From Quasipolynomial to Data-Parallel Algorithms for Verification Games Played on Graphs

Vanessa Flügel[1,2], Marcin Jurdziński[3], and Guillermo A. Pérez[1]

[1] University of Antwerp, Belgium
{vanessa.flugel,guillermo.perez}@uantwerpen.be
[2] Université de Sherbrooke, Canada
[3] University of Warwick, UK
jurdzinski@warwick.ac.uk

Abstract. We present a practical acceleration of recent quasipolynomial-time progress measure algorithms for parity games. For this, we combine a novel Strahler-universal tree encoding with data-parallel (SIMD) computation. Our key contribution is showing that common instances admit very small structural parameters, enabling compact representations that fit into SIMD registers. This insight allows us to redesign the core lifting (i.e. successor computation) step into a parallel form, yielding significant empirical speedups over existing implementations. This work demonstrates that bridging algorithmic structure (Strahler measures) with hardware-aware optimization (SIMD) can materially improve parity game solving.

1 Introduction

Parity games [8] are played by two players on a directed graph whose vertices are partitioned between them. A token is moved indefinitely along edges, and the winner is determined by the parity of the highest priority visited infinitely often. These games are central in formal verification: model checking for the modal μ-calculus reduces to solving parity games [3], and synthesis of reactive systems routinely produces such games as a final step [4]. Despite extensive study, the problem is not known to be in **P**, motivating continued work on it.

Among the most successful approaches are progress measure algorithms [5], recently improved to quasipolynomial time via succinct tree encodings [6,1]. These methods assign to each vertex a label from an ordered tree and repeatedly *lift* labels: replacing a vertex's value by the minimal value consistent with its successors under the parity condition.

Contribution We focus on the Strahler-universal tree framework of [1]. Our key observation is empirical: for standard benchmark suites, the parameters governing these trees are very small, i.e. labels are short sequences of short bitstrings.

This structural sparsity has an important consequence: entire labels fit into SIMD registers. We exploit this by reformulating the lifting step as a vectorized procedure operating on packed bit representations.

Our implementation in the Oink solver [2] shows that: (i) a scalar Strahler-based solver is slower than existing implementations, but (ii) the SIMD version significantly outperforms them, while (iii) similar SIMD treatment does not benefit simpler algorithms such as BSSPM.

2 Preliminaries

Progress measure algorithms map each vertex to a label drawn from a finite ordered domain. In succinct encodings, these labels correspond to leaves of a tree and are represented as sequences of bitstrings ordered lexicographically.

Formally, we consider binary strings with order

$$0\beta < \varepsilon < 1\beta \text{ and } b\beta < b\beta' \iff \beta < \beta'.$$

A label is a tuple $\langle \beta_0, \ldots, \beta_{h-1} \rangle$ encoding a path in a tree of height h.

Strahler-Universal Trees refine this representation by constraining branching structure via the Strahler number k, which measures how often branching of equal depth occurs. Labels satisfy: (i) exactly $k-1$ non-empty components, (ii) at most t non-leading bits in total. These constraints ensure that labels are both succinct and structured: they can be seen as packed bitstrings with bounded width and sparsity. This is the key property enabling vectorization.

Lifting and Level-p Successors Lifting updates a vertex label based on its successors. For a vertex of priority p, the new label is obtained by taking the minimum (or maximum, depending on the player) among the level-p successors of its successors' labels.

Operationally, given a label ℓ, its level-p successor is the smallest label strictly larger than ℓ when comparing only the first $p+1$ components. This corresponds to moving to the next leaf in the tree consistent with the parity constraint.

3 Computing Level-p Successors

The computation of level-p successors is the core operation. It consists of two steps: (1) locating the lowest position where the label can be increased without violating structural constraints, and (2) reconstructing a minimal valid suffix.

The first step is governed by combinatorial conditions on bit usage and structure (e.g. limits on non-leading bits and non-empty components). The second restores invariants by padding with appropriate bitstrings.

Scalar Implementation In a scalar implementation, labels are stored as sequences of bits annotated with their component indices. The algorithm scans these sequences backwards to find the first position where an increment is possible, then rewrites the suffix.

While efficient, this representation is inherently sequential: conditions are checked one position at a time, and control flow depends on early exits.

SIMD Reformulation Our key idea is to exploit the bounded size of labels. When k and t are small, all components of a label can be packed into fixed-width vectors. We store: (i) bit patterns in SIMD registers, (ii) masks indicating active bits, and (iii) auxiliary vectors tracking counts such as non-leading bits. This enables the evaluation of all candidate positions simultaneously. The structural conditions for incrementability become Boolean expressions over vectors, producing masks that identify all valid positions in parallel.

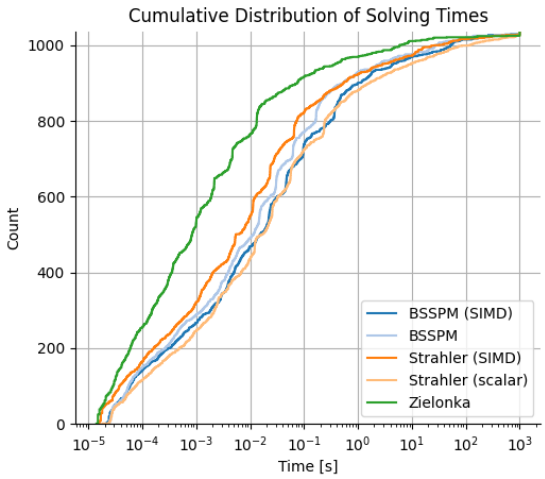
The algorithm then: computes a mask of valid increment points, selects the lowest such position using bit operations, updates the corresponding component, and restores invariants via masked vector assignments.

Both the search for a successor and its construction are expressed as vectorized bitwise and comparison operations, eliminating most branching.

4 Promising Experimental Results

We implemented both scalar and SIMD variants in the Oink solver, using BSSPM as a baseline. The solver iteratively increases parameters (k, t).

On the Keiren benchmark suite [7], we observe that all instances are solved with $k, t \leq 6$. This confirms that labels fit comfortably within SIMD registers.



The SIMD implementation significantly outperforms both the scalar Strahler solver and some existing quasipolynomial solvers. Zielonka's recursive algorithm remains the fastest overall, but the gap is reduced.

References

1. Daviaud, L., Jurdzinski, M., Thejaswini, K.S.: The strahler number of a parity game. In: ICALP. pp. 123:1–123:19. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
2. van Dijk, T.: Oink: An implementation and evaluation of modern parity game solvers. In: TACAS (1). pp. 291–308. Lecture Notes in Computer Science, Springer (2018)
3. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: CAV. pp. 385–396. Lecture Notes in Computer Science, Springer (1993)
4. Jacobs, S., Pérez, G.A., Abraham, R., Bruyère, V., Cadilhac, M., Colange, M., Delfosse, C., van Dijk, T., Duret-Lutz, A., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, K.J., Michaud, T., Pommellet, A., Renkin, F., Schlehuber-Caissier, P., Sakr, M., Sickert, S., Staquet, G., Tamines, C., Tentrup, L., Walker, A.: The reactive synthesis competition (SYNTCOMP): 2018-2021. Int. J. Softw. Tools Technol. Transf. **26**(5), 551–567 (2024)
5. Jurdzinski, M.: Small progress measures for solving parity games. In: STACS. pp. 290–301. Lecture Notes in Computer Science, Springer (2000)
6. Jurdzinski, M., Lazic, R.: Succinct progress measures for solving parity games. In: LICS. pp. 1–9. IEEE Computer Society (2017)
7. Keiren, J.J.A.: Benchmarks for parity games. In: FSEN. pp. 127–142. Lecture Notes in Computer Science, Springer (2015)
8. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**(1-2), 135–183 (1998)