The Teraflux approach for massive parallel processing on-die

Prof. Avi Mendelson,
Technion & Microsoft R&D, Israel

2nd Workshop on Future Architecture Support for Parallel Programming (FASPP12)

Agenda

- Introduction
- What is Teraflux?
- General design approach
  - SW
  - HW
- System level
  - Faults
  - OS

Moore's Law – historical note

- Two of the many versions of Moore's Law
  - Number of transistors on a die doubles every 18 months (the original form)
  - Measured performance of computer systems doubles every two years (one of many variations)
- Implications:
  - Enables adding value to the user
  - Enables innovation
  - Enables new applications and markets
  - **Allows to maintain prices and revenue**

What's had been changed?

- Process stop scale "ideally"; we can still get the growth in transistor budget but with cost of power luck of frequency scale.
- Small sizes of transistors implies (1) higher rate of soft-errors increases and (2) process variation in respect to power and performance
- In order to meet the expectation of constant growth of "benefit to the user", number of cores on die increases.
- It is expected that in the near future we could put tens or even hundreds of cores on the same die (Silicon)
- How many cores SW can use?

It is mostly depends on software/OS/algorithms/developing environment/etc., rather then on HW capabilities.

TERAFLUX.EU

Exploiting Dataflow Parallelism in Teradevice Computing

What is **TERAFLUX** about?

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

University of Siena

Barcelona Supercomputing Center

University of Augsburg

University of Cyprus

INRIA

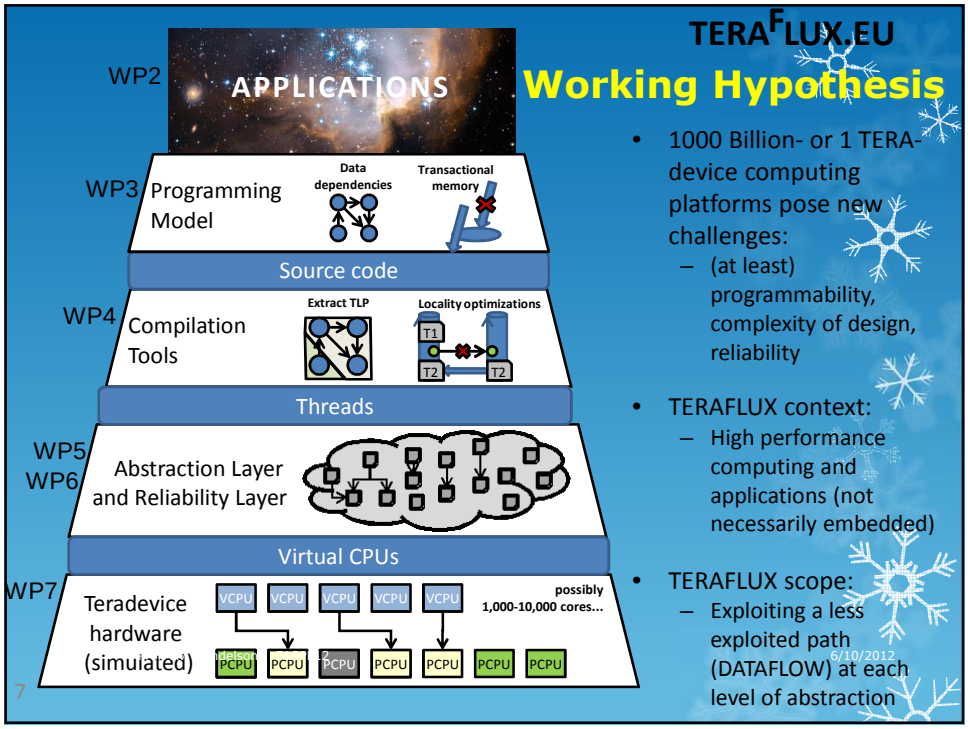
FUNDING OPPORTUNITIES from the FUTURE & EMERGING TECHNOLOGIES scheme

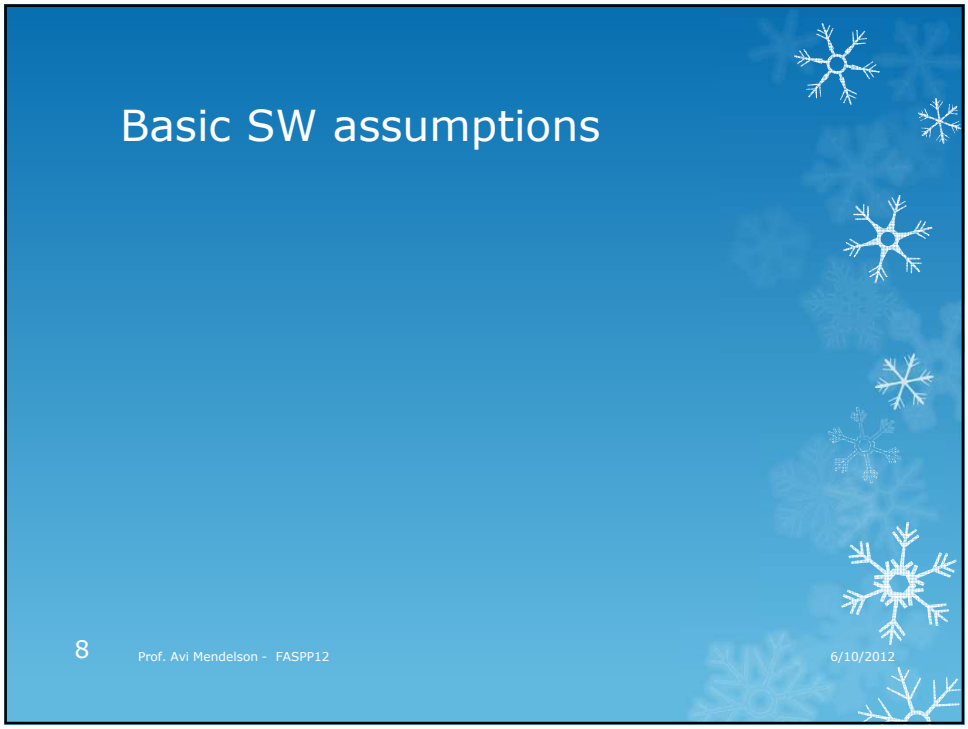
University of Manchester

Teraflux in a nutshell

- An EU research project (FET).
- Assumes 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


TERAFLUX.EU
Working Hypothesis
WP2
APPLICATIONS
WP3
Programming Model
Data dependencies
Transactional memory
Source code
WP4
Compilation Tools
Extract TLP
Locality optimizations
T1
T2
T2
Threads
WP5
WP6
Abstraction Layer and Reliability Layer
Virtual CPUs
WP7
Teradevice hardware (simulated)
VCPU
PCPU
possibly 1,000-10,000 cores...
• 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
7
6/10/2012


Basic SW assumptions
8
Prof. Avi Mendelson - FASPP12
6/10/2012

Fundamental approach (General):

General Purpose

- Target to run any program in a reasonable performance and power consumption
- Mostly assume to be latency sensitive
- Use "reverse engineering" (e.g., branch prediction) to unveil the internal structure of the program

Special purpose

- Targeted specific class of applications
  - Applications the don't fit into this category may not run or run in a very inefficient way.
- Usually Use SW/HW co-design
- Can be an order of magnitude more efficient than general purpose architectures for specific class of application

Fundamental approach (Teraflux):

- The system is dynamically partitioned between cores that can run General purpose applications and cores that can run "special purpose" accelerator code, A.K.A Teraflux cores/
- The code for the Teraflux cores is based on Special branch of the DataFlow paradigm, called Task-Parallelism (similar to Actors)
- The Teraflux cores subsystem is built as SW/HW codesign

Data-Flow is back

Back to the future

- Dataflow can extracts massive parallelism out of sequential code
- HW only methods; e.g., OOO, are not sufficient. Parallelism needs to be exposed at all levels, such as compilers, algorithms, tools, etc.
- Dataflow languages are limited. Needs to apply DF techniques for procedural and shared memory based languages such as C, OpenMP and hybrid languages such as Scala and Heterogeneous OpenMP (OpenMP+MPI)
- Use of modern HW techniques to overcome performance power and reliability issues

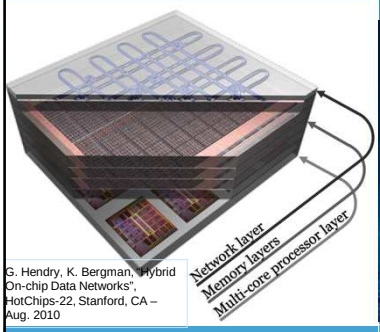
Data Flow model

- DataFlow was defined by Prof. Dennis as "A Scheme of Computation in which an activity is initiated by presence of the data it needs to perform its function"
- Data flow preserves "pure execution"; i.e., no side effects
- Task parallelism reserve (1 & 2) while doing it at the level of tasks (a collection of instructions)
- The code is generated automatically (in the future) from C, Java , OpenMP, etc. programming languages.
- The "pure execution code is achieved by a combination of run-time scheduling an/or transactional memory (out of the scope of this discussion.
- Task can be suspended at any point (for any reason) and be re-executed if needed

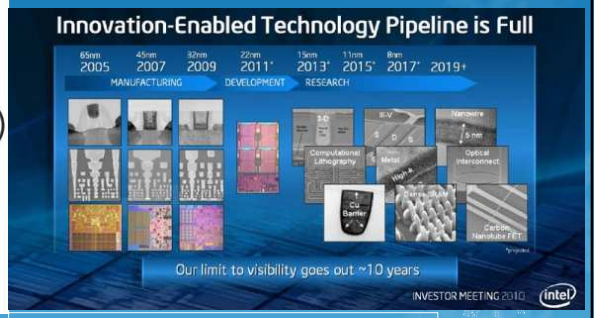
Basic HW assumptions

Future Scenarios

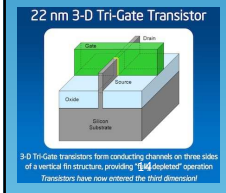
== 3D stacking, 8nm, 3D transistors, Graphene

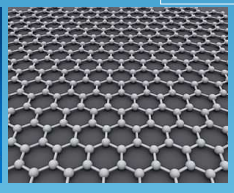


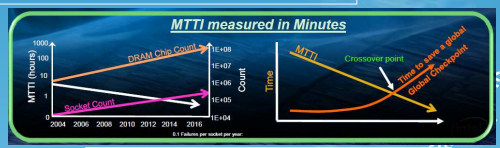
G. Hendry, K. Bergman, "Hybrid On-chip Data Networks", HotChips-22, Stanford, CA – Aug. 2010



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



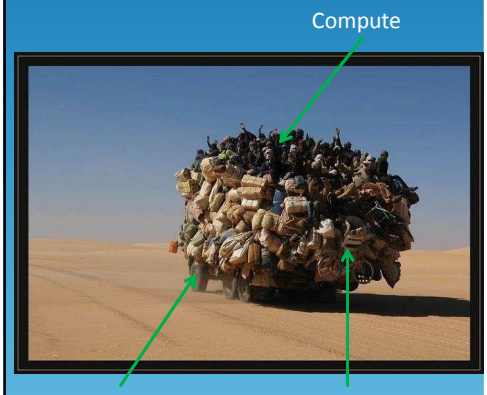




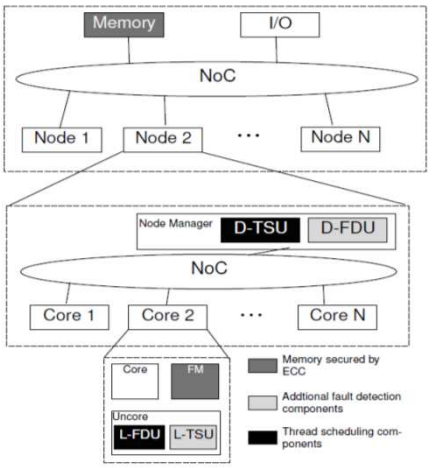
Pawloski, May 2011, Exascale Seminar, Ubert

How to fit 1000 cores on die?

The unstructured option



The Structured (hierarchical) option



Basic Architecture

TERAFLUX

- 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

Put it all together

TERAFLUX

System Overview Target System

Cores View

Linux

CPU CPU CPU CPU

CPU CPU CPU CPU

CPU CPU CPU CPU

CPU CPU CPU CPU

L4

CPU == Cluster

Memory View

L4

Configuration Page

Message Buffers

Linux

Target System OS Requirements

Linux (Full OS)

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

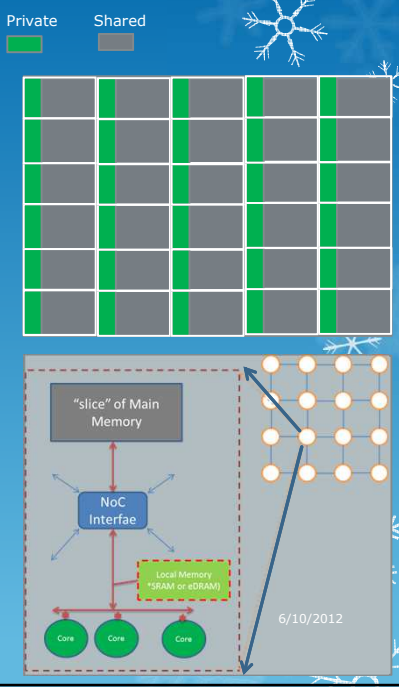
Single chip
Multi cores

- 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

L4 (uKernel)

Memory hierarchy

- The Teraflux virtual address space is divided into equal disjoint "segments"
  - A segment is controlled by a cluster
  - Coherency is handled by HW within the cluster (sequential consistency)
- Physical memory may be partitioned into local memory vs. global memory
- The "physical memory" as was seen by the User (OS) is the collection of all Global Memory parts (Segments) connected to the NOC
- Between clusters no HW coherency



How it works

TERAFLUX

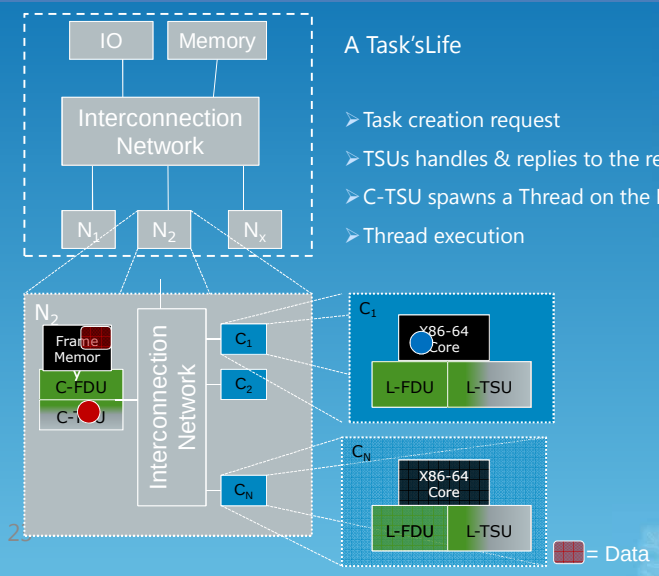
- Compiler generate DF code out of sequential code (e.g., C) or programing languages that support parallelism (e.g., OpenMP, Java, Scala)
- The execution always starts on the service cores that generate the Tasks (Tokens) and send them to the different clusters.
- All tasks sent to a cluster are kept 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 Task is removed from the clustered queue.

How it works

TERAFLUX

- 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-level
  - The Cluster-level sends the information to the service-cluster
  - The service cluster uses the health conditions of the cluster into account when decide where to create new Tasks.

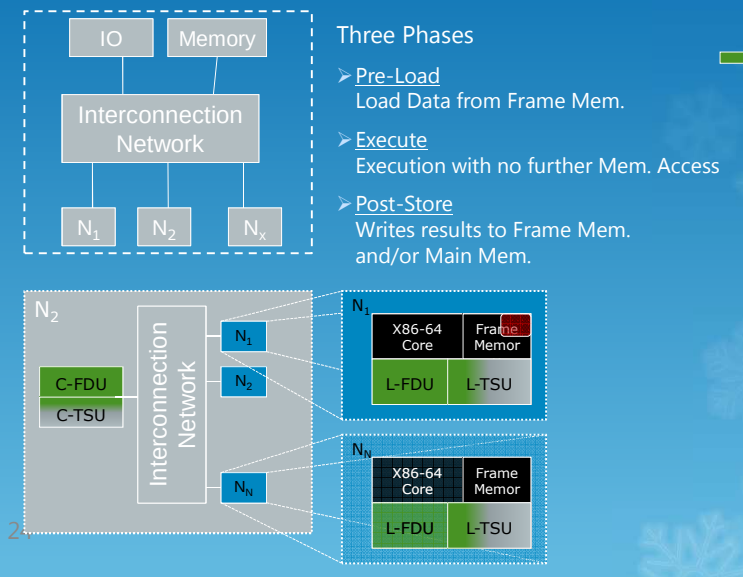
Task life



A Task'sLife

- Task creation request
- TSUs handles & replies to the request
- C-TSU spawns a Thread on the L-TSU
- Thread execution

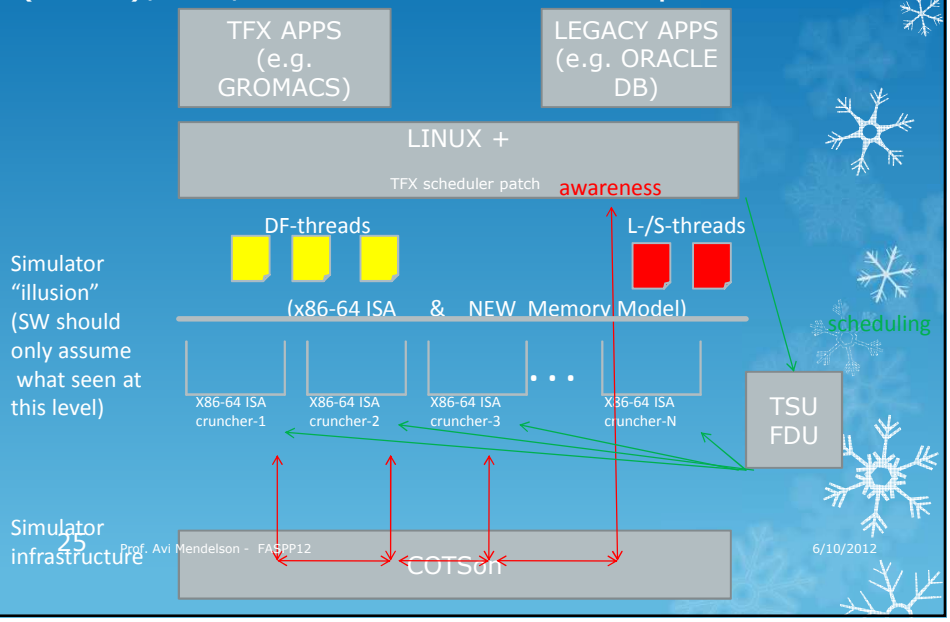
A Thread Execution



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.

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



Faults in large systems

TERAFLUX

Two types of faults

Hard/Permanent faults

If re-execute – the fault remains The same.

Soft/transient faults

If re-execute – the system behaves correctly.

For large scale systems, one should assume that the probably for a faulty part at any given time is significant.

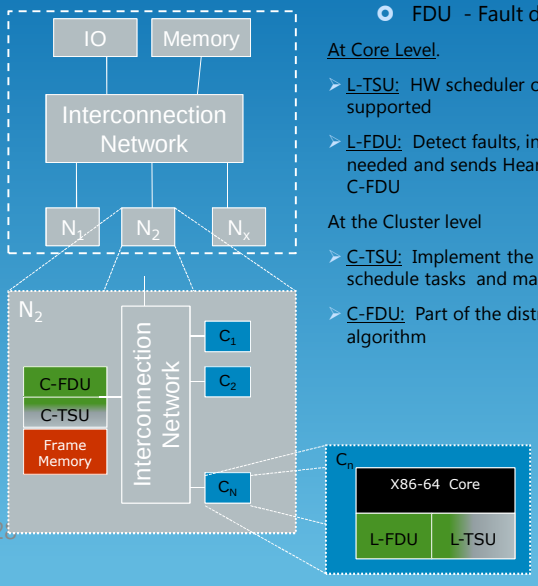
A reliable system with 1000s of processing elements should be built of non-reliable components

At Teraflux we decide to address faults, at all levels as "first-class citizens".

Reliability and Fault-Tolerant – high level

- We use a SW/HW co-design in order to address the very complicated issue
  - At SW level we take advantage of the DF model that allows to re-execute a "task" w/o causing side effects.
  - At HW level we build the system to detect faults, to avoid single point of failure and to dynamically reconfigure
- FT is implemented at all different levels of the hierarchy
  - At the Global Level – the Linux OS manages 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

Management and Reliability Components



At each level of the Hierarchy we have 2 dedicated HW/SW units to help handling faults

- TSU – Task Scheduling Unit
- FDU - Fault detection Unit

At Core Level.

- L-TSU: HW scheduler of tasks (if MT at core level is supported
- L-FDU: Detect faults, indicates that re-execution is needed and sends Heartbeat (HB) Messages to the C-FDU

At the Cluster level

- C-TSU: Implement the match-logic, communication, schedule tasks and maintain load balance
- C-FDU: Part of the distributed Fault Detection algorithm

Soft-Errors (Transient errors) - WIP

TERAFLUX

Classification

TERAFLUX

- 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

TERAFLUX

- 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.

Future work

TERAFLUX

- Heterogeneous cores
  - Same ISA
  - Different ISA
  - System on a chip
  - Combination
- Multi-chips
- OS for heterogeneous systems
- Memory hierarchy
- Distributed I/O
- Distribute the system level algorithms.

TERAFLUX.EU

Exploiting Dataflow Parallelism in Teradevice Computing

Microsoft R&D Israel

THALES

University of Siena

BSC Barcelona Supercomputing Center

FUNDING OPPORTUNITIES from the FUTURE & EMERGING TECHNOLOGIES scheme

University of Augsburg

CAPS

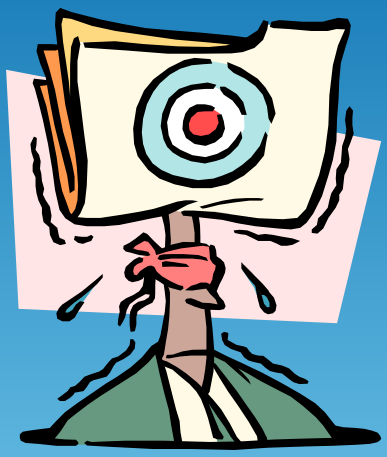
LABS hp

INRIA

MANCHESTER 1824 University of Manchester

University of Cyprus

Questions?



backup

TERAFLUX