Alcom-FT Technical Report Series
ALCOMFT-TR-02-45

Any Load-Balancing Regimen for Evolving Tree Computations on Circulant Graphs is Asymptotically Optimal*
(Preliminary Version)

Rolf Wanka
Dept. of Mathematics & Computer Science and Heinz Nixdorf Institute
Paderborn University, 33095 Paderborn, Germany. wanka@upb.de

Abstract

We analyze evolving tree computations on circulant (rings with chords) and related graphs. In an evolving α-ary tree computation, a complete tree grows level by level, i. e., every leaf generates α new nodes that become the new leaves. The load balancing task is to spread the new nodes on a network of processors in the moment they were created in such a way that the accumulated number of nodes per processor, i. e., its load, is as close as possible to the average number of nodes per processor. As a single processor can hold many leaves, it has to handle many new tokens. Gao/Rosenberg [2] introduced evolving computations and investigated the growth of complete binary trees on rings of processors. They showed that the so-called KS-regimen where one new node generated by a leaf remains at the processor where it has been created and the other one is transmitted to the right neighboring processor behaves optimally in the course of long computations. In this paper, we generalize evolving computations to arbitrary trees and we generalize the regimen notion. We show that *any* regimen behaves optimally. For this purpose, we model the actual load distribution, the generation process, and the distribution regimen by formal infinite polynomials. Then we show that evaluating these polynomials at certain points leads to the analysis of these regimens on circulant and related graphs. It is shown that *any* regimen leads to a close to optimal load distribution.

1 Introduction

Background. In the standard abstract formulation of load balancing in a distributed network, processors are modeled as the vertices of a graph and links between them as edges. Each processor initially has a collection of unit-size jobs which we call *tokens*. In a dynamic setting, some of these tokens generate new tokens. so we distinguish between generating and non-generating tokens. The object is to balance the number of tokens by transmitting the new tokens along edges according to some local scheme. This problem has obvious applications in job scheduling and other coordination tasks in parallel and distributed systems. It also arises in the context of finite element computations, and in simulations of physical phenomena.

In order to avoid expensive reordering of all tokens or special computations of destinations for every new token individually, it is desirable to have a simple general strategy that governs the load transmission procedure. For example, if every generating token which resides in processor i creates four new tokens, a very simple strategy would be to send one of them to processor $i-1$, to hold two,

*Partially supported by DFG SFB 376 “Massively Parallel Computation” and by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

and to send one to processor $i+2$, where the processor numbers are considered modulo the size n of the network and having edges to realize these transmissions.

The topic of this paper is the following. We assume the token generation process to grow a tree where, in one round, only the leaf nodes generate new tokens. Every leaf generates the same number of tokens. After the new tokens have been created in the processors of the network it has to be decided which tokens are held and which tokens are sent to which neighboring processors.

Previous work. Gao/Rosenberg introduced in [2] the notion of *(regularly) evolving (binary) tree computations* on a ring of n processors. In their setting, a complete binary tree is grown. Initially, only one processor holds a generating token. Every generating token creates in a single round two new generating tokens, but will not create further tokens for the rest of the computation. The new tokens have to be spread among the processors according to a *regimen*. Gao/Rosenberg define the KS-regimen ("Keep-left-Send-right") that works on a ring of processors as follows: One new token stays at the processor where it has been generated, one token is sent to the, say, right neighbor of the processor where it has been generated. So, during the computation, the total load of generating and non-generating tokens accumulates in every processor. They show that this policy is asymptotically optimal, i. e., that the tokens are evenly distributed among the processors. More specifically, let the token generation process be run t rounds on a ring of n processors (so the total number of tokens is $M_t = 2^{t+1}-1$), and let $l_i^{(t)}$ the number of tokens in processor i. Then $|l_i^{(t)} - \frac{M_t}{n}| \leq (2\cos(\pi/n))^{t+1} = o(2^{t+1})$, and thus, $l_i^{(t)} = (1 \pm o(1)) \cdot \frac{M_t}{n}$.

In another seminal paper [1], Bhatt/Cai investigate evolving binary tree computations on d-dimensional hypercubes. The tree is allowed to grow arbitrarily, so not only leaves can generate a token in one round. New tokens are sent along a random path of length $O(\log d)$ to their destination. With M denoting the size of the tree, their algorithm ensures that the maximum load per processor is $O(1+M/2^d)$, with high probability. To obtain this result, random walks on the hypercube are analyzed accurately.

New result. In this paper, we generalize the regularly evolving tree computations approach to arbitrary α-ary trees that grow level by level. Here, in one round every generating (leaf) token creates $\alpha = \alpha_1 + \cdots + \alpha_k$ new tokens, $k \geq 1$ and $\alpha_i \in \mathbb{N}$ fixed. Let the mentioned tokens be created in processor i. A global regimen states that α_j of these tokens are transmitted to processor $i+\delta_j$, $\delta_j \in \mathbb{Z}$ fixed, where the processor numbers are considered modulo the size n of the network and having edges to realize these transmissions. For the Gao/Rosenberg KS-regimen, $(\alpha_1, \alpha_2) = (1,1)$ and $(\delta_1, \delta_2) = (0,1)$. We show that *any* global regimen on circulant graphs, i. e., rings of length n that have chords that correspond to the δ_j sequence given above, is asymptotically optimal. That means that when we have after t rounds a tree of size $M_t = (\alpha^{t+1}-1)/(\alpha-1)$, the load in processor i is $l_i^{(t)} = (1 \pm o(1)) \cdot \frac{M_t}{n}$.

In order to prove this property, we describe global load-balancing regimens by generating polynomials which simplifies the analysis of the Gao/Rosenberg KS-regimen, or any regimen, considerably by applying algebraic tools.

Organization of paper. In Section 2, we give a more formal description of evolving computations and regimens and state the result of this paper in this framework. In Section 3, we describe the token generation process by generating polynomials on an infinite linear array with chords. Then, in Section 4, we show that winding-up the infinite array on an n-cycle with chords can be modeled and analyzed by evaluating the generating polynomials for powers of nth primitive roots of unity. Finally, in Section 5 we demonstrate how this approach can be used for analyzing evolving tree computations on multi-dimensional tori with regular chords.

2 Definitions and Result

Let $\alpha \in \mathbb{N}$. An *evolving α-ary tree computation* is a complete α-ary tree that *grows* in rounds. Initially, there is only the root. In one *round*, every leaf of the tree generates α new leaves. So after t rounds, there are α^t leaves, and the size of the tree is $\frac{\alpha^{t+1}-1}{\alpha-1}$. As synonym for the term "node" we use the term "token." The leaves are called *generating* tokens, the interior nodes are *non-generating* tokens.

Let $\alpha = \alpha_1 + \cdots + \alpha_k$, $k \geq 1$, $\alpha_j \in \mathbb{N}$ fixed. The α-ary tree computation grows on a host network of processors that are numbered from 0 through $n-1$. For simplicity, think of the network as an n-cycle, i. e., processor j is connected to processor $(j+1) \bmod n$. The root of the tree computation is in some processor j. Let $(\delta_1, \ldots, \delta_k)$, $-n < \delta_j < n$ be fixed. The global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$ *regimen* governs one round of the tree computation as follows: For every generating token stored in processor i, α_j new tokens are sent to processor $(i + \delta_j) \bmod n$, for every $j \in \{1, \ldots, k\}$.

If for every processor i the edges $\{i, (i+\delta_j) \bmod n\}$ are added to the n-cycle, we call the resulting graph a *circulant graph*. The graph is a *full* circulant graph, if there is only one connected component induced by the edges defined by the δ-sequence. E. g., when $n = 6$ and $(\delta_1, \delta_2) = (0, 2)$, there are two connected components, whereas, when $n = 5$ there is only one. An example of an evolving 4ary tree computation with $n = 6$ and the $(1,2,1;0,1,2)$-regimen is given in Figure 1.

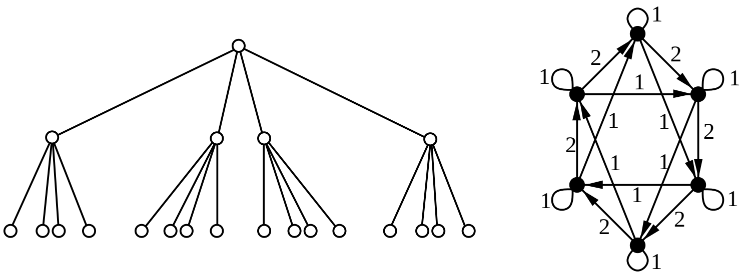

Figure 1: An evolving 4ary tree computation after $t = 2$ rounds and the circulant host network with $n = 6$ corresponding to the $(1,2,1;0,1,2)$-regimen and to $p(x) = 1 + 2x + x^2$.

Note that the dilation of the tree edges in the host network is 1.

In the rest of this paper, let $a_i^{(t)}$ denote the number of generating tokens stored in processor i after round t. Similarly, let $l_i^{(t)}$ denote the number of tokens stored in processor i.

We prove the following theorem.

Theorem 1 *For any global $(\alpha_1, \ldots, \alpha_k; \delta_1, \ldots, \delta_k)$ regimen on a full circulant graph of size n described by the δ-sequence holds, as t advances:*

$$l_i^{(t)} = \big(1 \pm o(1)\big) \cdot \frac{1}{n} \cdot \frac{\alpha^{t+1}-1}{\alpha-1}$$

That means that with increasing number of rounds the load in every processor comes closer and closer to the average load, independent of the used regimen. In a certain way this resembles the properties of random walk on a cycle where after some time the probability distribution comes close to the uniform distribution.

3 The Infinite Setting and the Generating Polynomials

Before we turn to the cycle, we first show how an evolving tree computation and a global $(\alpha_1,\ldots,\alpha_k;\ \delta_1,\ldots,\delta_k)$-regimen can be modeled on an *infinite* linear array as the tree's host network. In this array, the processors are numbered from $-\infty$ through ∞. As in circulant graphs, we assume that each processor i is connected to all processors $i+\delta_j$. For processor i, let $a_i^{(t)}$ denote the number the generating tokens stored in i after round t, and let $l_i^{(t)}$ denote the total number of tokens in processor i.

As this infinite graph is invariant under shifts right and left, we assume, w. l. o. g., that the root is stored in processor 0. Hence, $a_0^{(0)} = l_0^{(0)} = 1$.

We describe the distributions of the generating tokens and of all tokens among the processors after round t by two generating functions,

$$a^{(t)}(x) = \sum_{i=-\infty}^{\infty} a_i^{(t)} \cdot x^i \qquad \text{and} \qquad l^{(t)}(x) = \sum_{i=-\infty}^{\infty} l_i^{(t)} \cdot x^i,$$

resp., and let, for a global $(\alpha_1,\ldots,\alpha_k;\delta_1,\ldots,\delta_k)$-regimen,

$$p(x) = \sum_{j=1}^{k} \alpha_j \cdot x^{\delta_j} \quad .$$

E. g., for the $(1,1;0,1)$-regimen (i. e., the KS-regimen), $p(x) = 1+x$, and for the $(1,2,1;-1,0,1)$-regimen, $p(x) = x^{-1}+2+x$.

$p(x)$ describes the regimen completely, and calculations with it can simulate the evolving tree computation because there is the following connection between the regimen, the tree computation and the polynomials.

Lemma 1 *Let the global* $(\alpha_1,\ldots,\alpha_k;\delta_1,\ldots,\delta_k)$*-regimen in the infinite network setting be given.*

(a) $a^{(t)}(x) = p(x)^t$

(b) $l^{(t)}(x) = \sum_{\tau=0}^{t} p(x)^t = \dfrac{p(x)^{t+1}-1}{p(x)-1}$

Proof. (a) $a_i^{(t)} \cdot x^i$ is the monomial in $a^{(t)}(x)$ that describes the number $a_i^{(t)}$ of generating tokens in processor i. These tokens create, for $j \in \{1,\ldots,k\}$, $a_i^{(t)} \cdot \alpha_j$ tokens that have to be transmitted to processor $i+\delta_j$, i. e.,

$$\sum_{j=1}^{k} a_i^{(t)} \cdot \alpha_j \cdot x^{i+\delta_j} = a_i^{(t)} \cdot x^i \cdot \sum_{j=1} \alpha_j \cdot x^{\delta_j} = a_i^{(t)} \cdot x^i \cdot p(x)$$

must, as the contribution of processor i, be added to $a^{(t+1)}(x)$. Thus $a^{(t+1)}(x) = a^{(t)}(x) \cdot p(x)$, which completes the proof of (a).

(b) follows from (a) and the fact that $l^{(t)}(x) = \sum_{\tau=0}^{t} a^{(\tau)}(x)$. $\square$

Hence, the load distribution of the KS-regimen with polynomial $p(x) = 1+x$ is given by $l^{(t)}(x) = \dfrac{(1+x)^{t+1}-1}{x}$. Due to symmetry, the "reversed" KS-regimen $p(x) = x^{-1}+1$ results in the same sequence of loads except that the number of tokens that has been in processor i, is in this case in processor $-i$. So, also $l^{(t)} = x\cdot(1+1/x)^{t+1} - x$ returns the same load per processor as the KS-regimen. In general, we can multiply $p(x)$ by x^j, $j \in \mathbb{Z}$, without changing the load sequence. A further consequence is that with $p(x) = 1+x$ and $\hat{p}(x) = p(x)^2/x = x^{-1}+2+x$, the two regimens result, after $2t$ and t rounds, resp., in the same load sequence.

Of course, in the infinite setting the regimen cannot result in a good distribution of the total load that is obviously $l^{(t)}(1)$. E. g., with $p(x) = 1+x$, in $l^{(t)}(x)$ the largest coefficient is $\binom{t+1}{\lfloor t/2 \rfloor}$, and the smallest non-zero coefficient is 1.

In the next section, we will show that $l^{(t)}(x)$ also can be used to analyze the finite setting.

4 The Finite Setting: Winding Up the Infinite Array

In the following, we show that winding up the infinite array and its load on the n-cycle leads to an exact description of the evolving tree computation on the cycle (and its chords). As we have seen above, the load difference can be very large in the infinite setting. However, the winding-up flattens this difference dramatically.

Let $p(x)$ be a regimen, and let $\hat{l}^{(t)}(x) = \sum_{i=-\infty}^{\infty} \hat{l}_i^{(t)} x^k$ the resulting load polynomial in the infinite setting. Consider the same regimen on the n-cycle (with appropriate chords). Then the load $l_i^{(t)}$ of processor i of the cycle is related to the infinite setting as follows.

Lemma 2 *For the load $l_i^{(t)}$ of processor i on the n-cycle,* $l_i^{(t)} = \sum_{k=-\infty}^{\infty} \hat{l}_{i+kn}^{(t)}$

This lemma follows from the fact that generating tokens are not influenced by other generating tokens. So they can be bijectively identified in the cycle and the infinite array.

In the following, we shall see how we can calculate $l_i^{(t)}$ by evaluating $\hat{l}^{(t)}(x)$ for appropriate complex numbers x. Let $\omega_n = e^{2i\pi/n} = \cos(2\pi/n) + i\sin(2\pi/n)$ be an nth primitive root of unity. ω_n has the the following important properties:

- $\sum_{j=0}^{n-1} \omega_n^j = 0$ and
- $\omega_n^i = \omega_n^{i+k\cdot n}$ for all k.

Lemma 3 *The load $l_i^{(t)}$ of processor i after t rounds is*

$$l_i^{(t)} = \frac{1}{n}\cdot\sum_{j=0}^{n-1} \omega_n^{-ji}\cdot\frac{p(\omega_n^j)^{t+1}-1}{p(\omega_n^j)-1}$$

Proof.

$$\frac{1}{n}\cdot\sum_{j=0}^{n-1}\omega_n^{-ji}\cdot\hat{l}^{(t)}(\omega_n^j) = \frac{1}{n}\cdot\sum_{k=-\infty}^{\infty}\hat{l}_k^{(t)}\cdot\underbrace{\sum_{j=0}^{n-1}\omega_n^{j(k-i)}}_{=z} = \sum_{k=-\infty}^{\infty}\hat{l}_{i+kn}^{(t)}$$

By the properties of ω_n, $z = n$, if $k-i = 0 \bmod n$, and $z = 0$ otherwise. By Lemmata 1(b) and 2, the statement follows. □

The technique used above is called *multisection* (e g., see [3, p. 89]).

In the case of the n-cycle and the "two rounds" KS-regimen, i. e., $p(x) = x^{-1} + 2 + x$, it is particularly easy to get rid of the complex numbers, so we have explicitly

$$\begin{aligned} l_i^{(t)} &= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-ji} \cdot \frac{(\omega_n^{-j} + 2 + \omega_n^j)^{t+1} - 1}{\omega_n^{-j} + 1 + \omega_n^j} \\ &= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \cos\left(\frac{2\pi ji}{n}\right) \cdot \frac{(2 + 2\cos(\frac{2j\pi}{n}))^{t+1} - 1}{1 + 2\cos(\frac{2j\pi}{n})} \end{aligned}$$

Now we show that $l_i^{(t)}$ comes very close to the average load when t becomes large. That means we prove Theorem 1, reformulated appropriately.

Theorem 1 *For any global regimen described by $p(x)$ on a full circulant graph holds with total load $\hat{l}^{(t)}(1)$ and as t advances:*

$$l_i^{(t)} = \big(1 \pm o(1)\big) \cdot \frac{\hat{l}^{(t)}(1)}{n}.$$

Proof. Let $\lambda = \max_{0<j<n} |p(\omega_n^j)| = |p(\omega_n)|$. As $\lambda < p(1)$, $\lambda^t = o(p(1)^t$ for t growing. By applying Lemma 3, we obtain

$$l_i^{(t)} = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-ji} \cdot \hat{l}^{(t)}(\omega_n^j) = \frac{\hat{l}^{(t)}(1)}{n} + \underbrace{\frac{1}{n} \cdot \sum_{j=1}^{n-1} \omega_n^{-ji} \cdot \frac{p(\omega_n^j)^{t+1} - 1}{p(\omega_n^j) - 1}}_{=\Delta}$$

Δ is the deviation of the load in processor i after round t from the average load. Now we show that $|\Delta| = o(p(1)^t)$.

$$|\Delta| \leq \frac{1}{n} \cdot \sum_{j=1}^{n-1} \left| \frac{p(\omega_n^j)^{t+1} - 1}{p(\omega_n^j) - 1} \right| \leq \frac{1}{n} \cdot \sum_{j=1}^{n-1} \sum_{\tau=0}^{t} \left| p(\omega_n^j) \right|^{\tau} \leq \frac{n-1}{n} \cdot \frac{\lambda^{t+1} - 1}{\lambda - 1} = o(p(1)^t)$$

The first two estimations are true because of the Triangle Inequality. The ω_n^{-ji} factor can be dropped from the sum because of the Cauchy-Schwarz Inequality. □

For the "two rounds" KS-regimen mentioned above, $\lambda = 2 + 2\cos(2\pi/n) = 4\cos^2(\pi/n) \approx 4 - \frac{4\pi^2}{n^2}$.

5 Related Networks and Remarks

Now we consider evolving tree computations on a d-dimensional torus as host network. To model the development of the loads, we can first use a d-dimensional array that is infinite in all directions. For every dimension j, a formal variable x_j can be used, so this time the generating polynomial for the loads is $l^{(t)}(x_1, \ldots, x_d)$, as well as the regimen can be described by a polynomial $p(x_1, \ldots, x_d)$. When we have an $n_1 \times \cdots \times n_d$ torus (with appropriate internal chords reflecting the regimen), we have do multisection with $\omega_{n_1}, \ldots, \omega_{n_d}$ as described above. Then we can repeat the calculations of the proof of Theorem 1 which also shows that any regimen eventually lead to a good balancing of the tokens.

Our analysis also extends to case where more than one tree grows. In this case, $a^{(0)}(x)$ is not only just 1, but a polynomial. Also more complex regimens can be analyzed with our approach. Here, a regimen can consists of several polynomials $p_1(x), \ldots, p_m(x)$. Obviously. the development of loads is then described by the polynomial that is the product of the p_j.

Evolving tree computations are very similar to Random Walk on the host network. Besides using integers, the difference is the accumulation of load in a single processor. In [5], a Random Walk approach is used to analyze diffusive load-balancing algorithms. Here the number of tokens and their distribution is fixed, and the goal is to distribute the tokens evenly among the processors. In order to upper bound the deviation between the Random Walk where rational number occur and the distribution, in [5] the measure *local divergence* has been introduced which is similar in some respects to our $l_i^{(t)}$. Future work could relate the local divergence to evolving tree computations.

References

[1] S. Bhatt and J.-Y. Cai. Taking random walks to grow trees in hypercubes. *J. ACM*, 40:741–764, 1993.

[2] L.-X. Gao and A. L. Rosenberg. Toward efficient scheduling of evolving computations on rings of processors. *Journal of Parallel and Distributed Computing*, 38:92–100, 1996.

[3] D. E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, Reading, 3rd edition, 1997.

[4] K. Li. Asymptotically optimal randomized tree embedding in static networks. In *Proc. 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, pages 423–430, 1998.

[5] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. In *Proceeding of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 694–703, 1999.