Atomic Power in Forks: A Super-Logarithmic Lower Bound for Implementing Butterfly Networks in the Nonatomic Binary Fork-Join Model *

Michael T. Goodrich† Riko Jacob‡ Nodari Sitchinava§

Abstract

We prove an $\Omega(\log n \log\log n)$ lower bound for the span of implementing the n input, $\log n$-depth FFT circuit (also known as butterfly network) in the nonatomic binary fork-join model. In this model, memory-access synchronizations occur only through fork operations, which spawn two child threads, and join operations, which resume a parent thread when its child threads terminate. Our bound is asymptotically tight for the nonatomic binary fork-join model, which has been of interest of late, due to its conceptual elegance and ability to capture asynchrony. Our bound implies super-logarithmic lower bound in the nonatomic binary fork-join model for implementing the butterfly merging networks used, e.g., in Batcher's bitonic and odd-even mergesort networks. This lower bound also implies an asymptotic separation result for the atomic and nonatomic versions of the fork-join model, since, as we point out, FFT circuits can be implemented in the atomic binary fork-join model with span equal to their circuit depth.

1 Introduction

The parallel random access machine (PRAM) [33, 34] is a computational model where p synchronous processors share a common memory of potentially unbounded size. It has been studied for decades, with many interesting results, but it has also been criticized because its synchronization requirement that processors "march in lockstep" is unrealistic for modern parallel computer systems, where processors are asynchronous, due to such factors as different CPU speeds, caching effects, branch predictions, interrupts, and interactions between multiple concurrent jobs, e.g., see [9].

In contrast, the ***fork-join*** model of parallel computation (also known as the ***multi-threaded RAM*** [21, 23]) offers an alternative, as it captures asynchrony in a conceptually simple way, and it has gained further acceptance from being embodied in several parallel programming environments, e.g., see [2, 12, 22, 28, 35, 36, 37]. In the fork-join model, a dynamic set of ***threads*** shares a common memory of potentially unbounded size, with each thread comprising a sequential computation that can perform standard RAM instructions expressed in a program stored in memory. Each thread may also issue a `fork` instruction, which spawns k child threads (for a parameter $k \geq 2$) that immediately begin executing in parallel, and which, in turn, may spawn own child threads by issuing their own `fork` instructions. For every `fork` instruction, there is a corresponding `join` instruction, which acts as a barrier synchronization for all threads spawned by the associated `fork` instruction.[1] The performance of an algorithm in the fork-join model is measured by its ***work*** and ***span***. We will define these terms formally in Section 2, but, informally, work is equal to the total number of instructions and span is the length of a sequence of instructions that are always executed in serial, and is hence a lower bound on the parallel execution time, even if there are infinitely many processors.

One important detail, which is too often reduced to footnotes, is how threads are allowed to access the shared memory. In one version of the fork-join model, which we refer to as the ***atomic fork-join*** model, threads are allowed to read or write memory cells arbitrarily in the shared memory, with the assumption that the model supports some type of atomic memory-access primitive, which manages potential concurrent accesses to the same memory cell. For example, a `test-and-set` (TS) operation atomically tests whether a memory cell is 0 and, if so, sets it to 1 and returns true; otherwise, it returns false [1]. A `compare-and-set` (CAS) operation, which is also known as `compare-and-swap`, atomically

*This material is based upon work initiated during discussions at the AlgoPARC Workshop on Parallel Algorithms and Data Structures at the University of Hawaii at Manoa, in part supported by the National Science Foundation under grants CCF-1930579, CCF-1533823, and CCF-1911245.

†University of California, Irvine, USA, goodrich@uci.edu.

‡IT University of Copenhagen, Denmark, rikj@itu.dk.

§University of Hawaii at Manoa, USA, nodari@hawaii.edu.

[1] The recent work by Blelloch *et al.* [9] demonstrates that some algorithms don't need the `join` operations and their ***forking model*** doesn't even implement this instruction.

tests a memory location against a value, x, and updates this value to another value, y, if it was equal to x, e.g., see [32], Blelloch *et al.* [9] demonstrate that these atomic operations provide the threads with an ability to detect (and if needed impose) a specific order during concurrent write accesses.

In another version of the fork-join model, which we call the ***nonatomic fork-join*** model, there are no atomic memory access primitives, e.g., see [38, 21]. A thread in the nonatomic fork-join model may only write to memory cells that will not be read or written by a non-descendent thread, under any scheduling [3, 10, 13, 14, 11] of the threads.

Another consideration for the fork-join model is its fanout. In the ***binary fork-join*** model, a fork operation always creates exactly two child threads, i.e., $k = 2$, which better captures real-world constraints, e.g., see [9, 38, 37]. Indeed, it is easy to see that if k can be as large as the number of processors, p, then we can simulate a p-processor CREW PRAM[2] algorithm that runs in time T and work W in the nonatomic fork-join model with span $O(T)$ and work $O(W)$. Namely, for every step i, fork p child threads and have each thread perform step i for one processor and halt. Thus, if we allow arbitrary fanout, we might as well be working in the CREW PRAM model. Furthermore, if we also allow atomic operations, then by the same simulation approach, we can simulate any CRCW PRAM algorithm that runs in time T and work W in the atomic fork-join model with span $O(T)$ and work $O(W)$ [9]. Therefore, we focus on the *binary* fork-join model in this paper.

Much in the same way that CRCW PRAM model [6, 29, 30] imposes stronger hardware requirements on a parallel system than EREW PRAM model [18, 31], the atomic fork-join model imposes stronger requirements on the memory hardware and scheduling mechanisms than the nonatomic fork-join model does, e.g., see [10]. And a natural question is whether there is any substantive difference between the atomic and nonatomic versions of the binary fork-join model in terms of the performance and efficiency of the algorithms designed in these variants. The atomic binary fork-join model certainly seems more powerful, but are there any (natural) problems for which the atomic binary fork-join model has provably more efficient solutions than what is possible for the nonatomic binary fork-join model?

[2]The three variants of the PRAM model – Exclusive Read Exclusive Write (EREW), Concurrent Read Exclusive Write (CREW) and Concurrent Read Concurrent Write (CRCW) – differentiate if and what type of concurrent accesses are allowed in the algorithms designed in the respective model.

1.1 Prior Related Work. Duda and Czachórski [25] study several versions of fork-join synchronization primitives, including their visualization using computation DAGs. Mellor-Crummey [38] studies detecting data races in programs with nested fork-join parallelism, and also introduces the computation DAG concept for visualizing computations in the fork-join model. Bender *et al.* [7] also study data race detection for programs in the fork-join model, designing an on-the-fly method for maintaining dynamic computation DAGs.

Cole and Ramachandran explore the problem of false sharing [16, 17] in the fork-join model, where the local copy of a variable can become stale because of updates done by another thread, and they also provide a number of algorithms in the nonatomic fork-join model to deal with the problem. In particular, they present a sorting algorithm with $O(\log n \log\log n)$ span [19] and observe that the $O(n \log n)$-work algorithm of Frigo *et al.* [27] exhibits $O(\log n \log\log n)$ span when implemented in the nonatomic binary fork-join model [16]. Blelloch *et al.* [10] study properties related to the nonatomic fork-join model, showing that if a multithreaded parallel computation in the fork-join model is race free, write-after-read conflict free, and has work W and span S, a scheduler can guarantee a time bound of roughly $O(W/p + S)$ parallel time using p processors, even in the presence of faults.

Blelloch *et al.* [8] give several algorithms in the atomic fork-join model based on the CAS primitive, Dhulipala *et al.* [23] present a number of parallel graph algorithms in the atomic fork-join model, using the TS atomic primitive, and Blelloch *et al.* [9] give additional combinatorial algorithms in the atomic binary fork-join model, also using the TS primitive, including a randomized sorting algorithm with span $O(\log n)$ w.h.p. The elegance and efficiency of these algorithms for the atomic fork-join model therefore motivates the question we asked earlier, namely, whether there is a substantive difference between the atomic and nonatomic versions of the binary fork-join model in terms of performance and efficiency.

1.2 Our Results. The main result of this paper is a separation result that shows that the atomic version of the binary fork-join model is indeed more powerful than the nonatomic version in terms of its performance and efficiency for a well-known, natural computational problem. In particular, we show that any nonatomic binary fork-join implementation of an algorithm whose dependency graph contains a depth-k *FFT graph*, also known as a *butterfly network* and defined precisely in Section 2.3, requires a span of $\Omega(k \log k)$.

This result immediately implies $\Omega(\log n \log\log n)$ lower bound for the span of the Cooley-Tukey algorithm for computing Fast Fourier Transforms (FFT) [20] and the merging step in the Batcher's odd-even mergesort and bitonic mergesort [5]. Our lower bound implies that the $O(\log n \log\log n)$ span for computing FFT in the nonatomic binary fork-join model, as observed by Cole and Ramachandran [16], is asymptotically optimal.

To demonstrate the separation between the atomic and nonatomic binary fork-join models, we also show that the computation of FFT in the *atomic* binary fork-join model requires only $O(\log n)$ span.

2 Preliminaries

Throughout the paper all our logarithms are binary, i.e., $\log x = \log_2 x$, and we use the convention $0 \log 0 = \lim_{x\to 0^+} x \log x = 0$. We use the following standard graph-theoretic definitions: (a) in a directed acyclic graph (DAG), a ***source*** is a vertex with in-degree 0 and a ***sink*** is a vertex with out-degree 0; (b) $G[S]$ denotes a subgraph of $G = (V, E)$ induced by the subset of vertices $S \subseteq V$.

2.1 Computation and circuit graphs.

To prove the lower bound for computing FFT in the nonatomic binary-fork-join model, we will view the FFT computation as a circuit graph.

A ***circuit graph*** is a directed acyclic graph $H = (V, E)$, where the source vertices $V^{(0)} \subseteq V$ represent the input values for the computation and the remaining vertices $V \setminus V^{(0)}$ represent operations of the computation. An edge $(u, v) \in E$ indicates that the operation represented by v takes the output of u as input. The output of the whole computation is produced by the sink vertices.

To analyze the complexity of a parallel computation in a specific model $\mathcal{M}$, it is often convenient to view its implementation in that model via its ***computation DAG (cDAG)***. A cDAG in some model $\mathcal{M}$ is a directed acyclic graph $G_{\mathcal{M}} = (V, E)$, where for each operation there is a dedicated vertex $v \in V$ that represents this operation, and there is an edge $(u, v) \in E$ iff the operation represented by u is programmed to be executed directly before the operation represented by v. The precedence in the execution order can be, for example, because the two operations are programmed to be executed on a single (sequential) computing unit or due to a communication or synchronization requirement.

For simplicity, we will assume that every computation is defined via only elementary operations that take $O(1)$ time to execute, i.e., all complex operations are broken down into a sequence of elementary operations.[3] Then the complexity of a computation described by $G_{\mathcal{M}}$ is defined by two metrics:

- ***work*** is the number of vertices in $G_{\mathcal{M}}$, and
- ***span*** is the length of the longest path in $G_{\mathcal{M}}$.

Clearly, every cDAG $G_{\mathcal{M}}$ must satisfy the constraints of model $\mathcal{M}$ in which the computation is being implemented. For example, the cDAG of an implementation in any sequential model must be a linked list, while every connected subgraph of the cDAG of an implementation in a p-processor PRAM model must have a vertex separator of size at most p. In Section 2.2 we will describe the properties of cDAGs for computations in the nonatomic fork-join model.

Given a computation described by a circuit graph H, we can analyze its implementation in some computational model $\mathcal{M}$ by first defining a *valid* embedding of H into a cDAG $G_{\mathcal{M}}$ and then analyzing the work and span of $G_{\mathcal{M}}$.

DEFINITION 2.1. *For a circuit graph $H = (V, E)$ and cDAG $G_{\mathcal{M}} = (V', E')$, an embedding $\sigma\colon V \to V'$ is called **valid** if:*

1. *Every input (source) vertex $u \in V^{(0)}$ of H is mapped to the global source s of $G_{\mathcal{M}}$, i.e., $\sigma(u) = s$.*
2. *No two non-input vertices of H are mapped to the same vertex of $G_{\mathcal{M}}$.*
3. *For every pair of vertices $u, v \in V$: if there is a path from u to v in H, then there is a path from $\sigma(u)$ to $\sigma(v)$ in $G_{\mathcal{M}}$.*

In what follows, we can always add the global source s to $G_{\mathcal{M}}$ and map all source vertices $V^{(0)}$ to it. Therefore, we will not explicitly mention this mapping. Finally, we will omit the subscript $\mathcal{M}$, if the model of $G_{\mathcal{M}}$ is clear from the context.

2.2 Fork-join graphs.

We will refer to the cDAGs that satisfy the constraints of the nonatomic binary fork-join model as ***binary fork-join (BFJ) graphs***:[4]

DEFINITION 2.2. *A **binary fork-join (BFJ) graph** is a DAG $G_{s,t} = (V, E)$ with dedicated source and sink vertices $s, t \in V$, iff*

[3] Alternatively, the definition can be extended to weighted graphs, but it unnecessarily complicates the exposition.

[4] This definition can be easily generalized to the arbitrary-fanout fork-join model, but to keep the discussion focused, we only present the binary version here.

- $s = t = v$ *and* $G_{s,t} = G_{v,v} = (\{v\}, \emptyset)$*, i.e., a single vertex* $v = s = t$ *is a BFJ graph;*
- $G_{s,t} = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(t_1, s_2)\})$ *is a* ***series composition*** *of two BFJ graphs* $G'_{s,t_1} = (V_1, E_1)$ *and* $G''_{s_2,t} = (V_2, E_2)$*;*
- $G_{s,t} = (V_1 \cup V_2 \cup \{s, t\}, E_1 \cup E_2 \cup \{(s, s_1), (s, s_2), (t_1, t), (t_2, t)\})$ *is a* ***parallel composition*** *of two BFJ graphs* $G'_{s_1,t_2} = (V_1, E_1)$ *and* $G''_{s_2,t_2} = (V_2, E_2)$*.*

The two additional vertices s and t in the parallel composition correspond to the `fork` and `join` operations, respectively, and are the only ones in the BFJ graph with out- and in-degree of more than 1.

To reduce the clutter we will use G instead of $G_{s,t}$ when describing BFJ graphs, if the omission of the subscripts does not affect the clarity of exposition.

Each BFJ graph G admits a natural recursive decomposition, which can be represented as a rooted binary tree T_G. The root node of T_G represents the whole BFJ graph G. Every internal node u of T_G is either an S-node, or a P-node. An S-node u represents a subgraph G_u of G, which is a series composition of the subgraphs represented by the children of u. The left-to-right order of the children of u defines the order, in which the series composition is applied. A P-node u represents a subgraph G_u of G, which is a parallel composition of the subgraphs represented by the children of u. Finally, at the base case, the leaves of T_G represent individual vertices of G. Such decomposition tree allows for a recursive computation of the span of any BFJ graph G, which follows directly from the definition of BFJ graphs:

LEMMA 2.1. *The span of any BFJ subgraph* G_u *represented by a node* u *of the decomposition tree* T_G *is:*

- *1, if* u *is a leaf;*
- *the sum of the spans of the subgraphs of the two children of* u*, if* u *is an S-node; and*
- *2 plus the maximum of the spans of the subgraphs of the two children of* u*, if* u *is a P-node.*

The BFJ graphs are a special case of the series-parallel graphs [26] and the above decomposition tree is related to the ***sp-tree*** [15], which has been used to study the property of series-parallel graphs.[5] However, the generality of series-parallel graphs and sp-trees unnecessarily complicates the computation of the span for cDAGs and, consequently, our exposition in Section 4.

[5] sp-trees are a special case of SPQR trees [24], which have been used to study the properties of biconnected graphs.

The following definition of ***convexity*** for a subset of vertices of a DAG is central to our lower bound proof.

DEFINITION 2.3. (CONVEX VERTEX SUBSET) *In a DAG* $G = (V, E)$*, a subset of vertices* $S \subseteq V$ *is* ***convex*** *if for every pair of vertices* $u, v \in S$*, every vertex* w *on a directed path from* u *to* v *in* G *is also in* S*.*[6]

LEMMA 2.2. *Let* σ *be an arbitrary valid embedding of a DAG* $H = (V', E')$ *into a BFJ graph* $G = (V, E)$*,* T_G *be the decomposition tree of* G*, and for every node* x *of* T_G*, let* $G_x = (V_x, E_x)$ *be the subgraph of* G *represented by* x *and* $S_x := \{v \in V' : \sigma(v) \in V_x\}$*, i.e.,* S_x *be the subset of vertices of* H *embedded into* G_x*. Then* V_x *is convex in* G *and* S_x *is convex in* H*.*

Proof. Convexity of V_x follows from a straightforward induction on the height of x in T_G. To prove the convexity of S_x, for the sake of contradiction, let $w \in V' \setminus S_x$ be a vertex on some path $p_{u \rightsquigarrow v}$ from $u \in S_x$ to $v \in S_x$. Because σ is a valid embedding, by Property 3 of Definition 2.1, there must be paths $p_{\sigma(u) \rightsquigarrow \sigma(w)}$ and $p_{\sigma(w) \rightsquigarrow \sigma(v)}$ in G. Since V_x is convex, by the definition of convexity $\sigma(w) \in V_x$, i.e., w is embedded into G_x. Consequently, by the definition of S_x, w must be in S_x, which contradicts the assumption that $w \in V' \setminus S_x$. □

2.3 FFT Graphs. In Section 4 we show a lower bound on the span of the BFJ graph G if the circuit graph H that it embeds contains an ***FFT graph***. FFT graphs are defined in various ways, therefore, before we proceed let us present a concrete definition of the FFT graph, so we can refer to various terms throughout our exposition (see Figure 1 for an illustration).

DEFINITION 2.4. *An* ***FFT graph of order*** k*, for a non-negative integer* k*, is a directed acyclic graph* $F^k = (V, E)$ *defined recursively as follows.*

- $k = 0$*:* F^0 *is a single vertex, i.e.,* $V = \{u\}$ *and* $E = \{\}$*.*
- $k \geq 0$*: Let* $F_A^{k-1} = (V_A, E_A)$ *and* $F_B^{k-1} = (V_B, E_B)$ *be two FFT graphs of order* $k - 1$*, each with* m *sources and* m *sinks* $\{u_0, \ldots, u_{m-1}\} \subseteq V_A$ *and* $\{u_m, \ldots, u_{2m-1}\} \subseteq V_B$*, respectively. Let* $V_C = \{v_0, \ldots, v_{2m-1}\}$ *be a set of* $2m$ *additional vertices. Then* $V = V_A \cup V_B \cup V_C$ *and* $E = (E_A \cup E_B) \bigcup_{0 \leq i \leq m-1} (u_i, v_i) \cup (u_{m+i}, v_i) \cup (u_i, v_{m+i}) \cup (u_{m+i}, v_{m+i})$*.*

[6] Our definition of convexity differs from the definition in the metric graph theory (defined on undirected graphs), where convex subgraph contains the vertices of only the shortest paths between every pair of vertices [4].

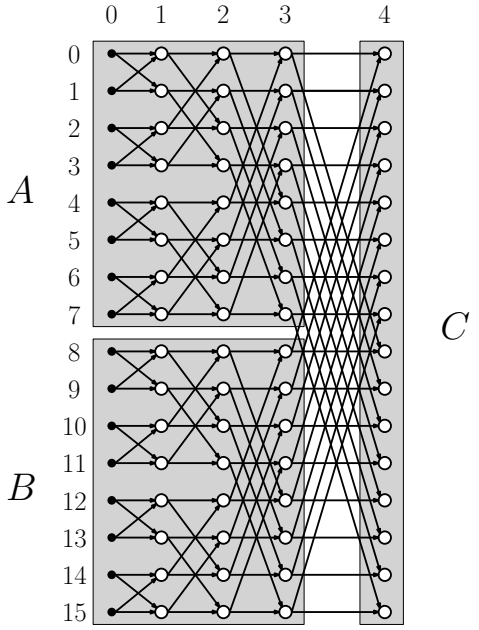

Figure 1: An example of F^4 as a composition of a pair of F^3 (A and B), followed by 2^4 additional vertices (C) that become the sinks of F^4. The vertices on level 0 are of different shape to indicate that they perform no computation.

Observe that the subgraph induced by every quadruplet of vertices u_i, u_{m+i}, v_i, and v_{m+i} in the above definition is an FFT graph of order 1. We call the pair of non-source vertices v_i and v_{m+i} in the above definition the ***companion vertices***.

Clearly, FFT graph is a circuit graph. FFT graphs appear as subgraphs of circuit implementations of various algorithms. The companion vertices in these implementations typically compute complimentary operations. For example:

- In the radix-2 Cooley-Tukey algorithm for computing the Fast Fourier Transform [20] (this is how FFT graph gets its name) the companion vertices implement either an addition or a subtraction of the two input values, one of which is scaled by the complex root of unity (the so-called "twiddle" factor).
- In the merge step of Batcher's bitonic mergesort and odd-even mergesort networks [5] the companion vertices compute the minimum and the maximum of the two input values. Typically, descriptions of these networks combine every pair of companion vertices into a single ***compare-and-exchange*** gate.

Observe that requiring that the companion vertices are executed together as a single gate is (slightly) more restrictive. However, our lower bound in Section 4 assumes no such restrictions, as long as the execution satisfies the dependencies defined by the edges of the FFT graph. Therefore, it immediately implies the $\Omega(\log n \log\log n)$ lower bound for the implementations of the above algorithms in the nonatomic binary fork-join model. At the same time, in Section 3 we will show a matching upper bound for implementing these more restrictive algorithms in the nonatomic binary fork-join model.

2.3.1 Visualizing FFT graphs. It is easy to show by induction that F^k consists of $2k \cdot 2^k$ edges and $(k+1) \cdot 2^k$ vertices, of which 2^k are sources and 2^k are sinks, and that every non-source vertex has two incoming edges and every non-sink vertex has two outgoing edges.

It is often convenient to visualize F^k as in Figure 1, namely, being embedded on a $2^k \times (k+1)$ grid in the plane, placing the 2^k sinks in the k-th column, on rows $0, \ldots, 2^k - 1$, and the two FFT graphs of order $k-1$ embedded recursively in columns $0, \ldots, k-1$: one on rows $0, \ldots, 2^{k-1} - 1$ and the other one on rows $2^{k-1}, \ldots, 2^k - 1$. Let i_j denote an integer, such that $|i - i_j| = 2^j$, i.e., the binary representations of i and i_j differ precisely in the j-th least significant bit. Then we can define a canonical label $v_{i,j}$ for each vertex in the i-th row and j-th column of the grid, and for each $0 \le i < 2^k$ and $0 \le j < k$, there is a pair of directed edges $(v_{i,j}, v_{i,j+1})$ and $(v_{i_j,j}, v_{i,j+1})$.

Such visualization makes it obvious how to implement any algorithm, whose circuit graph is an FFT graph F^k, in the EREW PRAM model using $m = 2^k$ processors in $O(k)$ parallel time in-place: given an input stored in an array $A[0..(m-1)]$, in the j-th step, $0 \le j < k$, the processor p_i reads $A[i]$ and $A[i_j]$ as the operands to the operation defined by the vertex $v_{i,j+1}$ and writes the result back to $A[i]$. Therefore, we say j is the ***level*** of $v_{i,j}$ and denote all vertices of F^k on level j by $V^{(j)}$.

3 Upper Bounds

We start with some simple upper bounds.

It has been noted before that the algorithm of Frigo *et al.* [27] computes FFT on n values in the nonatomic binary fork-join model in span $O(\log n \log\log n)$. The following is a slightly simplified description of their algorithm and highlights the structure of the FFT graph that guided us in our discovery of the lower bound presented in Section 4.

Given an FFT graph F^k, consider the first $\lfloor (k+1)/2 \rfloor$ levels of F^k. These levels define $2^{\lceil k/2 \rceil}$ independent instances of $F^{\lfloor k/2 \rfloor}$, each of which can

be solved recursively in parallel. Similarly, the last $k - \lfloor (k+1)/2 \rfloor = \lceil k/2 \rceil$ levels of F^k, plus the outputs of the preceding level, define $2^{\lfloor k/2 \rfloor}$ independent instances of $F^{\lceil k/2 \rceil}$, each of which can also be solved recursively in parallel. Using a complete binary tree of forks and joins it takes $O(k)$ span to spawn and terminate so many parallel recursive calls in the nonatomic binary fork-join model. Thus, if we let $n = 2^k$, the span of such an implementation is defined by the recurrence $T(n) = 2T(\sqrt{n}) + O(\log n)$, which solves to $O(\log n \log\log n) = O(k \log k)$.

As mentioned earlier, in some applications the companion vertices of the FFT graph need to be implemented together as a single operation (or at least immediately after each other). We call such variation of the FFT graph a *comparator-FFT graph*:

Definition 3.1. *The **comparator-FFT graph of order** k is an FFT graph F^k, where each pair of the non-source companion vertices are united into a single two-input and two-output **comparator gate**. The two operations of each comparator gate are always executed serially (resulting in span 2).*

Observe that in the above recursive decomposition of F^k, every pair of companion vertices is assigned to the same instance of $F^{\lfloor k/2 \rfloor}$ or $F^{\lceil k/2 \rceil}$. Thus, if we stop the recursion when the base case consists of F^1, the above algorithm can be applied to the comparator-FFT graphs too, leading us to the following result.

Theorem 3.1. *For $k \geq 1$, the comparator-FFT graph of order k can be implemented with span $O(k \log k)$ in the nonatomic binary fork-join model.*

In contrast, in the *atomic* binary fork-join model we can achieve a better span:

Theorem 3.2. *For $k \geq 1$, the comparator-FFT graph of order k can be implemented with span $O(k)$ in the atomic binary fork-join model.*

Proof. For each comparator gate g, create two memory cells $x_{1,g}$ and $x_{2,g}$, as well as a control variable c_g initialized to 0. The cells $x_{1,g}$ and $x_{2,g}$ will be used to store the two input values for gate g as they become available.

The computation starts by spawning 2^k threads, as a complete binary tree of fork operations in span k. Each thread starts by reading a distinct input value, represented by the sources of the comparator-FFT graph and proceeds by following the edges of the comparator-FFT graph to the next gate. When a thread reaches g, it writes the value it holds to either $x_{1,g}$ or $x_{2,g}$, depending on whether it holds the first ($z = 1$) or the second ($z = 2$) input value for g. Then using an atomic operation (either CAS or TS) the thread checks if $c_g = 0$ and sets it to 1. If this assignment succeeds, it indicates that the thread was the first of the two to reach g and the thread is done. If the assignment fails (because $c_g \neq 0$), it indicates that the thread is second to reach g. Therefore, it reads the other input value from the appropriate $x_{z,g}$, executes the operation of g and forks two child threads, providing them with the output values of g (one for each thread). The newly spawned threads will proceed along the two out-edges of g. At the last level, all threads are terminated by issuing join operations in a binary tree fashion. This requires span $O(\log(k2^k)) = O(k)$, since we issued a binary fork for each gate in the comparator-FFT graph. □

Since the comparator-FFT graphs are more restrictive than FFT graphs, the above results imply that we can also implement any n-inputs, $O(\log n)$-depth FFT graph in the atomic binary fork-join model with span $O(\log n)$ and in the nonatomic binary fork-join model with span $O(\log n \log\log n)$.

4 Lower Bound

The main contribution of this paper is a matching lower bound for the nonatomic binary fork-join model. However, before we proceed, we need to define some terms and prove several simple, but important technical lemmas.

4.1 Technical Lemmas.

Definition 4.1. (Cut) *Given a DAG $G = (V, E)$ and a subset $S \subseteq V$, a **cut-edge** of $G[S]$ is a directed edge $(u, v) \in E$, such that $u \notin S$ and $v \in S$. The set $C_{G[S]}$ of cut-edges is called the **cut** and its cardinality $|C_{G[S]}|$ is called the **cut size**.*

Observe that the cut size of the set of all computing (non-source) vertices of an FFT graph F^k of order k is $|C_{F^k[V \setminus V^{(0)}]}| = 2 \cdot 2^k$.

Lemma 4.1. (Monotonicity of cut sizes) *Let $F^k = (V, E)$ be an FFT graph of order k. For any pair of convex vertex subsets $S \subseteq Q \subseteq V \setminus V^{(0)}$: $|C_{F^k[S]}| \leq |C_{F^k[Q]}|$.*

Proof. For every edge $e = (v_{i,j}, v_{i',j+1}) \in F^k[Q]$ we define a ***predecessor edge***

$$e^\pi = \begin{cases} (v_{i,j-1}, v_{i,j}) & \text{if } i' = i \\ (v_{i_{j-1},j-1}, v_{i,j}) & \text{if } i' = i_j \end{cases}$$

For each cut-edge $e \in C_{F^k[S]}$, let $\mu(e) \in C_{F^k[Q]}$ be the first cut-edge of $F^k[Q]$ reached by following the

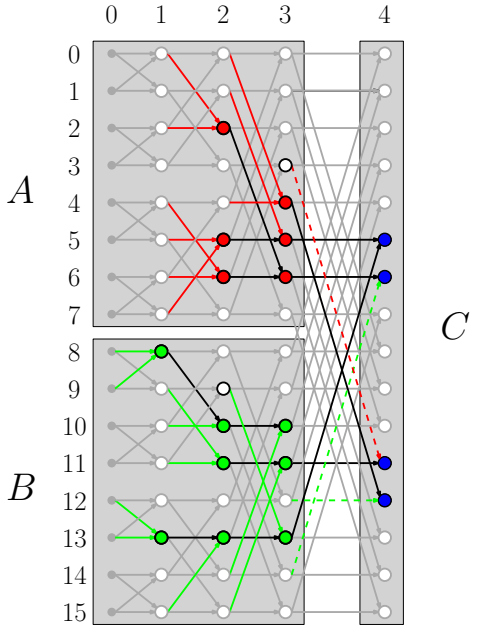

Figure 2: An illustration for the proof of Lemma 4.2. The vertices of S_A, S_B, and S_C are red, green and blue, respectively. The colored solid edges are the cut edges of $\mathcal{C}_A$ and $\mathcal{C}_B$, and colored dashed edges are the cut edges $\mathcal{C}_C^A$ and $\mathcal{C}_C^B$.

predecessor edges starting from e. $\mu(e)$ exists because $S \subseteq Q$ and Q contains no source vertices. The set of edges visited during such a traversal defines a directed path p from $\mu(e)$ to e in F^k. Since S is convex, e is the only cut-edge of $F^k[S]$ in p. Moreover, our definition of the predecessor ensures that no two edges in $F^k[Q]$ share a predecessor. Therefore, all such paths from $C_{F^k[Q]}$ to $C_{F^k[S]}$ are edge disjoint, i.e, $\mu(e)$ is unique to each $e \in S$. Hence, the mapping $\mu : C_{F^k[S]} \to C_{F^k[Q]}$ is injective and we can conclude that $|C_{F^k[S]}| \leq |C_{F^k[Q]}|$. ∎

Lemma 4.2. (Density Lemma) *Given an FFT graph $F^k = (V, E)$, for every convex vertex subset $S \subseteq V \setminus V^{(0)}$ of size $|S| = n$ and cut size $|C_{F^k[S]}| = m$, we have $n \leq f(m) = (m/2)\log m$.*

Proof. Let $F^{k_{\min}}$ be the smallest FFT graph that is a supergraph of $F^k[S]$. By the recursive definition of the FFT graph, the supergraph $F^{k_{\min}}$ decomposes into two FFT graphs, $F_A^{k_{\min}-1} = (V_A, E_A)$ and $F_B^{k_{\min}-1} = (V_B, E_B)$, $2^{k_{\min}}$ sink vertices V_C, and $2 \cdot 2^{k_{\min}}$ edges E_C connecting vertices of V_A and V_B to vertices of V_C (see Figure 2 for an illustration). Let $S_A = S \cap V_A$, $S_B = S \cap V_B$, $S_C = S \cap V_C$, and for the simplicity of exposition define $n_A = |S_A|$, $n_B = |S_B|$, $n_C = |S_C|$, $m_A = |C_{F^{k-1}[S_A]}|$, and $m_B = |C_{F^{k-1}[S_B]}|$. Observe that there are no edges between V_A and V_B in F^k, and the three subsets S_A, S_B, and S_C define a partition of S, i.e., $|S| = n = n_A + n_B + n_C$.

The proof is by induction on $k_{\min}$. Assume inductively that the lemma holds for every positive $k' < k_{\min}$, i.e., $n_A \leq f(m_A)$ and $n_B \leq f(m_B)$.

Since every cut-edge of $C_{F^k[S]}$ ends at a vertex of either S_A, S_B or S_C, it defines a natural partition of $C_{F^k[S]}$ into subsets $\mathcal{C}_A$, $\mathcal{C}_B$, and $\mathcal{C}_C$. Every cut-edge of $\mathcal{C}_C$ starts at a vertex from either V_A or V_B. Thus, we can further partition $\mathcal{C}_C$ into $\mathcal{C}_C^A$ and $\mathcal{C}_C^B$. Let $m'_A = |\mathcal{C}_A| + |\mathcal{C}_C^A|$ and $m'_B = |\mathcal{C}_B| + |\mathcal{C}_C^B|$. Then $m = |C_{F^k[S]}| = |\mathcal{C}_A| + |\mathcal{C}_B| + \left(|\mathcal{C}_C^A| + |\mathcal{C}_C^B|\right) = m'_A + m'_B$. W.l.o.g. assume $m'_A \leq m'_B$, i.e., $0 \leq m'_A \leq m/2$.

Claim 4.1. $n_C = |S_C| \leq \min\{|\mathcal{C}_A| + |\mathcal{C}_C^A|, |\mathcal{C}_B| + |\mathcal{C}_C^B|\} = \min\{m'_A, m'_B\} = m'_A$

Proof. Let $S'_A \subseteq S_A$ and $V'_A \subseteq V_A \setminus S_A$ be two subsets of vertices of V_A that are neighbors of S_C. Since every vertex of S_C has a unique neighbor from V_A, $n_C = |S_C| = |S'_A| + |V'_A|$. Clearly, the edges from V'_A to S_C are cut-edges of $F^{k_{\min}}[S]$, i.e., $|V'_A| = |\mathcal{C}_C^A|$. Thus, all that remains to show is that $|S'_A| \leq |\mathcal{C}_A|$.

Since all vertices of S'_A are on the same (last) level of $F^{k_{\min}-1}$, clearly, $|S'_A| \leq |C_{F^{k_{\min}-1}[S'_A]}|$. Observe that both S'_A and S_A are convex vertex subsets and since $S'_A \subseteq S_A \subseteq S \subseteq V \setminus V^{(0)}$, by Lemma 4.1 $|C_{F^{k_{\min}-1}[S'_A]}| \leq |C_{F^{k_{\min}-1}[S_A]}|$. Finally, observe that $C_{F^{k_{\min}-1}[S_A]} = C_{F^k[S_A]} = \mathcal{C}_A$ because there are no edges between V_A and V_B in $F^{k_{\min}}$. ∎

Thus, using the inductive hypothesis, we can bound $n = n_A + n_B + n_C \leq f(m_A) + f(m_B) + m'_A$. The absence of edges between V_A and V_B in $F^{k_{\min}}$ implies that $m_A \leq m'_A$ and $m_B \leq m'_B$, and since the function $f(x)$ is convex, we get:

$$n \leq f(m'_A) + f(m'_B) + m'_A = f(m'_A) + f(m - m'_A) + m'_A$$

The right hand side is a convex function with respect to m'_A in the range $m'_A \in [0, m/2]$. Therefore, it is maximized at either boundary of this range. The case $m'_A = 0$ is the statement of the lemma, and the case $m'_A = m/2$ results in the inequality:

$$n \leq f(m/2) + f(m/2) + m/2 = \frac{m}{2}\log\frac{m}{2} + \frac{m}{2} = \frac{m}{2}\log m = f(m)$$

Finally, in the base case ($k_{\min} = 0$), F^0 consists of a single vertex. Since $S \subseteq V \setminus V^{(0)}$, i.e., contains no vertices of level 0, S must be empty ($n = 0$ and $m = 0$) and the base case follows from the convention $0 \log 0 = 0$. ∎

4.2 FFT Lower Bound. We are ready to prove the lower bound on the span of the BFJ graph G for any embedding of an FFT graph F^k into G. We proceed by first defining the ***binary fork-join lower-bound property*** and showing that any function ϕ satisfying this property bounds the span of G from below. Then we present a concrete ϕ and show that when applied to F^k, the value of this ϕ is bounded by $\Omega(k \log k)$.

DEFINITION 4.2. *A function $\phi\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$, has the **binary fork-join lower-bound property** if it fulfills the following three conditions:*

Base Case: $\phi(2,1) \leq 1$.

Parallel Decomposition: *For every $2 \leq m \leq 2n$ and any $1 < x < m-1$, $1 \leq y \leq n-1$ such that $1 < x \leq 2y$, we have $\max\{\phi(x,y), \phi(m-x, n-y)\} \geq \phi(m,n) - 1$.*

Serial Decomposition: *For every m and n, such that $2 \leq m \leq 2n \leq m \log m$, and every x, y, and $\tilde{x}$, such that $1 \leq y \leq n-1$, $2 \leq x \leq \min\{2y, m\}$, $\max\{2, m-x\} \leq \tilde{x} \leq \min\{2(m-y), m\}$, we have $\phi(x,y) + \phi(\tilde{x}, n-y) \geq \phi(m,n)$.*

LEMMA 4.3. *Let $F^k = (V,E)$ be an FFT graph of order $k \geq 1$, let G be a BFJ graph into which F^k is embedded, and let ϕ be a function with the binary fork-join lower bound property. Then, the span of G is at least $\phi(2 \cdot 2^k, k \cdot 2^k)$.*

Proof. Let T_G be the decomposition tree of G and for every node u of T_G, let G_u be the subgraph of G that is represented by u, $V_u \subseteq V \setminus V^{(0)}$ be the set of vertices of F^k being embedded into G_u, let $n_u = |V_u|$, and $m_u = |C_{F^k[V_u]}|$. We will show that for every node u of T_G, such that G_u embeds at least one vertex of $V \setminus V^{(0)}$, the span $span(G_u) \geq \phi(m_u, n_u)$. This will imply that $span(G) = span(G_{root}) \geq \phi(m_{root}, n_{root}) = \phi(2 \cdot 2^k, k \cdot 2^k)$, because the root represents the whole graph G, which embeds the set $V_{root} = V \setminus V^{(0)}$ of F^k, which is of size $n_{root} = k \cdot 2^k$, and has cut size $m_{root} = 2 \cdot 2^k$.

First, let us derive the bounds on n_u and m_u. Since every G_u embeds at least one non-source vertex of F^k, there are at least 2 cut edges in $F^k[V_u]$, i.e., $m_u \geq 2$. On the other hand, every vertex of F^k has 2 incoming edges, therefore, $m_u \leq 2n_u$. Finally, by Lemma 2.2, V_u is convex subset of V, so we can apply Lemma 4.2 to V_u to obtain the final inequality:

$$2 \leq m_u \leq 2n_u \leq m_u \log m_u \tag{4.1}$$

The rest of the proof is by induction on the height of u. At the base case, u is a leaf and G_u embeds a single vertex ($n_u = 1$) of F^k. Since G_u is non-empty, its span is $span(G_u) \geq 1$, and since the cut size of any single vertex in $V \setminus V^{(0)}$ is $m_u = 2$ and ϕ has the binary fork-join lower-bound property, $1 \geq \phi(2,1) = \phi(m_u, n_u)$.

Now consider an arbitrary internal node u with two children v and w. Assume inductively that $span(G_v) \geq \phi(m_v, n_v)$ and $span(G_w) \geq \phi(m_w, n_w)$. First, observe that if either $|V_v| = 0$ or $|V_w| = 0$, then $span(G_u) \geq \phi(m_u, n_u)$. In particular, without loss of generality assume $|V_v| = 0$, i.e., G_v embeds no vertices of $V \setminus V^{(0)}$. Then $n_u = n_w$ and $m_u = m_w$. By Lemma 2.1, $span(G_u) \geq \max\{span(G_v), span(G_w)\}$. Therefore, $span(G_u) \geq span(G_w) \geq \phi(m_w, n_w) = \phi(m_u, n_u)$, where the last inequality is the inductive hypothesis.

Thus, for the rest of the proof we can assume that both V_v and V_w are non-empty. Moreover, they form a partition of V_u, i.e., $n_u = n_v + n_w$ and $1 \leq n_v \leq n_u - 1$ and, consequently, $C_{F^k[V_u]} \subseteq C_{F^k[V_v]} \cup C_{F^k[V_w]}$, i.e., $m_u \leq m_v + m_w$.

There are two cases to consider:

1. **Node u is a parallel composition.** By Lemma 2.1 and the inductive hypothesis: $span(G_u) = 2 + \max\{span(G_v), span(G_w)\} \geq 2 + \max\{\phi(m_v, n_v), \phi(m_w, n_w)\}$. Since G_u is a parallel composition of G_v and G_w, all operations of G_v and G_w can be computed independent of each other, i.e., there are no edge between G_v and G_w and, consequently, each cut edge of $F^k[V_v]$ or $F^k[V_w]$ is also a cut edge of $F^k[V_u]$. Therefore, $m_u = m_v + m_w$ and, using inequalities (4.1), we get $2 \leq m_v \leq m_u - 2$. Thus, we can use that ϕ has the binary fork-join lower-bound property to obtain a lower bound on the span of G_u:
$$\begin{aligned} span(G_u) &\geq 2 + \max\left\{ \begin{array}{l} \phi(m_v, n_v), \\ \phi(m_u - m_v, n_u - n_v) \end{array} \right\} \\ &\geq 1 + \phi(m_u, n_u) \geq \phi(m_u, n_u) \end{aligned}$$
2. **Node u is a serial composition.** By Lemma 2.1 and the inductive hypothesis: $span(G_u) = span(G_v) + span(G_w) \geq \phi(m_v, n_v) + \phi(m_w, n_w)$. By Lemma 2.2, V_v and V_w are convex in F^k, and since $V_v \subseteq V_u$ and $V_w \subseteq V_u$, by Lemma 4.1, $m_v \leq m_u$ and $m_w \leq m_u$. And we have already established that $m_v + m_w \geq m_u$. Thus, we can use that ϕ has the binary fork-join lower-bound property to lower bound the span of u:
$$S_u \geq \phi(m_v, n_v) + \phi(m_w, n_u - n_v) \geq \phi(m_u, n_u)$$

∎

THEOREM 4.1. *Every implementation of an algorithm whose circuit graph contains an FFT graph with m input*

vertices requires a span of at least $\Omega((1+\log\log m)\cdot\log m)$ *in the nonatomic binary fork-join model.*

Proof. An FFT graph with m input vertices is of order $k=\log m$. Consider the following function:

$$\phi(m,n)=2+\log m+\tau(m,n),$$

where $\tau(m,n)=\frac{2n}{m}\log\frac{2n}{m}$. In the rest of this paper we will show that $\phi(m,n)$ exhibits the binary fork-join lower-bound property. Then by Lemma 4.3, the span of G is at least

$$\begin{aligned}\phi\left(2m, m\log m\right) &= 2+\log(2m)+\frac{2m\log m}{2m}\log\frac{2m\log m}{2m}\\ &= 3+(1+\log\log m)\cdot\log m\\ &\geq (1+\log\log m)\cdot\log m.\end{aligned}$$

∎

So all that remains is to prove that $\phi(m,n)$ exhibits the binary fork-join lower-bound property. Clearly, this function satisfies the Base Case condition. Since we will be using Calculus to lower bound ϕ, we will prove that $\phi(x,y)$ fulfills the Parallel Decomposition and the Serial Decomposition conditions for any positive real arguments, i.e., for any $x,y\in\mathbb{R}^+$. Then clearly $\phi(m,n)$ will satisfy these properties for $m,n\in\mathbb{N}^+$ as well.

A note on Calculus notation. For the sake of clarity, let us define the notation we use. Given a function f on two variables, we denote the partial derivative with respect to each variable as functions $D_1 f$ and $D_2 f$. Then we can use the notation $D_i f(a,b)$ as the value of the partial derivative with respect to the i-th variable evaluated at the point (a,b). Finally, if the arguments to function f can be expressed as functions of a single variable x, e.g. $g_1(x)$ and $g_2(x)$, then the value of the derivative $\frac{df}{dx}$ at the point $(g_1(x),g_2(x))$ is defined via the *chain rule* as follows:

$$\begin{aligned}\frac{d}{dx}f(g_1(x),g_2(x)) &= D_1 f(g_1(x),g_2(x))\cdot\frac{d}{dx}g_1(x)\\ &\quad + D_2 f(g_1(x),g_2(x))\cdot\frac{d}{dx}g_2(x).\end{aligned}$$

The partial derivatives of ϕ with respect to the two arguments are (the derivations can be found in the Appendix):

$$D_1\phi(x,y)=-\left(\frac{\tau(x,y)}{x}+\frac{2y-x}{x^2\ln 2}\right) \tag{4.2}$$

$$D_2\phi(x,y)=\frac{\tau(x,y)}{y}+\frac{2}{x\ln 2} \tag{4.3}$$

Observe that if $D_2\phi(x,y)\neq 0$, we can define

$$g(x,y)=\frac{D_1\phi(x,y)}{D_2\phi(x,y)}=\frac{1}{2}\cdot\left(\frac{1}{\ln\frac{2y}{x}+1}-\frac{2y}{x}\right) \tag{4.4}$$

LEMMA 4.4. *For any* $1<x\leq 2y$ *the partial derivatives* $D_1\phi(x,y)\leq 0$ *and* $D_2\phi(x,y)>0$, *where the equality holds when* $x=2y$.

Proof. Follows from the condition $1<x\leq 2y$. ∎

The next two lemmas prove that $\phi(x,y)$ fulfills the last two conditions of the binary fork-join lower-bound property.

LEMMA 4.5. (PARALLEL DECOMPOSITION CONDITION) *For every* $2\leq m\leq 2n$ *and any* $1<x<m-1$, $1\leq y\leq n-1$ *such that* $1<x\leq 2y$:

$$\max\{\phi(x,y),\phi(m-x,n-y)\}\geq\phi(m,n)-1.$$

Proof. Fix arbitrary m,n, that satisfy the condition $1<m\leq 2n$. Throughout the proof we will use the shorthand notation $\tilde{x}=m-x$ and $\tilde{y}=n-y$.

First observe that for $x=\tilde{x}=m/2$ and $y=\tilde{y}=n/2$

$$\max\{\phi(x,y),\phi(\tilde{x},\tilde{y})\}=\phi(m/2,n/2)=\phi(m,n)-1$$

It remains to show that this choice of x and y is minimal.

Observe that since $\phi(x,y)$ is a monotonic function in both parameters and the domain is convex, it is sufficient to consider the situation when $\phi(x,y)=\phi(\tilde{x},\tilde{y})$ (otherwise, we could adjust one of the parameters slightly inwards—e.g. move y towards $\tilde{y}$—and reduce the maximum). To this end for the rest of the proof we consider the manifold defined by the constraint $\phi(x,y)=\phi(\tilde{x},\tilde{y})$. Since m and n are fixed, $\tilde{x}$ and $\tilde{y}$ are functions of x and y, respectively, i.e., $\tilde{x}=\tilde{x}(x)$ and $\tilde{y}=\tilde{y}(y)$; and the manifold is defined by only two variables: x and y. Moreover, since ϕ is monotone in y, y can be expressed as a function of x, i.e., $y=y(x)$, and, consequently, $\tilde{y}=\tilde{y}(y(x))$. Finally, since ϕ is differentiable, so is $y(x)$. Let us denote its derivative evaluated at x by $y'(x)$.

Thus, we can compute the derivatives of $\phi(x,y(x))$ and $\phi(\tilde{x}(x),\tilde{y}(y(x)))$ using the chain rule:

$$\frac{d}{dx}\phi(x,y(x))=D_1\phi(x,y)+D_2\phi(x,y)\cdot y'(x) \tag{4.5}$$

$$\begin{aligned}\frac{d}{dx}\phi(\tilde{x}(x),\tilde{y}(y(x))) &\\ = -D_1\phi(\tilde{x},\tilde{y})-D_2\phi(\tilde{x},\tilde{y})\cdot y'(x)&\end{aligned} \tag{4.6}$$

Since we are considering the function on the manifold $\phi(x,y) = \phi(\tilde{x},\tilde{y})$, we can conclude that the right-hand sides of the above equations are equal to each other, giving us:

$$y'(x) = -\frac{D_1\phi(x,y) + D_1\phi(\tilde{x},\tilde{y})}{D_2\phi(x,y) + D_2\phi(\tilde{x},\tilde{y})}$$

And plugging it into Equation (4.5) we get:

$$\begin{aligned}
&\frac{d}{dx}\phi(x,y(x)) \\
&\quad = D_1\phi(x,y) - D_2\phi(x,y)\cdot\frac{D_1\phi(x,y) + D_1\phi(\tilde{x},\tilde{y})}{D_2\phi(x,y) + D_2\phi(\tilde{x},\tilde{y})} \\
&\quad = \frac{D_1\phi(x,y)\cdot D_2\phi(\tilde{x},\tilde{y}) - D_1\phi(\tilde{x},\tilde{y})\cdot D_2\phi(x,y)}{D_2\phi(x,y) + D_2\phi(\tilde{x},\tilde{y})}
\end{aligned}$$

By Lemma 4.4 the denominator is always positive for any $1 < x \le 2y$. Therefore, $\frac{d}{dx}\phi(x,y(x)) = 0$ iff

$$D_1\phi(x,y)\cdot D_2\phi(\tilde{x},\tilde{y}) = D_1\phi(\tilde{x},\tilde{y})\cdot D_2\phi(x,y)$$

One solution to this equation is $x = \tilde{x} = m/2$ and $y = \tilde{y} = n/2$. To show that there are no other solutions, we will show that for all $1 < x < m/2$ the derivative $\frac{d}{dx}\phi(x,y(x))$ on the manifold is negative, and for all $m/2 < x < m-1$ the derivative $\frac{d}{dx}\phi(x,y(x))$ on the manifold is positive.

$$\begin{aligned}
&\frac{d}{dx}\phi(x,y(x)) \\
&\quad = \frac{D_2\phi(x,y)\cdot D_2\phi(\tilde{x},\tilde{y})\cdot\left(\frac{D_1\phi(x,y)}{D_2\phi(x,y)} - \frac{D_1\phi(\tilde{x},\tilde{y})}{D_2\phi(\tilde{x},\tilde{y})}\right)}{D_2\phi(x,y) + D_2\phi(\tilde{x},\tilde{y})} \\
&\quad = \frac{D_2\phi(x,y)\cdot D_2\phi(\tilde{x},\tilde{y})}{D_2\phi(x,y) + D_2\phi(\tilde{x},\tilde{y})}\cdot\left(\frac{D_1\phi(x,y)}{D_2\phi(x,y)} - \frac{D_1\phi(\tilde{x},\tilde{y})}{D_2\phi(\tilde{x},\tilde{y})}\right) \\
&\quad = \frac{D_2\phi(x,y)\cdot D_2\phi(\tilde{x},\tilde{y})}{D_2\phi(x,y) + D_2\phi(\tilde{x},\tilde{y})}\cdot(g(x,y) - g(\tilde{x},\tilde{y}))
\end{aligned}$$

Because $D_2\phi(x,y) > 0$ and $D_2\phi(\tilde{x},\tilde{y}) > 0$ for any $1 < x \le 2y$, the sign of $\frac{d}{dx}\phi(x,y(x))$ depends only on the sign of the difference $g(x,y) - g(\tilde{x},\tilde{y})$.

Let $h(t) = \frac{1}{2}\cdot\left(\frac{1}{1+\ln t} - t\right)$ and observe that

$$\begin{aligned}
g(x,y) &= \frac{D_1\phi(x,y)}{D_2\phi(x,y)} = \frac{-\left(\frac{\tau(x_1,x_2)}{x} + \frac{2y-x}{x^2\ln 2}\right)}{\frac{\tau(x_1,x_2)}{y} + \frac{2}{x\ln 2}} \\
&= -\frac{y}{x}\cdot\frac{\tau(x_1,x_2)\cdot x\ln 2 + 2y - x}{\tau(x_1,x_2)\cdot x\ln 2 + 2y} \\
&= -\frac{y}{x}\cdot\frac{\frac{2y}{x}\log\frac{2y}{x}\cdot x\ln 2 + 2y - x}{\frac{2y}{x}\log\frac{2y}{x}\cdot x\ln 2 + 2y} \\
&= -\frac{\frac{2y}{x}\ln\frac{2y}{x} + \frac{2y}{x} - 1}{2\ln\frac{2y}{x} + 2} \\
&= -\frac{y}{x} + \frac{1}{2\ln\frac{2y}{x} + 2} \\
&= \frac{1}{2}\cdot\left(\frac{1}{\ln\frac{2y}{x} + 1} - \frac{2y}{x}\right) \\
&= h\left(\frac{2y}{x}\right)
\end{aligned}$$

Therefore, the sign of $\frac{d}{dx}\phi(x,y(x))$ is the same as the sign of $h\left(\frac{2y}{x}\right) - h\left(\frac{2\tilde{y}}{\tilde{x}}\right) = h\left(\frac{2y}{x}\right) - h\left(\frac{2(n-y)}{m-x}\right)$. Since $h(t)$ is a monotonically decreasing function the lemma will follow once we prove the following claim.

Claim 4.2. *For every $1 < x < m-1$ on the manifold $\phi(x,y) = \phi(\tilde{x},\tilde{y})$, $\frac{2y}{x} > \frac{2\tilde{y}}{\tilde{x}}$ if and only if $x < m/2$.*

Proof. Assume $1 < x < m/2$ or, equivalently, $1 < x < \tilde{x}$ (the case $m/2 < x < m$ is symmetric). Since $f(t) = \log t$ is a monotonically increasing function, $\log x < \log\tilde{x}$. Moreover, since $\phi(x,y) = \phi(\tilde{x},\tilde{y})$, we know that:

$$\begin{aligned}
\frac{2y}{x}\log\frac{2y}{x} &= \phi(x,y) - \log x \\
&> \phi(\tilde{x},\tilde{y}) - \log\tilde{x} = \frac{2\tilde{y}}{\tilde{x}}\log\frac{2\tilde{y}}{\tilde{x}}
\end{aligned}$$

And the claim follows because the function $f(t) = t\log t$ is monotonically increasing. ∎

This completes the proof of Lemma 4.5. ∎

Lemma 4.6. (Serial Decomposition Condition) *For every m and n, such that $2 \le m \le 2n \le m\log m$, and every x, y, and $\tilde{x}$, such that $1 \le y \le n-1$, $2 \le x \le \min\{2y, m\}$, $\max\{2, m-x\} \le \tilde{x} \le \min\{2(n-y), m\}$:*

$$\phi(x,y) + \phi(\tilde{x}, n-y) \ge \phi(m,n).$$

Proof. For $x = \tilde{x} = m$ and $y = n - y = n/2$ we have

$$\begin{aligned}\phi(x,y) + \phi(\tilde{x}, n-y) &= 4 + 2\log m + \frac{2n}{m}\log\frac{2n}{2m} \\ &= \left(2 + \log m + \frac{2n}{m}\log\frac{2n}{m}\right) + \left(2 + \log m - \frac{2n}{m}\right) \\ &\geq \phi(m,n),\end{aligned}$$

where the inequality follows from the constraint $2n \leq m\log m$.

It remains to show that no other x and y yield a lower value. By Lemma 4.4 we know that $D_1\phi(x_1,x_2) < 0$ for any $1 < x_1 \leq 2x_2$, i.e., the function $\phi(x_1,x_2)$ is strictly decreasing with x_1 and is minimized when $x_1 = x_{max}$, the largest value that the first parameter can take. Let $x_{max} = \min\{2y, m\}$ and $\tilde{x}_{max} = \min\{2(n-y), m\}$. Then

$$\phi(x,y) + \phi(\tilde{x}, n-y) \geq \phi(x_{max}, y) + \phi(\tilde{x}_{max}, n-y)$$

Without loss of generality, let us assume $2y \leq 2(n-y)$, i.e., $y \leq n/2$; the proof for the other case is symmetric. There are three cases to consider:

Case 1. $m \leq 2y \leq 2(n-y)$:

Then $x_{max} = \tilde{x}_{max} = m$ and we get

$$\begin{aligned}&\phi(x,y) + \phi(\tilde{x}, n-y) \\ &\quad\geq \phi(m,y) + \phi(m, n-y) \\ &\quad= \left(2 + \log m + \frac{2y}{m}\log\frac{2y}{m}\right) \\ &\qquad + \left(2 + \log m + \frac{2(n-y)}{m}\log\frac{2(n-y)}{m}\right) \\ &\quad= 4 + 2\log m + \frac{2}{m}(y\log y + (n-y)\log(n-y) + n) \\ &\qquad - \frac{2n}{m}\log m \\ &\quad\geq 4 + 2\log m + \frac{2}{m}(n\log(n/2) + n) - \frac{2n}{m}\log m \\ &\quad= 4 + 2\log m + \frac{2n}{m}\log\frac{n}{m} \\ &\quad= \left(2 + \log m + \frac{2n}{m}\log\frac{2n}{m}\right) + \left(2 + \log m - \frac{2n}{m}\right) \\ &\quad\geq \phi(m,n),\end{aligned}$$

where, the second inequality follows from the fact that function $f(t) = t\log t + (n-t)\log(n-t)$ is minimized at $t = n/2$ and the last inequality follows from the condition $2n \leq m\log m$.

Case 2. $2y \leq m \leq 2(n-y)$:

Then $x_{max} = 2y$ and $\tilde{x}_{max} = m$ and we get

$$\begin{aligned}&\phi(x,y) + \phi(\tilde{x}, n-y) \\ &\qquad\geq \phi(2y,y) + \phi(m, n-y) \\ &\qquad= 2 + \log(2y) + \phi(m, n-y)\end{aligned}$$

Observe that the derivative of $\phi(x_1,x_2)$ with respect to the second argument, $D_2\phi(x_1,x_2) = \frac{\tau(x_1,x_2)}{x_2} + \frac{2}{x_1\ln 2} = \frac{2\log(2ex_2/x_1)}{x_1}$, is a monotonically increasing function with respect to the second argument. Therefore, it achieves its maximum value, let's call it D_2^{max}, at the largest x_2 of the parameter range, i.e., $x_2 = n$ and $D_2^{max} = D_2\phi(m,n) = \frac{2\log(2en/m)}{m}$. Then for any non-negative $\Delta \leq n-1$:

$$\phi(m, n-\Delta) + \Delta \cdot D_2^{max} \geq \phi(m,n)$$

Thus, to prove that $\phi(x,y) + \phi(\tilde{x}, n-y) \geq \phi(m,n)$, it is sufficient to prove that

$$1 + \log(2y) \geq y \cdot D_2^{max} = y \cdot \frac{2\log(2en/m)}{m}$$

or, equivalently, that $\frac{1+\log(2y)}{y} \geq \frac{2\log(2en/m)}{m}$.

First, observe that $\frac{1+\log(2y)}{y}$ is minimized at the maximum value of y, i.e., at $y = m/2$. Thus,

$$\frac{1+\log(2y)}{y} \geq \frac{1+\log m}{m/2} = \frac{2\log(2m)}{m}$$

Since $2n \leq m\log m$ and $\log m \leq 2m/e$ for any $m \geq 2$, we get:

$$\frac{2\log(2m)}{m} \geq \frac{2\log(4n/\log m)}{m} \geq \frac{2\log(2en/m)}{m}$$

Case 3. $2y \leq 2(n-y) < m$:

In this case, $x_{max} = 2y$ and $\tilde{x}_{max} = 2(n-y)$. And we get

$$\begin{aligned}&\phi(x,y) + \phi(\tilde{x}, n-y) \\ &\qquad\geq \phi(2y,y) + \phi(2(n-y), n-y) \\ &\qquad= (2 + \log(2y)) + (2 + \log(2(n-y))) \\ &\qquad\geq 4 + \log 2n,\end{aligned}$$

where the last inequality follows from the fact that $\log a + \log b = \log(ab) \geq \log(a+b)$ for any $2 \leq a \leq b$.

Finally, since $2y \leq 2(n-y) < m$, we conclude that $2m > 2y + 2(n-y) = 2n$, i.e., $m > n$ and $\phi(n,m) < 2 + \log m + \frac{2n}{n}\log\frac{2n}{n} < 4 + \log m$. Thus,

$$\phi(x,y) + \phi(\tilde{x}, n-y) \geq 5 + \log m > \phi(m,n).$$

□

5 Conclusion

The main result of this paper is a lower bound of $\Omega(\log n \log\log n)$ for the span for implementing a depth-$O(\log n)$ FFT graph in the nonatomic binary fork-join model, which provides a separation result with

respect to the atomic binary fork-join model, for which simulating a depth-$O(\log n)$ FFT graph is easily done with $O(\log n)$ span. Our work should not be viewed as saying that the atomic version of the fork-join model is superior to the nonatomic version, or vice versa, however. Instead, our results show there is a clear trade-off: namely, that algorithms in the atomic fork-join model can achieve smaller spans, but this comes at a cost of requiring some kind of atomic memory primitive, such as test-and-set or compare-and-set.

References

[1] Yehuda Afek, Eli Gafni, John Tromp, and Paul MB Vitányi. Wait-free test-and-set. In *Proceedings of the International Workshop on Distributed Algorithms (WDAG)*, pages 85–94. Springer, 1992. doi:10.1007/3-540-56188-9_6.

[2] Kunal Agrawal, Charles E. Leiserson, and Jim Sukha. Helper locks for fork-join parallel programming. *ACM Sigplan Notices*, 45(5):245–256, 2010. doi:10.1145/1837853.1693487.

[3] Nimar S. Arora, Robert D. Blumofe, and C. Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. *Theory of Computing Systems*, 34(2):115–144, 2001. doi:10.1007/s00224-001-0004-z.

[4] Hans-Jürgen Bandelt and Victor Chepoi. Metric graph theory and geometry: a survey. In Jacob E. Goodman, János Pach, and Richard Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, volume 453 of *Contemporary Mathematics*, pages 49–86. AMS, 2008.

[5] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the ACM Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, 1968. doi:10.1145/1468075.1468121.

[6] Paul Beame and Johan Hastad. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*, 36(3):643–670, 1989. doi:10.1145/65950.65958.

[7] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Charles E. Leiserson. On-the-fly maintenance of series-parallel relationships in fork-join multithreaded programs. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 133–144, 2004. doi:10.1145/1007912.1007933.

[8] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Julian Shun. Internally deterministic parallel algorithms can be fast. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 181–192, 2012. doi:10.1145/2145816.2145840.

[9] Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 89–102, 2020. doi:10.1145/3350755.3400227.

[10] Guy E. Blelloch, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. The parallel persistent memory model. In *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 247–258, 2018. doi:10.1145/3210377.3210381.

[11] Guy E. Blelloch, Phillip B. Gibbons, and Yossi Matias. Provably efficient scheduling for languages with fine-grained parallelism. *Journal of the ACM*, 46(2):281–321, 1999. doi:10.1145/301970.301974.

[12] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 37(1):55–69, 1996. doi:10.1006/jpdc.1996.0107.

[13] Robert D. Blumofe and Charles E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, 1998. doi:10.1137/S0097539793259471.

[14] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999. doi:10.1145/324133.324234.

[15] Hans L. Bodlaender and Babette van Antwerpende Fluiter. Parallel algorithms for series parallel graphs and graphs with treewidth two. *Algorithmica*, 29(4):534–559, 2001. doi:10.1007/s004530010070.

[16] R. Cole and V. Ramachandran. Efficient resource oblivious algorithms for multicores with false sharing. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 201–214, May 2012. doi:10.1109/IPDPS.2012.28.

[17] R. Cole and V. Ramachandran. Analysis of randomized work stealing with false sharing. In *Proceedigns of the 27th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 985–998, May 2013. doi:10.1109/IPDPS.2013.86.

[18] Richard Cole and Michael T. Goodrich. Optimal parallel algorithms for point-set and polygon problems. *Algorithmica*, 7(1):3–23, 1992. doi:10.1007/BF01758749.

[19] Richard Cole and Vijaya Ramachandran. Resource oblivious sorting on multicores. *ACM Trans. Parallel Comput.*, 3(4), 2017. doi:10.1145/3040221.

[20] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. doi:10.2307/2003354.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 3rd edition, 2009.

[22] Leonardo Dagum and Ramesh Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. doi:10.1109/99.660313.

[23] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 393–404. ACM, 2018. doi:10.1145/3210377.3210414.

[24] G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 436–441, 1989. doi:10.1109/SFCS.1989.63515.

[25] Andrzej Duda and Tadeusz Czachórski. Performance evaluation of fork and join synchronization primitives. *Acta Informatica*, 24(5):525–553, 1987.

[26] David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992. doi:10.1016/0890-5401(92)90041-D.

[27] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-Oblivious Algorithms. *ACM Transactions on Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

[28] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the Cilk-5 multithreaded language. *ACM SIGPLAN Notices*, 33(5):212–223, May 1998. doi:10.1145/277652.277725.

[29] Michael T. Goodrich. Using approximation algorithms to design parallel algorithms that may ignore processor allocation. In *Proceedings of the 32nd IEEE Symposium of Foundations of Computer Science (FOCS)*, pages 711–722. IEEE, 1991. doi:10.1109/SFCS.1991.185439.

[30] Michael T Goodrich, Yossi Matias, and Uzi Vishkin. Optimal parallel approximation for prefix sums and integer sorting. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 241–250, 1994.

[31] Torben Hagerup and Christine Rüb. Optimal merging and sorting on the EREW PRAM. *Information Processing Letters*, 33(4):181–185, 1989. doi:10.1016/0020-0190(89)90138-5.

[32] Maurice Herlihy and Victor Luchangco. Distributed computing and the multicore revolution. *ACM SIGACT News*, 39(1):62–72, 2008. doi:10.1145/1360443.1360458.

[33] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[34] Richard M. Karp and Vijaya Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 869–941. Elsevier, 1990. doi:10.1016/B978-0-444-88071-0.50022-9.

[35] Doug Lea. A Java fork/join framework. In *Proceedings of the ACM Conference on Java Grande*, pages 36–43, 2000. doi:10.1145/337449.337465.

[36] Charles E. Leiserson. Cilk. In David Padua, editor, *Encyclopedia of Parallel Computing*, pages 273–288. Springer US, Boston, MA, 2011. doi:10.1007/978-0-387-09766-4_289.

[37] Xavier Martorell, Eduard Ayguadé, Nacho Navarro, Julita Corbalán, Marc González, and Jesús Labarta. Thread fork/join techniques for multi-level parallelism exploitation in NUMA multiprocessors. In *Proceedings of the 13th International Conference on Supercomputing (ICS)*, pages 294–301, 1999. doi:10.1145/305138.305206.

[38] John Mellor-Crummey. On-the-fly detection of data races for programs with nested fork-join parallelism. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*, pages 24–33. IEEE, 1991. doi:10.1145/125826.125861.

A Derivation of Equations (4.2)-(4.3)

Deriving partial derivatives in Equations (4.2)-(4.3) is a simple application of Calculus. However, we present the details of the derivation here for completeness and to help the reviewers verify the details if they so wish.

$$\begin{aligned}
D_1\tau(x,y) =& \frac{\partial}{\partial x}\left(\frac{2y}{x}\log\frac{2y}{x}\right)\\
=& \frac{d}{dx}\frac{2y}{x}\cdot\log\frac{2y}{x}+\frac{2y}{x}\cdot\frac{d}{dx}\log\frac{2y}{x}\\
=& -\frac{2y}{x^2}\log\frac{2y}{x}+\frac{2y}{x}\cdot\frac{1}{\frac{2y}{x}\ln 2}\cdot\frac{d}{dx}\frac{2y}{x}\\
=& -\left(\frac{\tau(x,y)}{x}+\frac{2y}{x^2\ln 2}\right)\\
D_2\tau(x,y) =& \frac{\partial}{\partial y}\left(\frac{2y}{x}\log\frac{2y}{x}\right)\\
=& \frac{d}{dy}\frac{2y}{x}\cdot\log\frac{2y}{x}+\frac{2y}{x}\cdot\frac{d}{dy}\log\frac{2y}{x}\\
=& \frac{2}{x}\log\frac{2y}{x}+\frac{2y}{x}\cdot\frac{1}{\frac{2y}{x}\ln 2}\cdot\frac{d}{dy}\frac{2y}{x}\\
=& \frac{\tau(x,y)}{y}+\frac{2}{x\ln 2}
\end{aligned}$$

$$\begin{aligned}
D_1\phi(x,y)=\frac{\partial\phi(x,y)}{\partial x} &= \frac{d}{dx}\log x + D_1\tau\\
&= \frac{1}{x\ln 2}-\left(\frac{\tau(x,y)}{x}+\frac{2y}{x^2\ln 2}\right)\\
&= -\left(\frac{\tau(x,y)}{x}+\frac{2y-x}{x^2\ln 2}\right)\\
D_2\phi(x,y)=\frac{\partial\phi(x,y)}{\partial y} &= \frac{d}{dy}\log x + D_2\tau\\
&= \frac{\tau(x,y)}{y}+\frac{2}{x\ln 2}
\end{aligned}$$