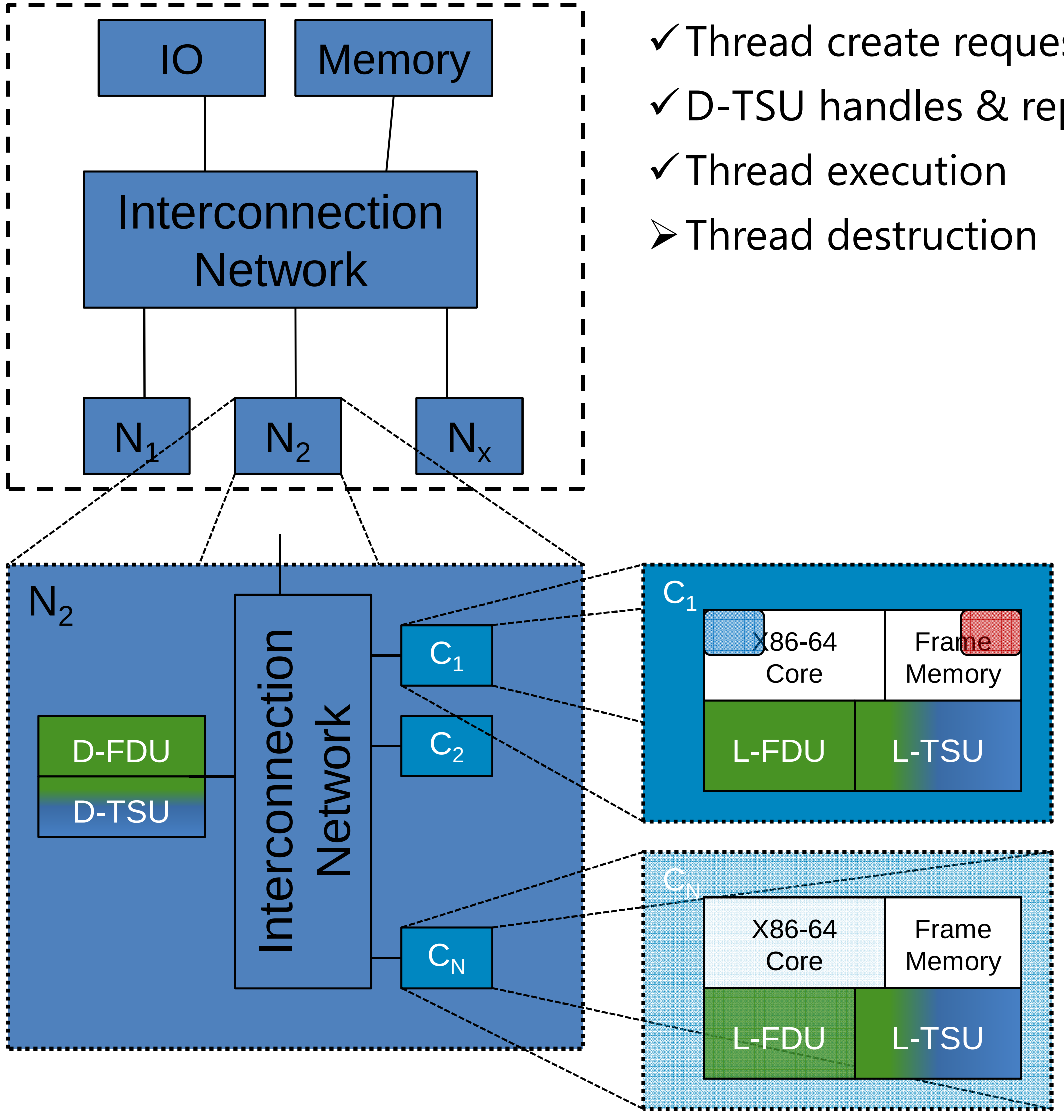


Three Phases

- Pre-Load
  Load Data from Frame Mem.
- Execute
  Execution with no further Mem. Access
- Post-Store
  Writes results to Frame Mem. and/or Main Mem.



A Thread's Life



- ✓ Thread create request
- ✓ D-TSU handles & replies the request
- ✓ Thread execution
- ➢ Thread destruction

How it works

- Compiler generate DF code out of sequential code (e.g., C)
- The execution always starts on the service cores that generate the threads (Tokens) and send them to the different clusters.
- All task sent to a cluster are kept in a “safe memory” queue and being scheduled to cores by the TSU
- After finishing the execution and assuming no fault happen, results are written to the task-memory and the TSU is reported it can write the results back to main memory. After successful update of the global memory, the thread is removed from the clustered queue.
- If an error occurs between during execution, the task is killed (no side effect) since Data Flow) and reschedule on another processor within the cluster.
- If error occurs while reading or writing data from or to the main memory, we assume a retransmit mechanism to guarantee the completion (at that point we assume that the operation must complete. We may release this assumption in the future.)

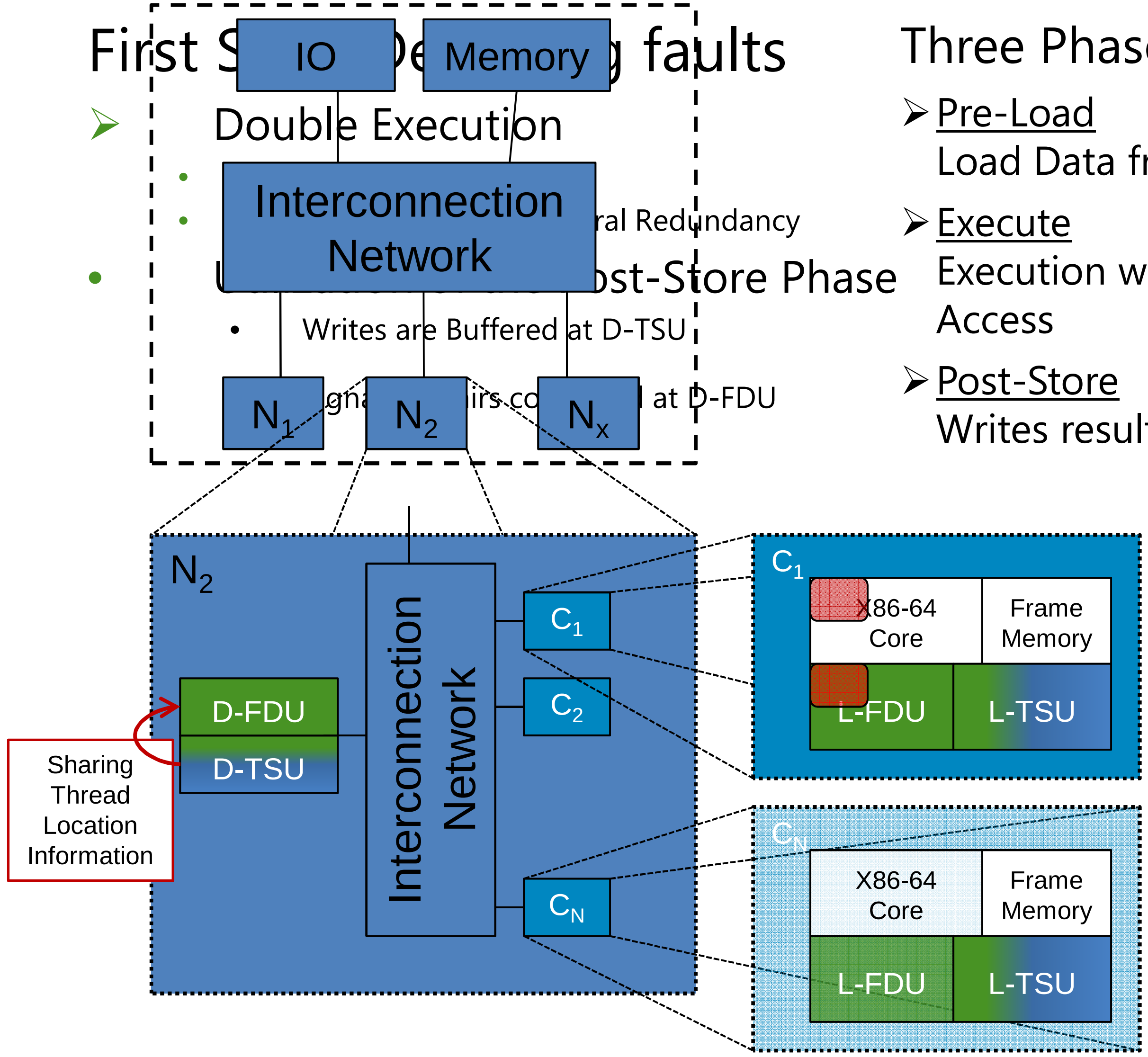
How it works

- Threads can be generated dynamically. At that point we assume that new threads are generated at the service cluster and being distributed to the clusters again
  - next step we will distribute the algorithm
- Health information and load balance
  - Cores sends health information (e.g., speed, temperature, number tasks completers, etc.) to the cluster-FDU
  - The Cluster-FDU sends the information to the service-cluster
  - The service cluster take the health conditions of the cluster into account in order to load balance new threads.

Reliability comes into play

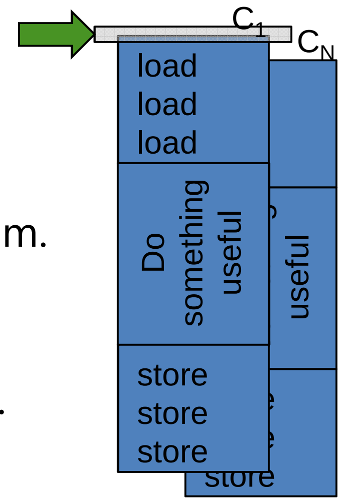
First S De g faults

- ➢ Double Execution
  - ral Redundancy
- U ost-Store Phase
  - Writes are Buffered at D-TSU
  - gna irs co l at D-FDU



Three Phases

- ➢ Pre-Load
  Load Data from Frame Mem.
- ➢ Execute
  Execution with no further Mem. Access
- ➢ Post-Store
  Writes results to Frame Mem.



L-FDU:
Sniffing Outgoing Writes
Caculates Signature from the values written values

Reliability comes into play

First Step: Detecting faults

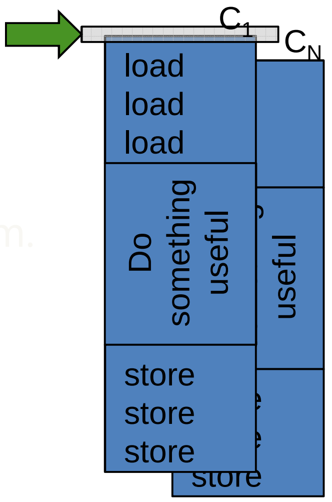
- Dual Module Redundancy
  - Thread Duplication
  - Spatial and Temporal Redundancy
  - Writes are redirected and compared

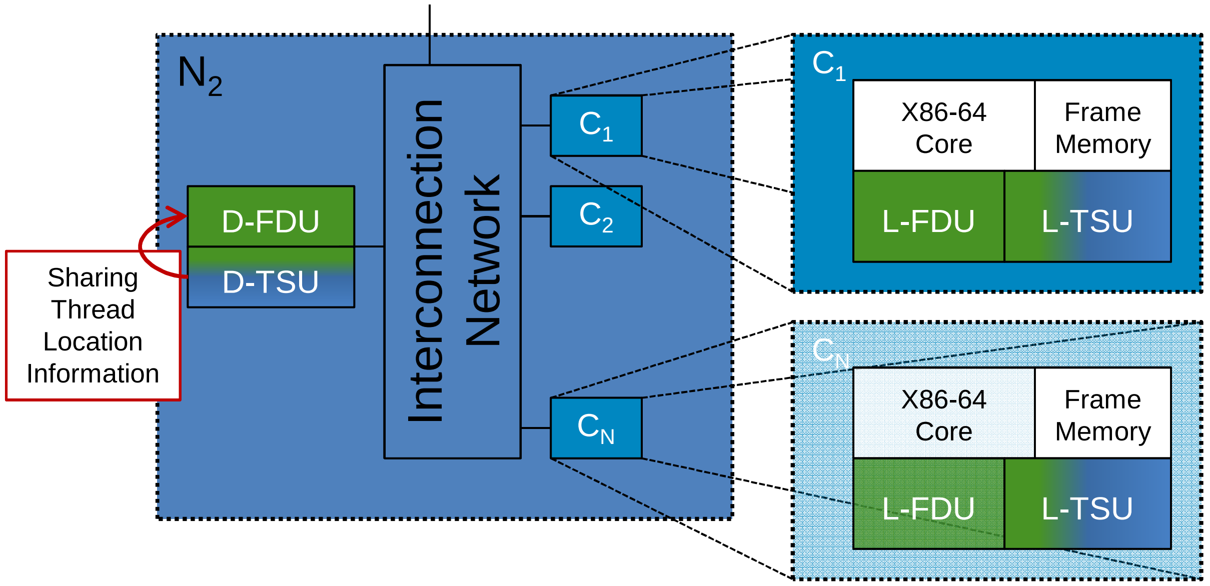
Second Step: Fault Recovery

- Thread re-execution Mechanism
- Utilization of the DF Semantics
- Single Assignment rule & Side Effect Free execution
- Utilization of the Post-Store Phase
- Shared Thread Information at the D-FDU

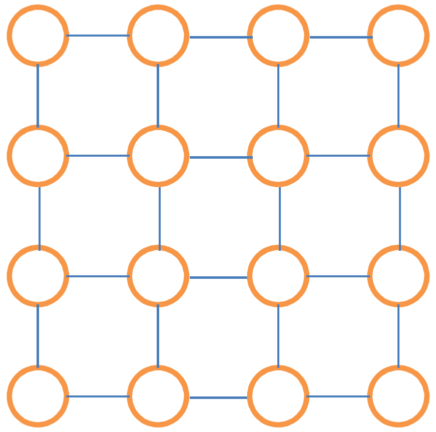
Three Phases

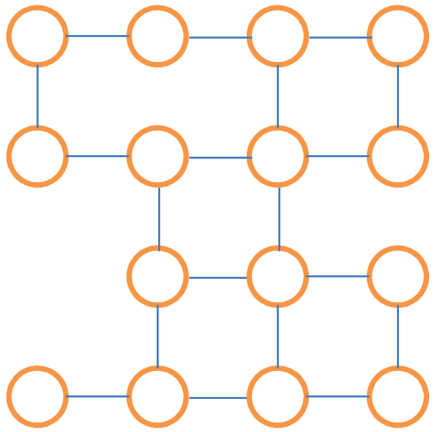
- Pre-Load
  Load Data from Frame Mem
- Execute
  Execution with no further Mem. Access
- Post-Store
  Writes results to Frame Mem.

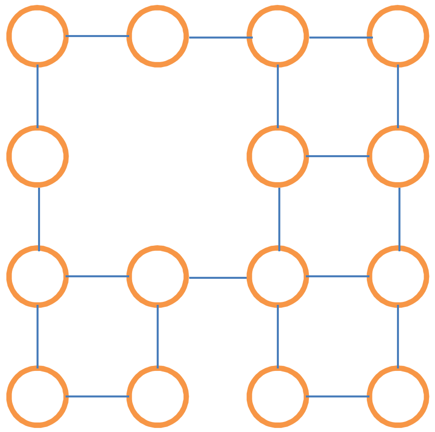




WP5 Reliability – Overall Goal


Basic mesh


chip1


chip2

- Target of software remains always the basic mesh
- Faulty elements are detected and hidden by low-level software virtualization (OS, FDUs, cores, FDU/TSU)

Soft-Errors (Transient errors) - WIP

Classification

- Detection and handling soft-errors can be relatively simple or extremely difficult depending on the assumptions and HW mechanisms we are introducing. At that level of the research we are focusing on the following assumptions:
  - All memory structures and buses are shielded.
  - The DF mode of operations we described before allows to terminate an execution w/o any side effects
  - We assume that if the “update global memory” phase began, eventually it will be completed
- Base of these current assumptions (that most likely will be refined later on), at that point we are focusing on detection errors in the control logic so we can indicate that an error occur.

Detection mechanism – re-execution

- Can be done via space redundancy of time redundancy
  - Space redundancy: execute the code on 2 cores (3 are needed for recovery but only 2 for detection), compare the observable outputs and raise a flag if found not to match
  - Time redundancy: execute the code twice on the same core and compare results. If dine smartly can cost only 4-10% performance hit.
- Need to take care on endless loops and few other corner cases
- Need to address the I\O, exceptions, etc.

Checkpoint

- Checkpoints are needed if we like to optimize for performance. For example, allow to update global memory before we conclude the execution of the current one.
- Checkpoint can be done at different levels the tradeoffs are between fast generate, save and restore but need to be done at a very fast paste, and slow generation that takes long time to generate and retrieve, but done infriquest.

Summary

- We present a system level FT mechanisms
- We take advantage of the cluster architecture and the Data-Flow
- Memory model, coherency and resource management may have a major impact on the structure of future architectures.
- FT and reliability needs to be considered as first class citizens.

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

Thanks