Dynamic Evaluation of Hardware Trust

D. McIntyre
Computer Science
Cleveland State University
Cleveland, Ohio 44106, USA
mcintyre@csuohio.edu

F. Wolff, C. Papachristou, S. Bhunia
Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, Ohio 44106, USA
{fxw12,sxb21,cap2}@case.edu

D. Weyer
Engineering Services
Rockwell Automation
Cleveland OH 44124
djweyer@ra.rockwell.com

***Abstract*— Current research into Trojan detection suggests that exhaustive Trojan detection in a chip during limited manufacturing test time is an extremely difficult problem. Indeed, an especially nefarious form of Trojan known as the time bomb has a payload activated in a delayed manner making it extremely hard to detect. As a result, chip trust detection at manufacturing test time may not be adequate especially for critical applications. This suggests that some form of dynamic trust detection of the chip both preliminary (possibly during a preproduction phase) and during in-field use at run time is required. We explore an approach to this problem that combines multicore hardware with dynamic distributed software scheduling to determine hardware trust during in-field use at run time. Our approach involves the scheduling and execution of functionally equivalent variants (obtained by different compilations, or different algorithm variations) simultaneously on different PEs and comparing the results. The process dynamically achieves trust determination by identifying the existence of Trojans with a high level of confidence.**

I. Introduction

Insertion of malicious circuits (referred as Hardware Trojan) in a design during the chip manufacturing process has emerged as a major security issue [1]. Existing Hardware Trojan detection methods using manufacturing test either target Trojan detection using conventional logic testing [2] or side-channel analysis such as power supply monitoring [3], [4] and delay testing [5], [6]. These approaches however do not guarantee exhaustive detection of hardware Trojan which can vary in circuit size, trigger function and payload. Since complete verification of trust of a chip during production test is extremely challenging, an in-field procedure for online checking of possible malicious effects due to hardware Trojan can potentially assure trustworthy computing.

In this paper, we propose an in-field dynamic trust verification approach in multicore system using distributed scheduling, which dynamically evaluates the trustworthiness of each core during task execution. In the process, the trustworthiness of the system evolves over time. It gradually learns to do scheduling and allocation of tasks in manner that assures high confidence as well as minimal impact on system throughput.

The advent of multicore processing suggests the use of multiple processing elements (PEs) or cores on the same chip, allowing both simultaneous execution of the same functionality combined with verification. Multicore systems offer the additional benefit of redundancy so that as trust in the cores are evaluated, centralized or distributed software scheduling algorithms can be used to avoid low-trust cores. Also more refined scheduling can be devised to match job trust requirements to evolving core trust levels, e.g. more critical jobs being assigned to more trustworthy cores.

The intent of such a multicore task scheduler is to gradually, over time, determine the hardware trust of each core while permitting the in-field use of the hardware system in an increasingly trusted manner. Additionally, as individual core trust evolves, the scheduler [7] can perform with improved efficiencies (both in utilization and speed) by matching desired task trust levels with discovered core trust.

II. Multicore System Description

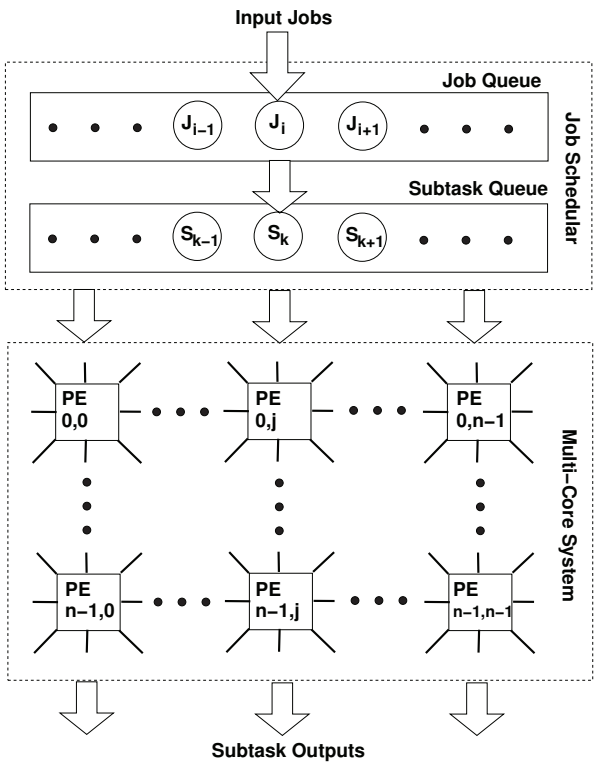

Fig. 1. Multi-core architecture using $n \times n$ array of PEs and taurus interconnection network

Assume an $n \times n$ toroidal array of PEs with n PEs at row 0 capable of reading in subtasks as shown in Fig. 1. Assume that all n PEs on row $n-1$ are output PEs capable

978-1-4244-4804-3/09/$25.00 © 2009 IEEE

of outputting computed values from subtasks. We assume each job is composed of several interdependent subtasks that have been topologically sorted and placed into an input queue available to the PE array for input. The PE array is free to read in any subtask, S_k, from any job, J_i, provided all its inputs have be satisfied by previous subtasks that have completed, or already have values. Basically subtasks are read by input PEs in row 0 and begin their search for a free PEs to execute in. As a subtask moves from PE to PE in its search for a free PE, path information is stored in the PEs so that values can be sent back to row 0 and then (in one hop) to an output PE in row $n-1$. Also the path is used to determine trust as will be discussed.

For the proposed dynamic trust evaluation approach in a multicore platform, we note that our distributed scheduler can be a better choice than a centralize one. This is because as centralized scheduler (either hardware or software) can itself be vulnerable to malicious attack. Hence, we have considering a distributed scheduling framework in this paper to achieve trustworthy computing.

III. Using Subtask Variants to Determine Hardware Trust

Trojans are typically well hidden in hardware circuitry and depend upon a small number of input triggers to activate them. In addition, the activation may require previous states to have been set in combination with specific trigger activation values. Also, the delayed payload activation property of time-bomb Trojans makes them extremely hard to detect. This sensitivity suggests the unlikelihood of two functionally equivalent variant processes (for example, binaries for two different algorithms or different compiled version of the same algorithm) A and B, say, of a subtask both triggering the same Trojan. Therefore when two binary variants of a subtask are simultaneously executed on two different multicore PEs, $\mathrm{PE_A}$ and $\mathrm{PE_B}$, say, and their computed values $\mathrm{V_A}$ and $\mathrm{V_B}$ agree, it is highly unlikely that both executed the same Trojan (or different Trojan) and as a result obtained the same result. Thus it can be concluded with high confidence, that $\mathrm{V_A} = \mathrm{V_B}$ is the correct value of the subtask and it's value can be sent to the output PEs. Also if their computed values disagree, it is known with certainty that at least one of either $\mathrm{PE_A}$ or $\mathrm{PE_B}$ contained a Trojan (i.e. it is known with certainty that a Trojan has been detected). If they disagree, then the next step is to determine which of the PEs (or if both) contained a Trojan and reduce the trust level of the PE containing the Trojan. This can be accomplished by repeating the process and running an additional variant, C, say of the subtask. Then if $\mathrm{V_C} = \mathrm{V_A}$ (similar argument if $\mathrm{V_C} = \mathrm{V_B}$) then again it is highly unlikely that both variants C and A executed the same Trojan (or different Trojans) and as a result obtained the same result. Thus it is determined with high confidence, that $\mathrm{V_C} = \mathrm{V_A}$ is the correct value of the subtask and it's value can be sent to the output PEs. Also if $\mathrm{V_C}$ is not equal to either $\mathrm{V_B}$ or $\mathrm{V_A}$ then it is known with certainty that at least two of the three PEs ($\mathrm{PE_A}$, $\mathrm{PE_B}$, $\mathrm{PE_C}$) contained Trojans (this occurrence of two Trojans being activated for two variants of the same subtask would be extremely rare). It would be even rarer for all three PEs to contain Trojans. Clearly this process can theoretically be repeated an unlimited number of times to ultimately determine the PEs that contain Trojans and as a result adjust their trust levels. Of course the likelihood of having to repeat this process beyond variants A and B is almost zero. In reality, to have to repeat the process beyond variants A, B, and C will likely never happen. It also should be noted that eventually, frequently using only two variants, the variant execution process must end and a correct value for the subtask with high confidence computed. Finally, it is interesting to note that the rarer the activation of a particular Trojan is, the higher the confidence level becomes, when that Trojan is activated that the Trojan will be detected.

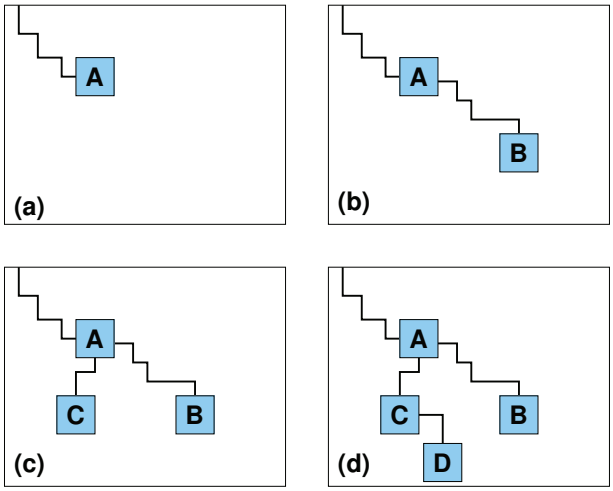

Fig. 2. Example of placement of subtask variants

IV. Dynamic Hardware Trust Determination and High Confidence Subtask Evaluation

We briefly describe the process of dynamic hardware trust determination. This process also determines the correct evaluation of subtasks with high confidence. Initially, each PE in the multicore is given a trust level of 1 indicating that no Trojans have as yet been detected in the PE. Each time a Trojan is discovered in a PE its trust value is halved. This process is best described by examining the life cycle of a subtask. A job will typically consist of number of subtasks that are initially topologically sorted by a pre-scheduler. Each subtask contains several functionally equivalent variants (A, B, C, D,etc). Subtasks are initially read by an input PE (row 0) and the subtask flows through the PE array until it finds a free PE. It begins execution on process variant A (we call this PE, $\mathrm{PE_A}$) see Fig 2a.

The $\mathrm{PE_A}$ then spawns a subtask containing variant B which then flows until it finds a free PE, $\mathrm{PE_B}$, and begins execution, see Fig. 2b. Eventually both variants finish execution, variant B's value follows the path back to $\mathrm{PE_A}$ where the two values,

V_A and V_B say, are compared. If they are equal then with high confidence, neither process A nor process B have executed Trojans and so the correct value is sent from PE_A back up the path to the input PE (row 0) and from there to the adjacent output PE (row $n-1$) for output. If, however, V_A and V_B differ then a subtask containing variant C is spawned and flows until it finds a free PE, PE_C, different from both PE_A and PE_B to execute, see Fig. 2c.

When C completes execution in PE_C the value of variant C, V_C, is compared with V_A and V_B. If V_C equals either V_A or V_B then variant C and the agreeing variant A or B are assumed to have not executed a Trojan. In this case PE_C sends V_C back following the path to PE_A. When it reaches PE_A then if V_C and V_A disagree then the trust of PE_A is multiplied by $1/2$ since it must have executed a Trojan. Otherwise the trust of PE_A is left unchanged and a packet is sent to PE_B to multiply PE_B's trust by $1/2$ since a Trojan has been detected in PE_B. V_C is then sent from PE_A back up the path to the input PE (row 0) and from there to the adjacent output PE (row $n-1$) for output.

In the highly unlikely case when the V_C differs from both V_A and V_B then PE_C spawns subtask variant D which is sent to find another free PE, PE_D, different from PE_A, PE_B and PE_C to execute, see Fig. 2d. When D completes execution the value of D, V_D, is compared with V_A, V_B and V_C. If V_D equals any of V_A, V_B or V_C then variant D and any of the agreeing variants A, B or C are assumed to have not executed a Trojan. V_D is sent back from PE_D to PE_C. When the V_D reaches PE_C then if V_D and V_C agree, then the trust of PE_C is left unchanged otherwise the trust of PE_C is multiplied by $1/2$ since it must have executed a Trojan. V_D is then sent from PE_C to PE_A. When the V_D reaches PE_A then if V_D and V_A disagree then the trust of PE_A is multiplied by $1/2$ since it must have executed a Trojan. Otherwise, if they agree then the trust of PE_A is left unchanged and a packet is sent to PE_B to multiply PE_B's trust by $1/2$ since a Trojan has been detected in PE_B. V_D is sent from PE_A up the path to the input PE (row 0) and from there to the adjacent output PE (row $n-1$) for output.

In the very highly unlikely case when the V_D differs from V_A, V_B and V_C then the process would again repeat in a similar manner to when V_C differed from both V_A and V_B . In this case PE_D would spawn a subtask variant E etc.

Distributed PE Trojan Avoidance Scheduling

Dynamic trust determination allows Trojan avoidance scheduling of subtasks. The benefit of such scheduling is that subtasks can be directed to PEs with high levels of trust (ideally 1) to avoid Trojan activation. This results in decreased job completion time (higher system throughput) since if variants A and B are placed in PEs with trust levels of 1 then rarely will the subtask variant C (or even worse D, E etc) be needed. This avoids significant delays which would otherwise be necessary in sequentially executing variants C, D, etc.

The primary responsibility of the scheduler is to use dynamically learned hardware trust to efficiently schedule subtasks among high trust PEs in order to reduce the likelihood of having to schedule subsequent subtask variants necessary to compute correct subtask values. Hence the scheduler does not distinguish between hardware caused and software caused errors.

The scheduler, governing the subtask and resultant value flow, is totally distributed software throughout the array of PEs (i.e. there is no sharing of memory). PE trust levels as described are a first level indicator of confidence levels in PE trust. A more sophisticated measure would integrate frequency of error detection with frequency of subtasks executed for each PE. The scheduler could assign subtasks to PEs with lower confidence (trust less than 1) when there is a shortage of free PEs – the alternative is to force the subtask to wait for free PEs of trust level 1). Since it is unlikely that any PE that has a Trojan will actually execute that Trojan on any subtask, it follows that the PE will likely correctly compute the value of that subtask. Hence this strategy will likely decrease the throughput time of the job containing this subtask.

V. Results

A 2500 line Java program was written to simulate trust determination and subtask scheduling. Each clock cycle the simulator manages the flow of subtasks and their values throughout the PE array. In all runs, trust determination and any necessary execution of variants were carried out in order to compute with high confidence valid results for all subtasks of the job. The scheduler uses the knowledge of learned PE trust levels to try and avoid sending packets to neighboring PEs with trust levels less than one. As a result, as PE trust levels are learned, subsequent subtasks are less likely to require the execution of additional variants C, D, E, etc. in order to correctly compute the value of the subtasks.

Simulations where carried on one job, J15, containing 15 subtasks. Fig. 3 shows the flowgraph of 15 subtasks including the execution times. PE array sizes of 3×3, 4×4, 8×8 and 16×16 were simulated. Trojans were distributed as follows in the various PE arrays as indicated in Table I.

TABLE I
Location of Trojans in PEs

$n \times n$	Trojan Distribution within the PE Array
3×3	All PEs in row 2 contain Trojans
4×4	All PEs in rows 1 and 3 contain Trojans
8×8	All PEs in rows 1 and 7 contain Trojans
16×16	All PEs in rows 1 and 15 contain Trojans

The frequency of Trojan activation was greatly increased in order to observe the effects on dynamic trust determination and scheduling. Table II illustrates scheduling for job J15. Job J15 was resubmitted to the four PE arrays four times. In all runs the

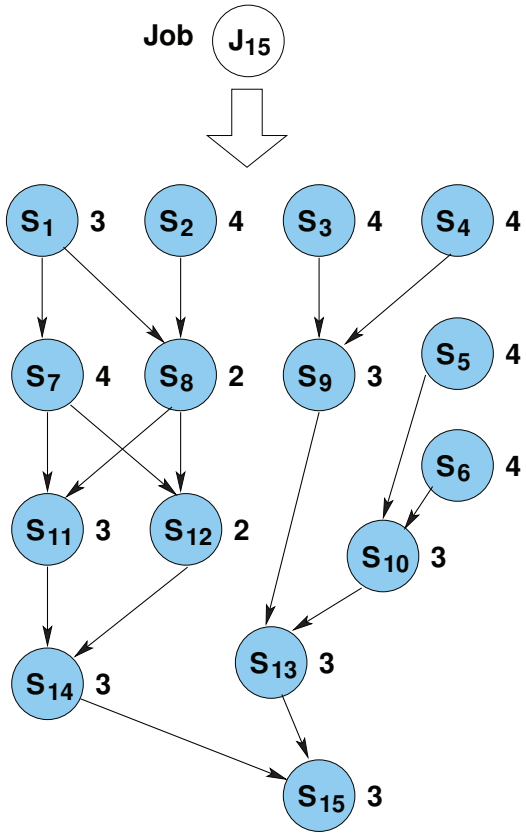

Fig. 3. Flowgraph of the 15 subtasks in Job J_{15}

correct values of all jobs were successfully computed despite frequent Trojan activation. As expected, the job completion times decreased significantly as the scheduler was able to use learned PE trust levels in later runs to avoid assigning subtask variants to PEs with low trust levels (< 1).

TABLE II
JOB J15 SCHEDULING WITH TRUST LEARNING

PE Array size	Job Completion Times			
	Run 1	Run 2	Run 3	Run 4
3×3	61	52	52	52
4×4	67	57	49	49
8×8	69	59	57	50
16×16	87	63	50	50

VI. CONCLUSION

We have explored the effectiveness of dynamic distributed multicore PE trust determination. This is achieved by simultaneously executing a variant of the subtask on another PE to discover Trojans. The subtask scheduling necessary to coordinate the subtask variants produces both new learning with high confidence of PE trusts and high confidence of valid subtask execution results. The scheduler is able to use learned PE trust to more efficiently execute future jobs with increased throughput.

Critical to our approach of dynamic trust determination is the generation and execution of functionally equivalent binary variants of a subtask (i.e. variants that functionally compute the same results but in a different manner). This approach relies on the Trojan property of rare activation. We consider a Trojan model which triggers on sequence of rare events and hence acts like a time-bomb. For example, a sophisticated Trojan can be designed which triggers on a special sequence of machine instructions with specific operand values. Note that the probability of a particular Trojan being triggered by both a task and its variant can be extremely low. This is due to differences in instruction mix, instruction order, operands and memory locations, which is very unlikely to trigger the same set of rare events in case of both task and its variants.

Our experiments have demonstrated the theoretical effectiveness of detecting Trojans given the ability to execute variants of each subtask on an as needed basis. We have shown the effectiveness of PE trust learning with high confidence. We have shown that the PE array can, in a distributed manner, learn trust levels of the cores during deployment. We have shown that a distributed scheduling can use the learned trust to better schedule subsequent subtasks. Preliminary results suggest significant increase in throughput for subsequent jobs.

REFERENCES

[1] S. Adee, "The hunt for the kill switch," *IEEE Spectrum*, pp. 34–39, May 2008.
[2] F. Wolff, C. Papachristou, S. Bhunia, and R. Chakraborty, "Towards trojan-free trusted ics: problem analysis and detection scheme," *Design, Automation, and Test in Europe (DATE'08)*, pp. 1362–1365, Mar. 2008.
[3] R. Rad, J. Plusquellic, and M. Tehranipoor, "Power supply signal calibration techniques for improving detection resolution to hardware trojans," *IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD'08)*, pp. 632–639, Nov. 2008.
[4] M. Banga and M. Hsiao, "A region based approach for the identification of hardware trojans," *IEEE Intl. Workshop on Hardware-Oriented Security and Trust (HOST'08)*, pp. 40–47, Jun. 2008.
[5] J. Li and J. Lach, "At-speed delay characterization for ic authentication and trojan horse detection," *IEEE Intl. Workshop on Hardware-Oriented Security and Trust (HOST'08)*, pp. 8–14, Jun. 2008.
[6] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," *IEEE Intl. Workshop on Hardware-Oriented Security and Trust (HOST'08)*, pp. 51–57, Jun. 2008.
[7] S. Narasimhan, S. Paul, and S. Bhunia, "Collective computing based on swarm intelligence," *Proceedings of the Design Automation Conference (DAC'08)*, pp. 349–350, Jun. 2008.