SCoRPiO: Significance based Computing for Reliability and Power Optimization

http://www.scorpio-project.eu/

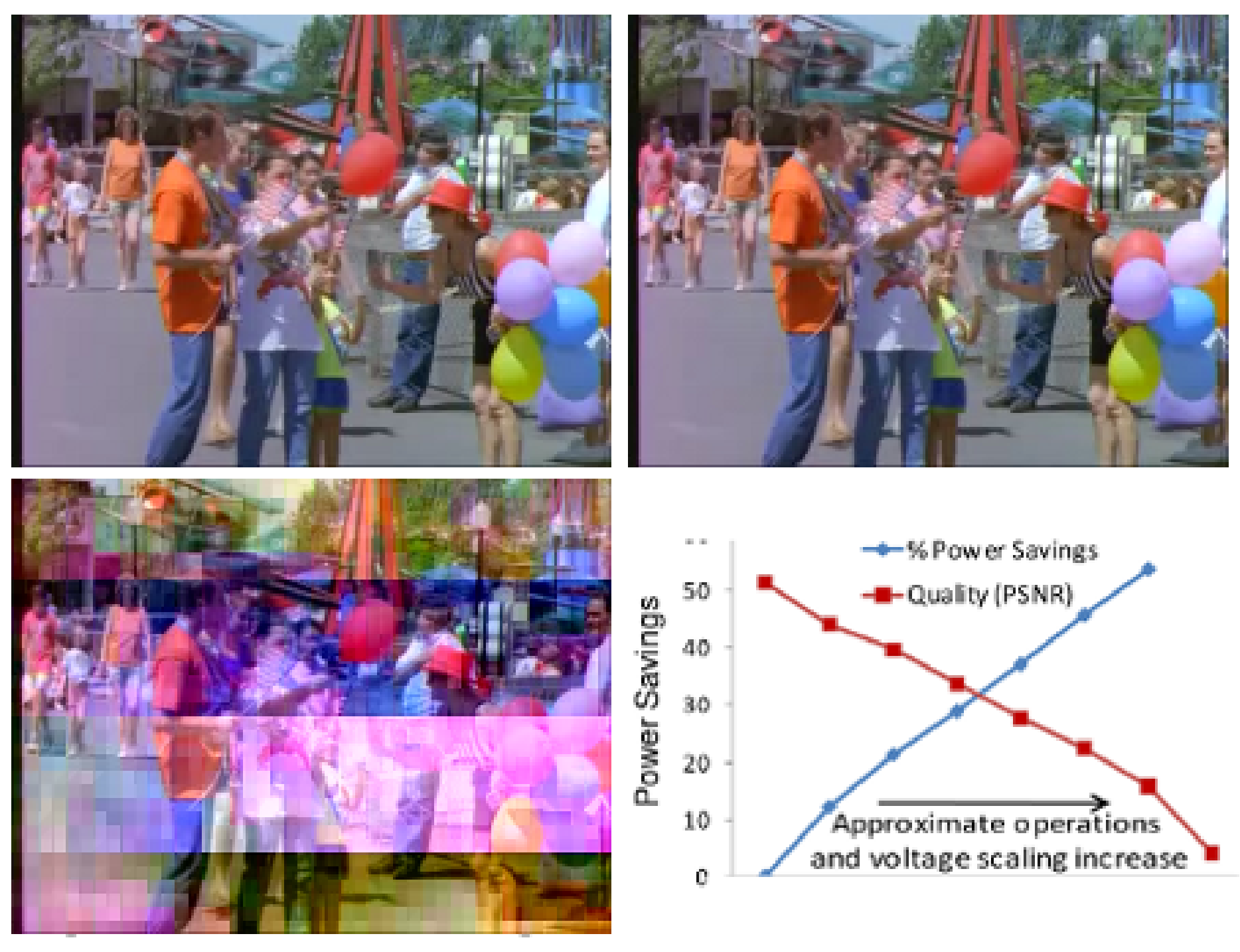
This poster has been prepared by the CERTH research group with input from RWTH-Aachen, EPFL and QUB research groups.

1. Vision

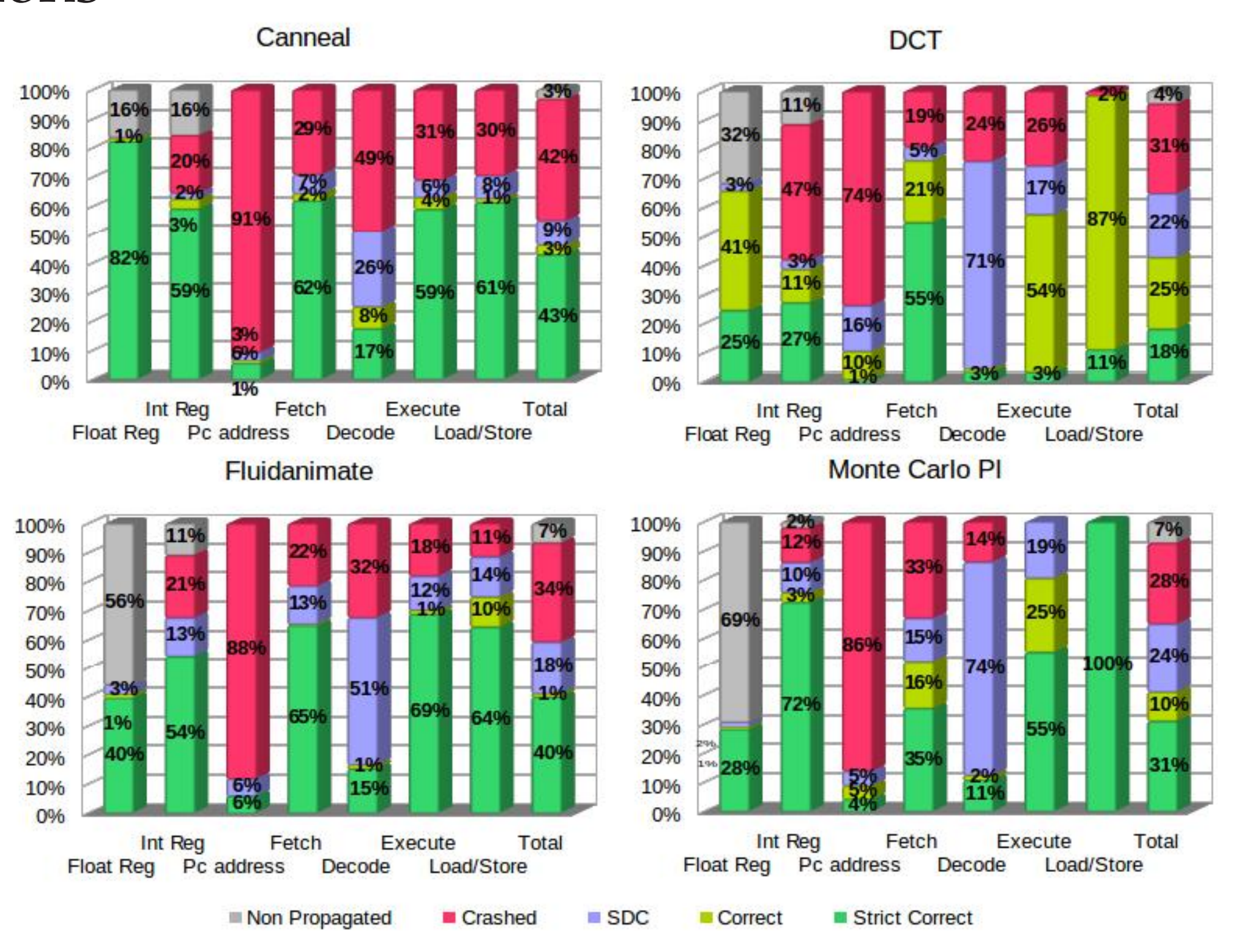
Just like parallelism, we want to elevate computational significance as a first class concern in the design of algorithms and systems.
Exploit computational significance to **build energy efficient hardware and software platforms that scale gracefully in case of errors induced by scaled voltages and/or parametric variations.**

2. Motivation

- Power/Energy dissipation remains #1 constraint of future systems.
- Several application domains offer the opportunity to trade-off quality of service for significant improvements in energy comsumption (e.g. JPEG).

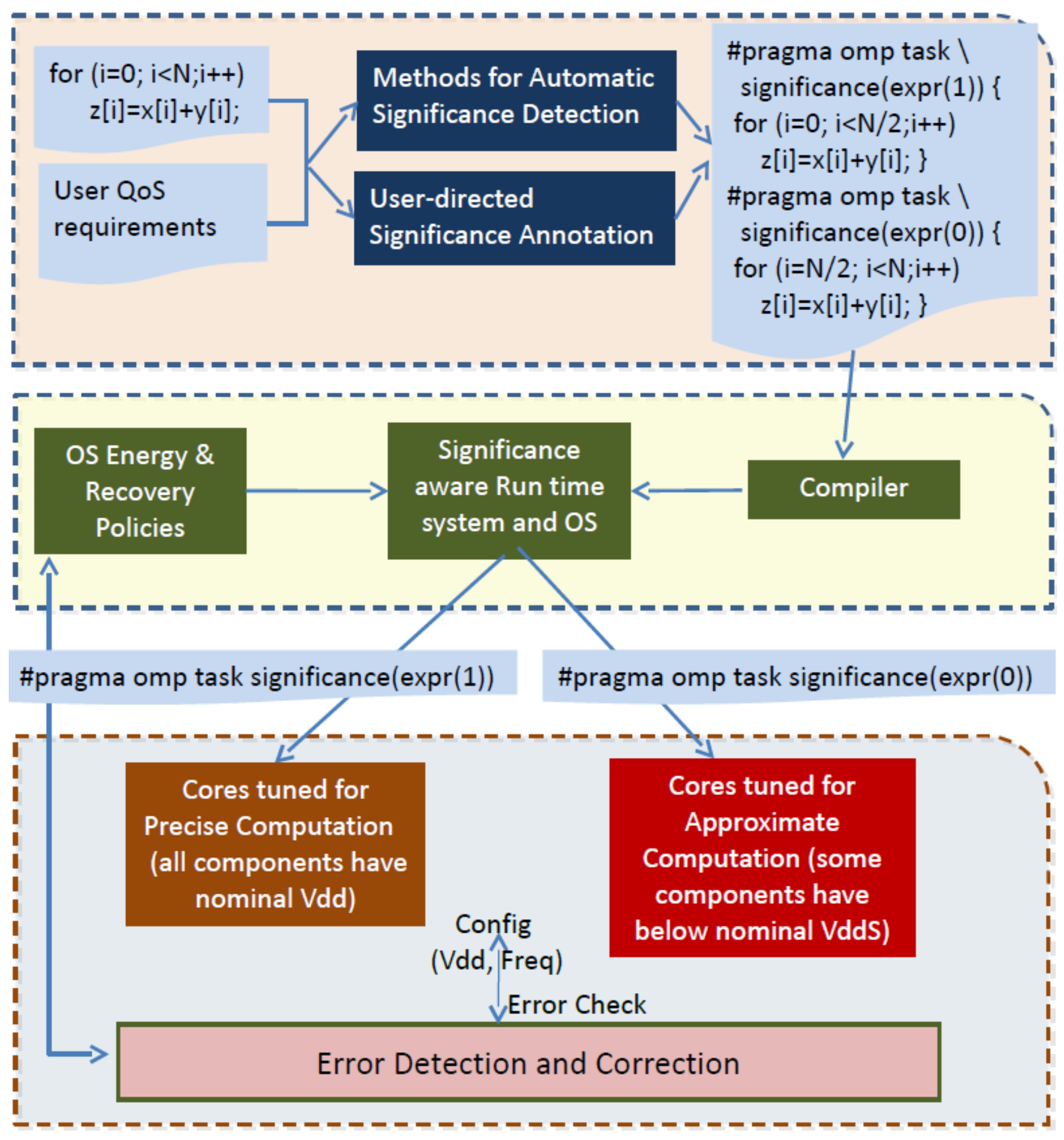


- Preliminary fault injection trials on GEM5 simulator suggest that tolerance to injected faults was highly dependent on the spatial location of the faults as well as the specific portion of the affected code.
- PC and IFetch modules are very vulnerable even to single errors.
- Arithmetic operations are often error tolerant, especially if they are not used for address calculations



3. Objectives

- Introduce computational significance as an algorithmic property and expose it to the level of the programmer.
- Devise techniques that facilitate automatic characterization of code and data significance using compile-time or runtime analysis.
- Lay the necessary foundations at all levels of system stack allowing controlled quality execution on unreliable hardware substrates.
- Enable drastic power dissipation reduction by opportunistically and aggressively powering parts of the platform below nominal values in a targeted way.
- Produce a vertically integrated system prototype to prove the efficacy of significance-based computing.



for (i=0; i<N;i++)
z[i]=x[i]+y[i];
User QoS requirements
Methods for Automatic Significance Detection
User-directed Significance Annotation
#pragma omp task \
significance(expr(1)) {
for (i=0; i<N/2;i++)
z[i]=x[i]+y[i]; }
#pragma omp task \
significance(expr(0)) {
for (i=N/2; i<N;i++)
z[i]=x[i]+y[i]; }
OS Energy & Recovery Policies
Significance aware Run time system and OS
Compiler
#pragma omp task significance(expr(1))
#pragma omp task significance(expr(0))
Cores tuned for Precise Computation (all components have nominal Vdd)
Cores tuned for Approximate Computation (some components have below nominal VddS)
Config (Vdd, Freq)
Error Check
Error Detection and Correction

4. Semi-Automatic significance analysis

- Introduce a rigorous mathematical definition of computational significance.
- Apply a set of criteria to (semi)automatically characterize significance.
- Partition source code to slices of varying significance.
- Example of interval arithmetic and algorithmic differentiation:
- Original code:

v1 = log(x1);
v2 = v1 + x2;
y = v2;

- Input range for x1 : [1, 2]
- Input range for x2 : [1, 20]
- Significance criterion : $w([v] \cdot \nabla_{[v]}[y]) > \epsilon$
- Significance bound : $\epsilon = 1$

	$[v]$	$\nabla_{[v]}[y]$	$[v] \cdot \nabla_{[v]}[y]$
$[x1]$	[1, 2]	[0.5, 1]	[0.5, 2]
$[x2]$	[1, 20]	[1, 1]	[1, 20]
$[v1]$	[0, 0.693]	[1, 1]	[0, 0.693]
$[y]$	[1, 20.7]	[1, 1]	[1, 20.7]

Interpretation: Intermediate $v1$ turns out to be insignificant over the complete input ranges of $x1$ and $x2$, since $w([v1] \cdot \nabla_{[v1]}[y]) < \epsilon$. Using the midpoint $m[v1] = 0.3465$ of $v1$ as constant initializer for $v1$, the code can be simplified for the complete input range of $x1$ and $x2$ to:

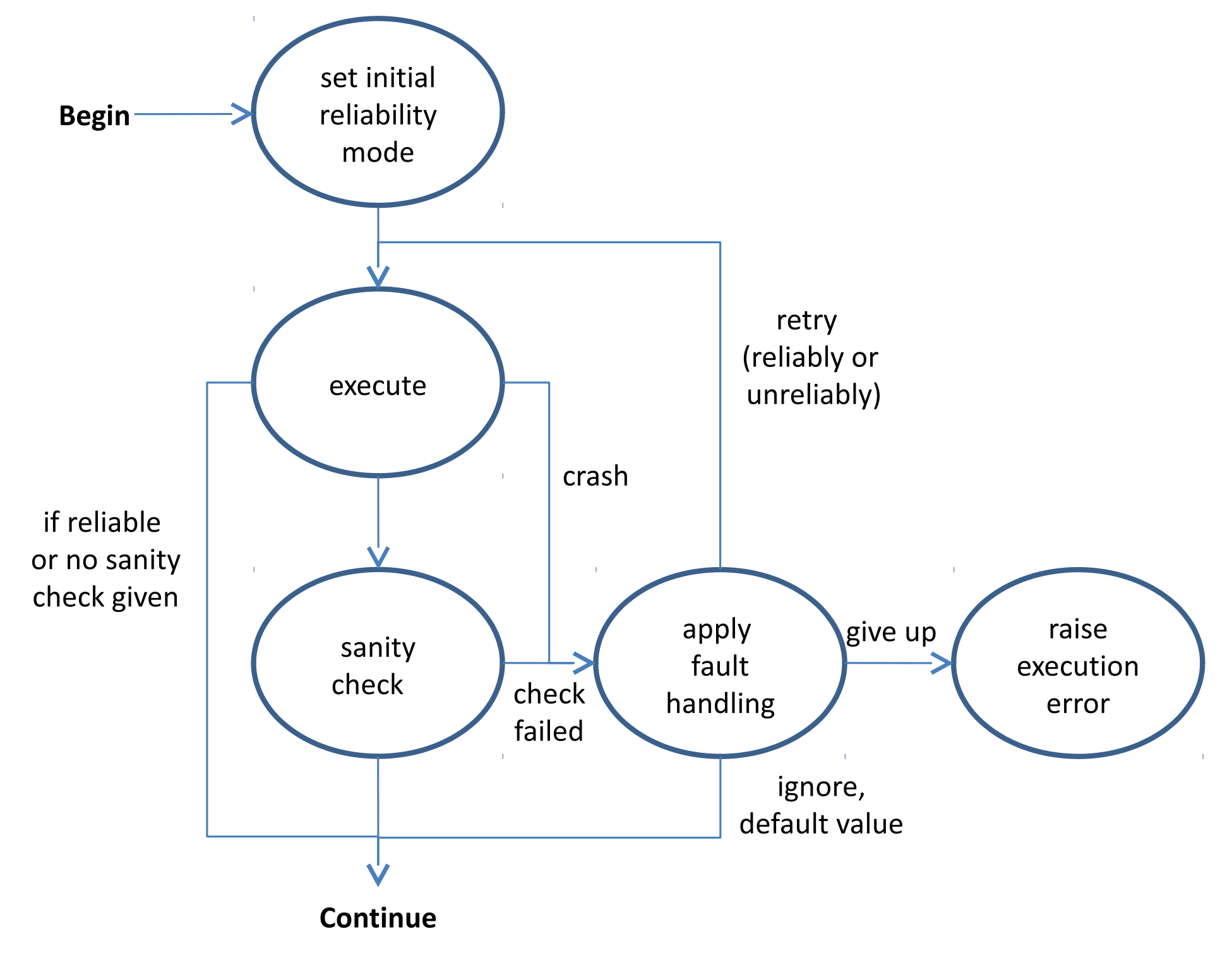
v1 = 0.3465;
y = v1 + x2;

Interval evaluation of the modified code for ranges $[x1] = [1, 2], [x2] = [1, 20]$ gives $[y] = [1.35, 20.3]$, which is a sub-interval of the output range of the original code.

5. Task-based programming model and runtime system

Programming Model:

- Code is tagged with significance information at task or even subtask level.
- Significance determines the (un)reliability of the core used for task execution.
- Offer recovery mechanisms coupled with sanity checking functions.



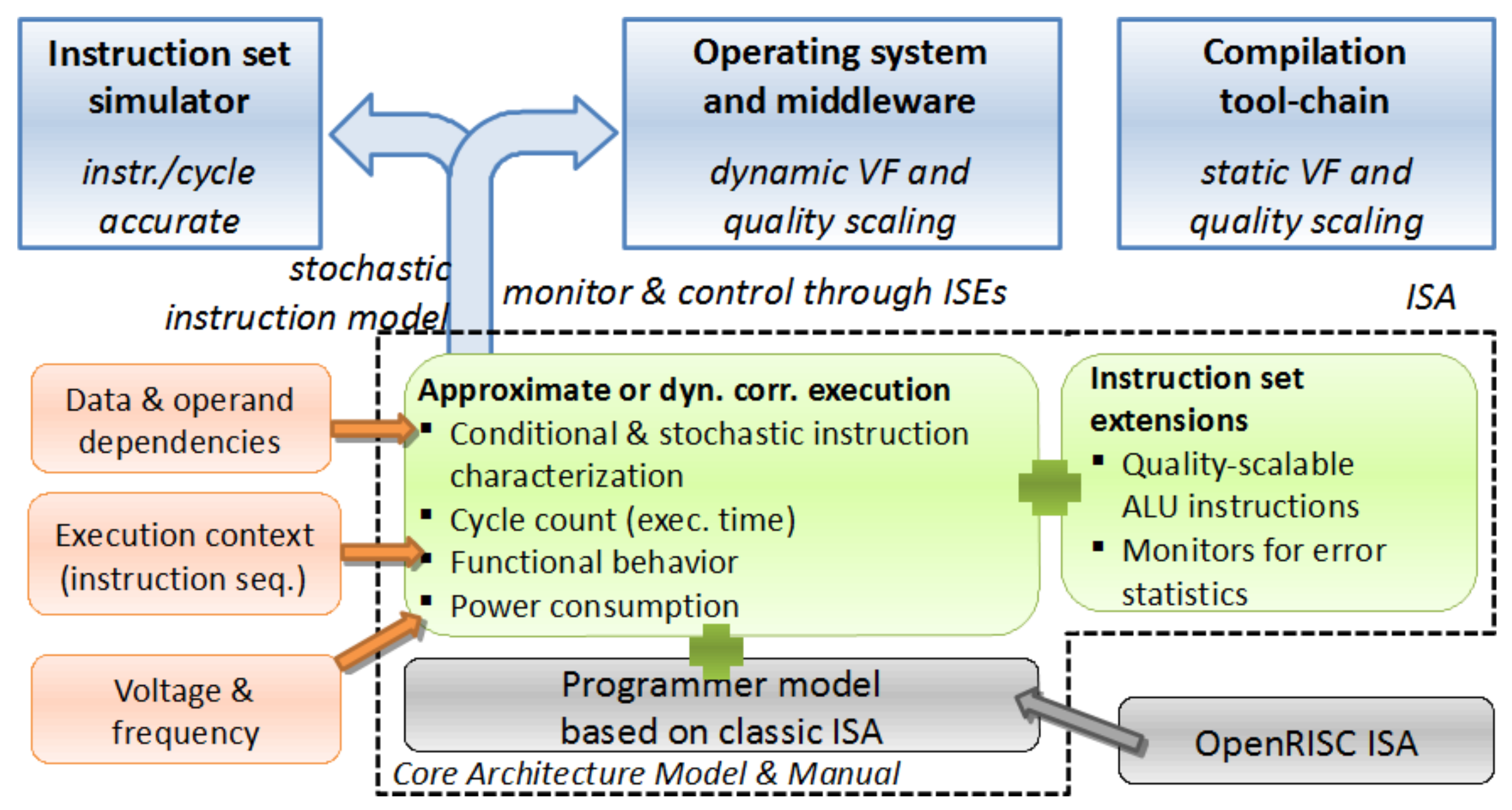
- Develop and deploy smart fault-handling mechanisms for graceful performance degradation such as statistical error correction.
- Pragmas for **instantiation and synchronization of tasks** (ompss extensions).

```
for(i = 0 ; i < WORKERS ; ++i) {
#pragma omp task label(work) significant(ratio(0.5f))\
 out(b[i*10,i*10+9]) in(a[i*10,i*10+9]@[149,180])\
 taskchecking(sanity(task_sanity, i), redo(2))
  task(&a[i*10], &b[i*10])
}
#pragma wait label(work) ratio(0.8f) time(1000,0)\
 groupchecking(sanity(group_sanity), redo(2))
is_correct = sgnf_groupsanity(work);
if( is_correct = SGNF_SUCCESS)
  printf("results are correct");
else
  printf("results are wrong");
```

Runtime System:

- Component-level control of power, performance and reliability
- Significance-aware scheduling and memory management
- Dynamic optimization under reliability constraints
- Significance-aware runtime system monitors the dynamic behavior of the cores:
  - Did the core crash due to a fault?
  - If not, how many faults could the hardware detect?
- Based on the information conveyed by the hardware, the runtime system calls the sanity function and takes corrective actions.

6. Hardware modeling and design



Instruction set simulator
instr./cycle accurate
Operating system and middleware
dynamic VF and quality scaling
Compilation tool-chain
static VF and quality scaling
stochastic instruction model
monitor & control through ISEs
ISA
Data & operand dependencies
Execution context (instruction seq.)
Voltage & frequency
Approximate or dyn. corr. execution
Conditional & stochastic instruction characterization
Cycle count (exec. time)
Functional behavior
Power consumption
Instruction set extensions
Quality-scalable ALU instructions
Monitors for error statistics
Programmer model based on classic ISA
OpenRISC ISA
Core Architecture Model & Manual

- Develop instruction level power and behavior models under various degrees of voltage scaling and variations.
- Exploit the dynamic timing profile and modify the circuits in conjunction with the core microarchitecture to enable graceful performance degradation.
- Cores are enhanced with low cost error detection/correction mechanisms to help in adjusting their reliability and energy efficiency.
- Extended ISA to support both approximate and accurate instructions.
- A simulator that supports the overall significance driven vertical stack and allows its evaluation at various operating modes will be developed.

This work has been supported by the EC within the 7th Framework Programme under the FET-Open grant agreement SCoRPiO, grant number 323872.