Probabilistic Certification of Divide & Conquer Algorithms on Global Computing Platforms. Application to Fault-Tolerant Exact Matrix-Vector Product.*

Jean-Louis Roch
Laboratoire d'Informatique de Grenoble (LIG)
51, av. Jean Kuntzmann
38330 Montbonnot Saint Martin, France
Jean-Louis.Roch@imag.fr

Sébastien Varrette
Laboratory of Algorithmics, Cryptology and Security (LACS)
6, rue Richard Coudenhove-Kalergi
L-1511 Luxembourg, Luxembourg
Sebastien.Varrette@imag.fr

ABSTRACT

In [6], a new approach for certifying the correctness of program executions in hostile environments has been proposed. The authors presented probabilistic certification by massive attack detection through two algorithms MCT and $EMCT$. The execution to certify is represented by a macro-dataflow graph which is used to randomly extract some tasks to be re-executed on safe resources in order to determine whether the execution is correct or faulty. Bounds associated with certification have been provided for general graphs and for tasks with out-tree dependencies.

In this paper, we extend those results with a cost analysis of parallel certification based on on-line scheduling by work-stealing. In particular, we focus on Divide & Conquer algorithms that are commonly used in symbolic computations and demonstrate the efficiency of $EMCT$ for the certification of the resulting Fork-Join graph. Finally, we show how to combine $EMCT$ with BCH codes to make a matrix-vector product both tolerant to falsifications and massive attacks.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Reliability

General Terms

Algorithms, Reliability, Security, Verification.

Keywords

Result-Checking, Global Computing, Divide & Conquer Algorithms, Fork-Join Macro-dataflow Graph

*This work is supported by the French government ANR SAFESCALE-BGPR n. ANR-05-SSIA-005.

PASCO'07, July 27–28, 2007, London, Ontario, Canada.

1. INTRODUCTION

Large scale global computing systems like the Grid and Peer-to-peer computing platforms gather thousands of resources for computing parallel applications. Even if a middleware is used to secure the communications and to manage the resources, the computational nodes operate in an unbounded environment and are subject to a wide range of attacks able to alter the computed results. Of course, global computations are expected to tolerate certain low rates of faults [14, 10], yet one should consider the possibility of *massive attacks* resulting in an error rate larger than tolerable by the application. Such massive attacks are especially of concern due to Distributed Denial of Service, virus or Trojan attacks, and more generally orchestrated attacks against widespread vulnerabilities of a specific operating system that may result in the corruption of a large number of resources.

Most research related to protecting large computations from massive attacks has been done in the restrictive context of independent tasks. [6, 7] introduce new probabilistic certification algorithms, *Monte Carlo Test* (MCT) and *Extended Monte Carlo Test* ($EMCT$) that establish whether the computations have been massively attacked. The proposed approach uses a portable representation for the distributed execution E of a parallel program on a fixed input: a bipartite Direct Acyclic Graph $G = (\mathcal{V}, \mathcal{E})$ known as a *macro-dataflow graph*. The first class of vertices is associated to the tasks (in the sequential scheduling sense) whereas the second one represents the parameters of the tasks (either inputs or outputs according to the direction of the edge). The total number of tasks T_j in G is denoted by $|G| = n$.

In the following, we will adopt the notation and assumptions of [6]. In particular, $G^<(T)$ denotes the sub-graph induced by all predecessors of a task $T \in G$ and $G^{\leq}(T) = G^<(T) \cup \{T\}$. The graphs of successors are denoted similarly, e.g. $G^>(T)$ and $G^{\geq}(T)$.

Tasks in G and therefore E are computed on *workers* which compose most of the resources of the Global Computing (GC) platform. Such computational nodes cannot be fully trusted as they execute in a non-secure environment. In the sequel, U (for unreliable resources) denotes the set of workers in the GC platform: each task T in E executes on U with inputs $i(T, E)$ and creates output $o(T, E)$. Conversely, we assume a set R of reliable resources able to host a (possibly distributed) *checkpoint server* (used to store the

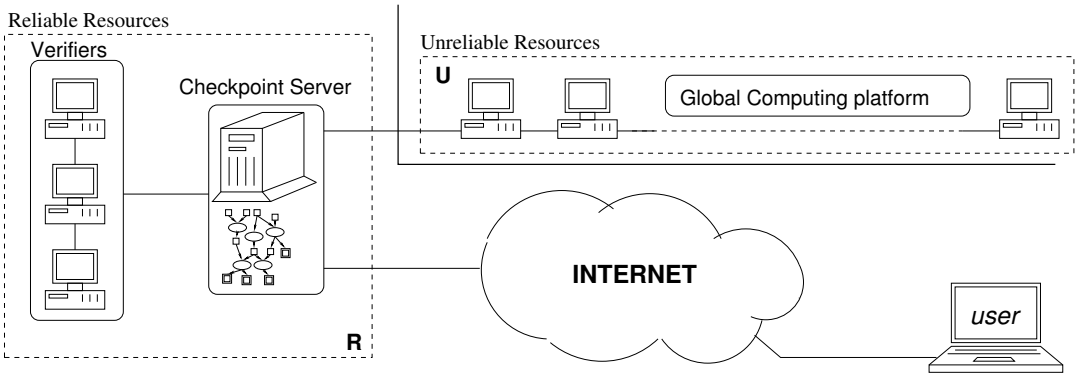

Figure 1: Configuration of a computing platform able to handle both the execution of a parallel program and its certification [15].

graph) and *verifiers* which securely re-execute randomly selected tasks $T \in G$ in a trusted way to determine whether T has been forged. More precisely, given a task T, a verifier re-compute T using $i(T, E)$ and check the outputs with $o(T, E)$.

The certification is conducted on R. As $|R| \ll |U|$, one of the challenge consists in doing the certification with a low overhead. The configuration of the execution platform is illustrated in Fig.1 [15].

As this paper focus on $EMCT$, the main steps of this algorithm are recalled here. More details can be founded in [6].

Algorithm $EMCT$
Input: G, an execution E composed of dependent tasks
Output: the correctness of E (CORRECT/FALSIFIED)
Pick up randomly $T \in G$;
forall $T_j \in G^{\leq}(T)$ / T_j *has not yet been checked* **do**
  $\hat{o}(T_j, E) \leftarrow$ `ReexecuteOnVerifier(`T_j, $i(T_j, E)$`)`;
  if $o(T_j, E) \neq \hat{o}(T_j, E)$ **then return** FALSIFIED;
end
return CORRECT;

Let n_f be the number of forged tasks in G. A *massive attack with ratio* $0 < q < 1$ consists in falsifying the execution of at least $\lceil qn \rceil \leq n_f$ tasks.

Assuming uniform random choice of the checked task $T \in G$, $N_{\epsilon,q} = \lceil \frac{\log \epsilon}{\log(1-q)} \rceil$ independent calls to $EMCT$ ensure a certification by detecting massive attack with ratio q and error probability bounded by ϵ [6].

The cost of the certification performed by $EMCT$ vary depending on the cost of the randomly chosen task T. Then, in all the sequel, we consider the average cost of certification over all (uniform) possible choices of T. The average number of tasks to re-execute on the verifiers in one call to $EMCT$ is $C_G = \frac{1}{n} \sum_{T \in G} |G^{\leq}(T)|$.

In this paper, we analyze the average cost of parallel certification for parallel divide&conquer computations. As presented in section 2, the cost model is based on an on-line scheduling by work-stealing. We show that in the worst case, the certification is too expansive as it conducts to a complete re-execution on the reliable resources. Conversely, section 3 exhibits upper bounds on the certification cost for tree and fork-join computations which are commonly used in algebraic computations. With such graphs, the certification can be achieved with a low overhead. To conclude, this new bound is applied to an algorithm-based fault-tolerant computation of a matrix-vector product.

2. ON-LINE SCHEDULING BY WORK STEALING AND WORST-CASE CERTIFICATION COST

Both the execution and the certification are scheduled on-line by work-stealing following the work-first principle. The principle is simple. Each processor serially executes the tasks it has locally created according to a depth-first order. When a processor becomes idle, it steals the oldest ready task (breadth first order) on a non-idle processor which is randomly chosen in general. This approach is implemented by the parallel programming interfaces Cilk [3, 1] and Kaapi [9]. In particular, Kaapi supports processors with changing speeds and volatility [5]. We restrict to the case where E is fully-strict series-parallel [1] such as the trees and fork-join computations considered in the sequel. Following [1, 12], on each platform U and R we assume a bounded ratio between the fastest and the slowest participating processors. Let Π_U and Π_U^{tot} be respectively the average speed (number of unit operations per second) per processor and the total average speed on U. Similarly, Π_R and Π_R^{tot} are defined for R (e.g., assuming n_R processors in R, $\Pi_R^{tot} = n_R \Pi_R$). Let W_1 be the total work (number of unit operations) of E and W_∞ its depth (maximal number of unit operations on a critical path) on an unbounded number of processors. Then, from Theorem 6 in [1], E is scheduled with high probability on U in time $T_U \leq \frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right)$. In practice, we have $\Pi_U^{tot} \gg \Pi_R^{tot}$ and only the probabilistic certification of E by EMCT is computed on R. Similarly, let W_1^C and W_∞^C be the total work and depth for the certification of E by $EMCT$ on an unbounded number of processors. Then, certification is achieved on R in time $T_R \leq \frac{W_1^C}{\Pi_R^{tot}} + \mathcal{O}\left(\frac{W_\infty^C}{\Pi_R}\right)$. Note that, since EMCT recomputes only predecessors of a randomly chosen task, $W_1^C \leq W_1$ and $W_\infty^C \leq W_\infty$. The next lemma bounds the time for both execution and certification on the computing platform.

LEMMA 1. *With high probability, the average time T_{EC} for both the execution and the certification on the computing platform is bounded by:*

$$T_{EC} \leq \frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right) + \frac{W_1^C}{\Pi_R^{tot}} + \mathcal{O}\left(\frac{W_\infty^C}{\Pi_R}\right).$$

and in the worst case, $W_1^C = \Omega(W_1)$ and $W_\infty^C = \Omega(W_\infty)$.

PROOF. The proof on T_{EC} is direct from bounds on T_U and T_R. In the worst case, the certification performs a complete re-execution on R therefore annihilating the advantage of using the grid U. The lower bound is for instance obtained if G is a chain of n unit tasks. In that case, the average cost of a single call to $EMCT$ is $C_G = \frac{n-1}{2} = \Theta(W_1)$ and, in average, $W_1^C = C_G$. □

If some graphs conduct to the worst-case cost during the certification by $EMCT$ with $T_R = \mathcal{O}(W_1)$, there exist numerous graphs with a much lower overhead. The next section stands that $T_R = \mathcal{O}(W_\infty^2)$ if G is a tree or a Fork-Join graph which are common in exact linear algebra.

3. AVERAGE CERTIFICATION COST ON TREES AND FORK-JOIN GRAPHS

A *Fork-Join* graph G is a directed acyclic graph where the tasks nodes are either of type Fork (a node having at most one predecessor) or Join (a node having at most one successor); G has exactly one source node and one sink node. The restricted family of Fork-Join graphs is described using the following grammar. G is either:

1. a graph with only one vertex (both source and sink) which is considered as a Fork task;

2. the parallel composition of $k > 0$ Fork-Join graphs $G_1, \ldots, G_k$ with addition of two vertices F (a task of type Fork) and J (a task of type Join): F is the source node and has k direct successors which are the sources of $G_1, \ldots, G_k$; J is the sink node and has k direct predecessors that are the sinks of $G_1, \ldots, G_k$. Such a graph is clearly of height $h = 2d - 1$; it can be seen as an out-tree G_F of height d (composed by Fork tasks) followed by the symmetric in-tree G_J of height $d - 1$ (composed by Join tasks), see Fig.2.

Note that the above definition forbids serial composition of fork-join graphs and thus is restrictive with respect to series-parallel graphs. However, subgraphs such as out-tree (respectively in-tree) can be encompassed by adding artificial Join tasks (respectively Fork tasks) with null work (`nop` operation); those artificial tasks are only considered for the random choices made by $EMCT$ but not for the effective execution, either on U or R.

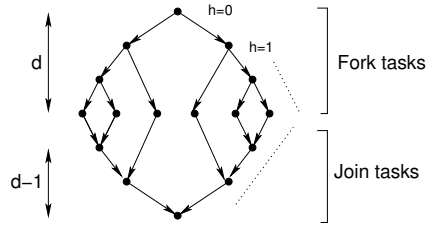

Figure 2: A Binary Fork-Join graph

The objective of this section is to prove following theorem:

THEOREM 1. *Let G be a graph of tasks with height h, total work W_1 and work depth W_∞. If G is either a tree or a Fork-Join graph, the average number of tasks to re-execute for a certification by $EMCT$ is $C_G = \mathcal{O}(h)$. In addition,*

$$T_{EC} \leq \frac{W_1}{\Pi_U^{tot}} + \mathcal{O}\left(\frac{W_\infty}{\Pi_U}\right) + \mathcal{O}\left(\frac{hN_{\epsilon,q}W_\infty}{\Pi_R^{tot}}\right) + \mathcal{O}\left(\frac{W_\infty}{\Pi_R}\right).$$

Basically, this theorem states that the certification of execution represented by trees or Fork-Join graphs of height h could be achieved with a very low overhead as $W_1^C = \mathcal{O}(hW_\infty) = \mathcal{O}(W_\infty^2)$. This cost is far from the worst-case certification cost exhibited in the Lemma 1 for the class of programs generally considered in parallel computing where $W_\infty \ll W_1$.

3.1 Certification cost with in/out-trees

To prove the Theorem 1, we first need to demonstrate some results relative to in-trees and out-trees. This is the purpose of the following lemma.

LEMMA 2. *Let G be an in-tree or an out-tree of height h. Then $C_G \leq h + 1$ and $W_1^C \leq (h + 1)N_{\epsilon,q}W_\infty$.*

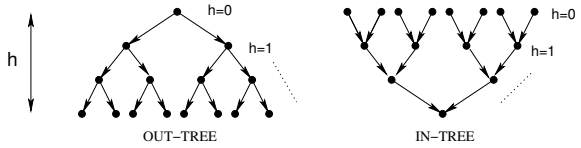

Figure 3: Complete binary trees

PROOF. We distinguish both configuration:

- For out-trees, any task T has at most $h+1$ predecessors (including itself) in G. Therefore $C_G \leq h + 1$.

- As regards in-trees, let proceed by recurrence on the height h of a tree with n nodes. The proposition is verified if $h = 0$ (then $n = 1$).

  For $h > 0$, the sink node is the root of k trees $(A_i)_{1 \leq i \leq k}$; A_i has n_i nodes and height at most $h - 1$. By recurrence, the average number of tasks to re-execute on the verifiers in one call to $EMCT$ applied to A_i satisfies $C_{A_i} \leq h$. Then

  $$n.C_G = n + \sum_{i=1}^{k} n_i C_{A_i} \leq n + (n-1)h < n(h+1)$$

  which leads to $C_G \leq h + 1$.

The bound on W_1^C follows from the fact that each task in G costs at most W_∞ and $N_{\epsilon,q}$ independent calls to $EMCT$ are sufficient to achieve a probabilistic certification with error probability bounded by ϵ. □

3.2 Certification cost with Fork-Join graphs

We now prove a similar lemma relative to Fork-Join graphs generated by the grammar mentioned at the beginning of this section.

LEMMA 3. *Let G be an Fork-Join graph of height h. Then $C_G \leq h + 3$ and $W_1^C \leq (h + 3)N_{\epsilon,q}W_\infty$.*

PROOF. The lemma is clearly correct if $h = 0$. We now assume $h > 0$; consider the out-tree G_F of height $d \geq 1$ composed by n_F Fork tasks followed by the symmetric in-tree G_J of height $d-1$ composed by n_J Join tasks The height h of G is $h = 2d - 1$ and $n = n_F + n_J$.

Let C_{G_F} (resp. C_{G_J}) be the average number of predecessors in G_F (resp. G_J).

Let $T \in G$: if T is a Fork Task, then $G^{\leq}(T) = {G_F}^{\leq}(T)$. Otherwise, $G^{\leq}(T)$ is composed of:

1. its $|{G_J}^{\leq}(T)|$ predecessors in G_J;
2. its corresponding predecessors in G_F which form an out-tree ${G_F}^{\geq}(T')$ with root task T' symmetric of T;
3. at most $d + 1$ predecessors of T' in G_F.

Then, the total number of predecessors in G is bounded by:

$$nC_G \leq n_F C_{G_F} + n_J(2C_{G_J} + d + 1).$$

However, $C_{G_F} < d + 1$ and $C_{G_J} < d$ from lemma 2. Then

$$nC_G \leq n_F(d+1)+n_J(3d+1) \leq (d+1)(2n_J+n) \leq 2(d+1)n$$

as $n_J \leq \frac{n}{2}$. Finally, $C_G \leq h + 3$.

As for trees, the bound on W_1^C follows the fact that each task in G performs at most W_∞ operations and that $N_{\epsilon,q}$ independent calls to $EMCT$ are sufficient to achieve a probabilistic certification with error probability bounded by ϵ. □

3.3 Proof of Theorem 1

We prove this theorem by a direct application of Lemma 1 using upper bounds on W_1^C provided by Lemmas 2 and 3.

4. APPLICATION TO AN EXACT MATRIX-VECTOR PRODUCT

Theorem 1 shows that the certification of Divide & Conquer algorithms whose execution is represented by a Fork-Join graph can be achieved with a very low average overhead. Assuming that an algebraic computation is defined by such an algorithm and corrects up to $\lceil qn \rceil$ falsifications, the non-detection of a massive attack with ratio q could authorize the correction at low cost. Low rate fault-tolerant algorithms, referred as Algorithm-Based fault-tolerant (ABFT) have been proposed for exact computations in groups and rings [13, 8, 4], for linear algebra based on parity checkpointing [10, 2] but not for tolerating forgery of results. We extend those results to efficiently support tolerance against malicious faults by combining BCH codes with massive attack detection.

For instance, let us consider an iteration with a matrix-vector product over a finite field $\mathbb{F}_m$, the same matrix being used for a huge number of iterations. In the sequel, for the sake of simplicity, we consider that arithmetic operations on $\mathbb{F}_m$ are computed in $O(1)$ time.

More precisely, let A denote a square matrix in $\mathcal{M}_k(\mathbb{F}_m)$. We assume that A is applied to a large number of vectors $x = (x_0, \ldots, x_{k-1}) \in \mathbb{F}_m^k$. Computing $y = x^{tr}A$ can be made tolerant to falsification in the following way, based on the following pre-computation on the reliable resources R:

1. first choose an attack ratio $0 < q < \frac{1}{2}$ and compute $n = \frac{k}{1-2q}$. For instance, $q = \frac{1}{6}$ implies $n = 1,5k$.
2. then construct a $[n, k]$ BCH error correcting code over $\mathbb{F}_m$ [11].This code has distance $d \geq n-k+1 \geq 2qn+1$ and corrector rate $t \geq nq$. Let G denote the generator matrix of the code corresponding to the generator polynomial $g(X)$. **On reliable resources R:** Precompute $B = A.G$.

As the code is cyclic, this can be computed by k parallel polynomial products modulo $X^n - 1$ over $\mathbb{F}_m$, therefore with $\mathcal{O}(k.n \log k) = \mathcal{O}(\frac{k^2 \log k}{1-2q})$ operations. Since the matrix A is used for a large number of iterations, we do not consider the overhead of this pre-computation[1]

Now, the algorithm for computing and certifying the result $y = x^{tr}A$ is:

- **On unreliable resources U:** compute $z = x^{tr}B = (z_0, \ldots, z_{n-1})$ with $W_1 = \mathcal{O}(kn) = \mathcal{O}(\frac{k^2}{1-2q})$ and $W_\infty = \mathcal{O}(\log k)$. Note that if no falsification occurs, $z = yG$. The execution corresponds to a Fork-Join graph.
- **On reliable resources R:** Use $EMCT$ to detect if a massive attack with ratio q has been performed. with $W_1^C = \mathcal{O}(hN_{\epsilon,q}W_\infty) = \mathcal{O}(N_{\epsilon,q} \log^2 k)$ operations and $W_\infty^C = \mathcal{O}(\log k)$. If yes, restart computation of z. Otherwise, correct z in z' by BCH-decoding and compute $y = z'/g(X)$ with $W_1' = \mathcal{O}(\frac{k \log k}{1-2q})$ and $W_\infty' = \mathcal{O}\left(\log^2\left(\frac{k}{1-2q}\right)\right)$.

Finally, without considering the pre-computation of B, each matrix-vector product is computed in time

$$T_U = \mathcal{O}\left(\frac{k^2}{(1-2q)\Pi_U^{tot}} + \frac{\log k}{\Pi_U}\right)$$

on U and certified on R in

$$T_R = \mathcal{O}\left(\frac{k \log k}{(1-2q)\Pi_R^{tot}} + \frac{\log^2 k}{\Pi_R}\right)$$

5. CONCLUSION

In this paper, we extend results on $EMCT$ with a cost analysis of a parallel certification based on on-line scheduling by work-stealing. We first proved that a probabilistic certification by massive attack detection using $EMCT$ could be achieved efficiently with a very low overhead on trees and Fork-Join graphs. Finally, we combine this probabilistic certification with error-correcting codes to make a matrix-vector product over a finite field $\mathbb{F}_m$ both tolerant to falsifications and massive attacks. This last technique uses BCH-coding and could be generalized to other exact linear algebra computations; but its generalization to non-linear computations (*e.g.*, arithmetic circuits of higher degree) is open.

[1]This hypothesis is reasonable if the number of matrix-vector products to perform is very large w.r.t. $\log k$. Besides, the pre-computation cost is almost linear in the size k^2 of A which compares favourably to the overhead of distribution of A on the global computing platform U.

6. REFERENCES

[1] M. A. Bender and M. O. Rabin. Online scheduling of parallel programs on heterogeneous systems with applications to cilk. *Theory Comput. Syst.*, 35(3):289–304, 2002.

[2] Z. Chen and J. J. Dongarra. Algorithm-Based Checkpoint-Free Fault Tolerance for Parallel Matrix Computations on Volatile Resources. Rhodes Island, Greece, april 2006.

[3] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the cilk-5 multithreaded language. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 212–223, 1998.

[4] N. V. R. R. George A. Reis, Jonathan Chang and D. I. August. SWIFT: Software Implemented Fault Tolerance. In *Proceedings of the Third International Symposium on Code Generation and Optimization (CGO)*, March 2005.

[5] S. Jafar, T. Gautier, A. W. Krings, and J.-L. Roch. A checkpoint/recovery model for heterogeneous dataflow computations using work-stealing. In *EUROPAR'2005*, August 2005.

[6] A. Krings, J.-L. Roch, S. Jafar, and S. Varrette. A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments. In *EGC2005*, LNCS 3470. Springer Verlag, February 14–16 2005.

[7] A. W. Krings, J.-L. Roch, and S. Jafar. Certification of large distributed computations with task dependencies in hostile environments. In *EIT 2005*, May 2005.

[8] A. Li and B. Hong. A low-cost correction algorithm for transient data errors. In *Ubiquity*, volume 7, May 2006.

[9] MOAIS Team. KAAPI. `http://kaapi.gforge.inria.fr/`, 2005.

[10] J. S. Plank, Y. Kim, and J. Dongarra. Fault tolerant matrix operations for networks of workstations using diskless checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):125–138, June 1997.

[11] V. Pless. *Introduction To The Theory of Error Correcting Codes*. John Wiley Sons, 1990.

[12] J.-L. Roch, D. Traore, and J. Bernard. On-line adaptive parallel prefix computation. In *LNCS 4128, EUROPAR'2006*, pages 843–850, August 2006.

[13] G. K. Saha. Software based fault tolerance: a survey. *Ubiquity*, 7(25):1–1, 2006.

[14] L. F. G. Sarmenta. *Volunteer Computing*. PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, March 2001.

[15] S. Varrette, J.-L. Roch, J. Montagnat, L. Seitz, J.-M. Pierson, and F. Leprvost. Safe Distributed Architecture for Image-based Computer Assisted Diagnosis. In *IEEE 1st International Workshop on Health Pervasive Systems (HPS'06)*, Lyon, France, june 2006.