SYSTEMATIC METHODS FOR TREE BASED PARALLEL ALGORITHM DEVELOPMENT*

Gary L.Miller Shang-Hua Teng

Department of Computer Science
University of Southern California
Los Angeles, California 90089

ABSTRACT

In this paper, we consider dynamic parallel evaluation of computation trees[1], and give systematic methods via the closure properties of certain classes of unary functions for the construction of processor efficient polylogarithmic parallel algorithms for dynamic evaluation of computation trees. We present optimal parallel algorithms for many computation tree problems which are important in algebraic parallel computation, numerical parallel computation and parallel code generation on Exclusive Read and Exclusive Write PRAMs. We define the dynamic tree complexity (random as well as deterministic) and present many new techniques for prudent evaluation, dynamic processor scheduling and load balancing.

1 Introduction

Computation tree evaluation plays an important roles in many computational problems, such as tree isomorphism testing, dynamic expression evaluation [16], optimal arithmetic code parallel generation [10], parallel message decoding [28], connected component finding [14], graph planarity testing [15]. In [16], Miller and Reif gave a parallel algorithm for tree contraction which takes $O(\log n)$ times and uses $O(n)$ processors deterministically and use only $O(n/\log n)$ processors randomly on Concurrent Read and Concurrent Write (CRCW) model parallel computational system. This is done by applying two operations: RAKE and COMPRESS simultaneously on a tree, and each phase reduces the tree size by a constant factor. By using new load balancing and isolation techniques, Gazit, Miller and Teng [12] presented an optimal parallel algorithm to do tree contraction in $O(\log n)$ time and $O(n/\log n)$ processors on EREW model parallel system randomly as well as deterministically. However, the generic parallel tree contraction algorithm can be applied to more general problems only if the RAKE and COMPRESS operations can be supported uniformly. For example, what is the parallel complexity of min-max-plus-times-division tree? We will give answers to this problems and many other problems in this paper.

1.1 The Problem

Let D be a set of constants (finite or infinite), called domain. Let OP be a finite set of operators on D. Let $\mathcal{F}$ be a family of unary functions over domain D. We define a tree as used in this paper to be a rooted, ordered tree with pointers from children to parent.

Definition 1.1 (Simple computation tree) *A simple computation tree is a 3-tuple $\{D, OP, T\}$ where T is a tree whose leaves are labeled with a value in D, whose internal nodes are labeled with operators in OP.*

We generalize the concept of simple computation tree by assigning a unary function from some given set to each edge of tree. We will show this extension provides an elegant way to establish systematic scheme for the parallel tree-based computation.

Definition 1.2 (Computation tree) *A computation tree is a 4-tuple $\{D, OP, \mathcal{F}, T\}$ where T is a tree whose leaves are labeled with a value in D, whose internal nodes are labeled with operators in OP and whose edges are labeled with functions in $\mathcal{F}$. The leaves of a tree is called the inputs of the tree. The size of T is the number of nodes.*

Definition 1.3 (Value of node) *The value of a node in a general computation tree (denoted by $\underline{vl}(v)$) is defined inductively:*

- *If v is a leaf, then $\underline{vl}(v)$ equals to its label.*

*This work was supported in part by National Science Foundation Grant DCR-85-14961

[1] A computation tree is a 4-tuple $\{D, OP, \mathcal{F}, T\}$ where T is a tree whose leaves are labeled with a value in D, whose internal nodes are labeled with operators in OP and whose edges are labeled with unary functions in $\mathcal{F}$.

- *If v is an internal node, labeled by $\odot$ and with children $w_1, \ldots, w_l$, then:*

$$\underline{vl}(v) = \odot(f_1(\underline{vl}(w_1)), \ldots, f_l(\underline{vl}(w_l)))$$

Where $f_1, \ldots, f_k$ are the unary functions for edges $(w_1, v), \ldots, (w_k, v)$ respectively.

Definition 1.4 (Computation tree evaluation) *By computation tree evaluation, we mean for any given computation tree, compute the value of all its nodes.*

Example 1.1 (Arithmetic tree) *An arithmetic tree is a computation tree with domain the set $\mathcal{R}$ of all real numbers (or any field) and over operator set $OP = \{+, -, \times, \div\}$. This is a tree form representation of arithmetic expression. The family of edge functions is the set of linear fraction functions, e.g:*

$$\mathcal{F} = \{\frac{a * x + b}{c * x + d} \mid a, b, c, d \in \mathcal{D}\}$$

1.2 Parallel Complexities of Some Computation Trees

- **Arithmetic Tree:** Any arithmetic tree of size n can be evaluated in $O(\log n)$ time, using $O(n/\log n)$ processors on EREW PRAM. This was proved by Miller and Reif in [16].
- **Min-max-plus-times-tree:** min-max-plus-times-tree is a computation tree over domain $\mathcal{R}$, the set of real number, and operator set $\{min, max, + \times\}$. We will show that any min-max-plus-times-tree of size $n$can be evaluated in $O(\log n)$ time, using $O(n/\log n)$ processors.
- **min-max-plus-times-division-tree:** We will show that any min-max-plus-times-division-tree over domain $\mathcal{R}^+$ of size n can be evaluated in $O(\log n)$ time, using $O(n/\log n)$ processors. While any min-max-plus-times-division-tree over domain $\mathcal{R}$ can be evaluated in $O(\log^2 n)$ time, using $O(n)$ processors.

Remark: We use the uniform cost criterion in this paper.

1.3 Some Widely Used Techniques for Fast Parallel Algorithm Designing

The primitive methods to develop fast parallel algorithms are forwarding, doubling and load balancing. Forwarding and doubling are techniques for reasonable partial computation. These techniques extend the eager evaluation idea in dataflow models [27] and have been used in some previous papers for tree contraction [16], list ranking [7], message decoding [28], arithmetic evaluation [17], [31], circuit computation [18], logic program querying [29] and CFLs' parallel recognition [5], [21], [19], [11]. For the formal definition of forwarding, general tree forwarding, doubling, see [18].

1.4 Unary Functions and Closure Properties

The role of unary functions and their closure properties in parallel computation was first studied in [18], where Miller and Teng studied the closure properties and uniformity of unary functions over a given domain D and operator set OP, which is suffice to support fast parallel circuit evaluation. Tree is a special kind of circuit in which each node has at most one parent. This property gives tree its own features for its dynamic parallel evaluation and algorithm generation.

Computation tree is decided by its tree structure, domain, operator set and its unary function class. It is clear that the existence of fast parallel algorithm for computation tree dynamic parallel evaluation, (even the parallel algorithm itself) depends on those four parameters heavily. By a fast parallel algorithm we mean an algorithm that takes logarithm time and uses only a polynomial number of processors. The goal of this paper is to study the necessary and sufficient conditions that the parameters should satisfy for the existence of fast parallel tree evaluation algorithm and and show how to generate the algorithm systematicly. The following are two important closure properties for bounded degree tree evaluation.

Definition 1.5 (Composition Closure Property) *Let $\mathcal{F}$ be a family of unary functions over domain D. $\mathcal{F}$ is closed under composition if $\forall\ f_1, f_2 \in \mathcal{F}$, $\exists\ f \in \mathcal{F}$, such that $\forall\ x \in D$: $f(x) = f_2(f_1(x))$ denoted by $f_2 \circ f_1 = f$.*

Definition 1.6 (Forwarding Closure Property) *Let $\mathcal{F}$ be a family of unary functions over domain D, OP be an operators set over D, $\mathcal{F}$ is closed under forwarding over OP if for all $a_1, \ldots, a_l \in D$, $\odot \in OP, f \in \mathcal{F}$, for all $i : 1 \leq i \leq l$:*

$$f(\odot(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_l)) \in \mathcal{F}$$

Definition 1.7 (Projective property) *A family of unary functions is closed under projection over operator set OP if for all $\odot \in OP$, for all $a_1, \ldots, a_l \in D$, for all $i : 1 \leq i \leq l$:*

$$\odot(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_l) \in \mathcal{F}$$

Proposition 1.1 *If a family of unary functions $\mathcal{F}$ is closed under composition, and projectively closed over OP, then $\mathcal{F}$ is closed under forwarding over OP. Moreover, if $\mathcal{F}$ contains the identical function, then $\mathcal{F}$ is closed under forwarding over OP iff $\mathcal{F}$ is projectively closed over OP.*

Definition 1.8 (Minimumly closed) *A family of unary functions $\mathcal{F}$ is minimumly closed over operator set OP, iff $\mathcal{F}$ is closed under composition and forwarding over OP and*

no proper subset of $\mathcal{F}$ has those properties.

Claim 1.1 *A family of unary functions $\mathcal{F}$, which is closed under composition and tree forwarding, is minimum iff $\forall f \in \mathcal{F}$, there is an OP-tree with one leaf labeled as variable which generates f.*

For simple computation tree with operator set OP, the family of unary functions which is minimumly closed over OP plays an important role in its parallel computation. So it is important theoretically as well as practically to find a systematic method for generating the minimumly closed set of unary functions over a given operator set. We will present a powerful method for automatic construction of minimumly closed family of unary functions over a large class of operator set in section 3.

Example 1.2 *Let $\mathcal{F}$ be the class of linear fraction over $\mathcal{R}$, then $\mathcal{F}$ is minimumly closed under composition and forwarding over $\{+,-,\times,\div\}$.*

[PROOF] Let $f_1(x) = \frac{a_1x+b_1}{c_1x+d_1}, f_2(x) = \frac{a_2x+b_2}{c_2x+d_2}$,

- $\mathcal{F}$ is closed under composition since for all $a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2 \in \mathcal{D}$,

$$f_2 \circ f_1(x) = f_2(f_1(x)) = \frac{a_3x + b_3}{c_3x + d_3} \in \mathcal{F}$$

  Where $a_3 = a_1a_2 + b_3c_1, b_3 = a_2b_1 + b_2d_1$, $c_3 = a_1c_2 + c_1d_2, d_3 = b_1c_2 + d_1d_2$.
- $\mathcal{F}$ is closed under forwarding over $\{+,-,\times,\div\}$ since $\mathcal{F}$ is closed under composition and $f_{+a}(x) = x + a, f_{-a}(x) = a - x, f_{\times a}(x) = x * a, f_{\div a}(x) = a/x \in \mathcal{F}$.
- $\mathcal{F}$ is minimumly closed over $\{+,-,\times,\div\}$ since for all $a, b, c, d \in \mathcal{D}$:

$$\frac{ax + b}{cx + d} = \frac{a}{c} + \frac{bc - ad}{c^2x + cd}$$

The closure properties of linear fraction is given in [16]. We use it here for the sake of illustration.

1.5 The Main Theorem

In this paper, we study the dynamic tree complexity under parallel manipulations and give a systematic method for computation tree parallel evaluation. We show that the existence of a family of uniform unary functions which is uniformly closed under composition and forwarding are sufficient conditions for computation tree fast parallel evaluation on Exclusive Read and Exclusive Write model system. We also study the dynamic random parallel tree complexity and some techniques for load balancing and dynamic processor scheduling. The following is the main theorem of this paper.

Theorem 1.1 (The first main theorem) *Let $\mathcal{T}$ be a tree over domain $\mathcal{D}$ and operator set OP with edge function set $\mathcal{F}$. If $\exists \mathcal{F}_1 \supseteq \mathcal{F}$ such that $\mathcal{F}_1$ is closed under composition, forwarding over OP and the maximum time and processor resources required to compute composition and forwarding are bounded by $T(n)$ and $P(n)$ respectively, then $\mathcal{T}$ can be evaluated in $O(T(n)*\log n)$ time, using $O(P(n)*n/\log n)$ processors deterministically on any EREW shared memory machine. Where n is the tree size of $\mathcal{T}$.*

The main theorem provide a powerful tool for proving a given tree evaluation problem lies in $\mathcal{NC}$, the class of problem can be solved in polylogarithmic time using polynomial number of processors.

Corollary 1.1 (Simple tree computation) *Let $\mathcal{T}$ be a simple computation tree over domain $\mathcal{D}$ and operator set OP. If there is a family of unary functions which is closed under composition, forwarding over OP and the maximum time, processors required to computer composition and forwarding are bounded by $T(n)$ and $P(n)$ respectively, then $\mathcal{T}$ can be evaluated in $O(T(n) * \log n)$ time, using $O(P(n) * n/\log n)$ processors.*

1.6 Major Results Of This Paper

- There is a systematic method via the closure properties of certain classes of unary functions for construction processor efficient polylogarithmic parallel algorithms for dynamic evaluation of computation trees.
- **Reduction Lemma:** If a family of unary functions $\mathcal{F}_1$ is minimumly closed under composition and tree forwarding over operator set OP_1, and all functions $L(x) \in \mathcal{F}_1$ are monotonic, then $\mathcal{F} = \{min[a, max(L(x), b)] \mid L(x) \in \mathcal{F}_1, a, b \in \mathcal{D}\}$ is minimumly closed under composition, tree forwarding over $OP_1 \cup \{min, max\}$.
- **Unformly Additively Bounded Theorem:** For any computation tree, if its edge functions is uniformly additively bounded, the time and processors count used to compute composition and forwarding operation on any functions f, g with size s, t is bounded by $Ti(s+t)$, $P(s+t)$ and $Ti(s), P(s)$ respectively, then $\mathcal{T}$ can be computed in $O(Ti(n)*\log n)$ time, use $O(max(P(n), n))$ processors in EREW model.
- Dynamic tree complexity is much smaller than static tree complexity in general case.
- As an application of the above general theorem, we show that min-max-plus-times-division tree over $\mathcal{R}$ can be evaluated in $O(\log^2 n)$ time using $O(n)$ processors, while min-max-plus-times-division tree over $\mathcal{R}^+$ can be evaluated in $O(\log n)$ time using $O(n/\log n)$ processors deterministically (or using randomization with a smaller constant) in EREW parallel random access model.

2 Theoretical Foundation for Tree-Based Parallel Algorithm Development

In this section, we will give a parallel algorithm for tree computation and prove our main theorem. The algorithm presented in this section serves as the template for the systematic design of many tree based parallel algorithms.

2.1 Tree Contraction

Tree contraction is a bottom-up procedure to manipulate a tree and reduce it to its root. Miller and Reif [16] presented an algorithm[2] which takes $O(\log n)$ time, uses $O(n)$ processors deterministically and $O(n/\log n)$ processors randomly. Tree contraction is done by applying two abstracted operations: RAKE and COMPRESS in a pipeline way, where RAKE removes all the leaves of the tree and COMPRESS reduces each chaining by doubling. (pointer jumping). The following is a tree contraction algorithm given in [16].

ALGORITHM Tree Contraction
begin
while tree is not a single point **do**
In parallel for all node v in tree **do**
RAKE: **if** v is a leaf **then delete** v.
COMPRESS: **if** v has only one child
forward its child to its father
end

Theorem 2.1 (Tree contraction) *Tree contraction can be done in $O(\log n)$ time using $O(n/\log n)$ processors on EREW PRAM deterministically.*

2.2 Operations for Computation Tree Evaluation

We now define some operations which will be applied to a computation tree to compute the values of all its nodes.

- Clean edge:
  For all leaves v, let f be the edge function associated with the edge from v to its father $father(v)$:
  Compute $f(vl(v))$, label v by $f(vl(v))$ and $(v, father(v))$ by an identical function.

- Make leaf:
  For all nodes v in the tree whose children are leaves, do
  $vl(v) \leftarrow \odot(vl(v_1), \ldots, vl(v_k))$
  Where $v_1, \ldots, v_k$ are the children of v and $\odot$ is the operator of v.
  delete edge (v_i, v) and make v be a leaf.

- Tree forwarding:
  for all nodes v with l children in which all but ith children are are leaves. do
  find $f \in \mathcal{F}$: $f(x) = f_0(\odot(u_1, ..., u_{i-1}, x, ..., u_l))$
  Where $u_1, \ldots, u_l$ is the value of its children and f_0 is the edge function associated with edges $(v, father(v))$, (c_i means the ith children), $\odot$ is the operator label of v.
  For all $j \neq i$ delete edge (v, c_j)
  delete edge $(father(v), v)$.
  insert edge $(father(v), c_i)$.
  label $(father(v), c_i)$ by f.

- Isolation:
  isolate all the chain in the tree.

- Local compress
  Use LOCAL-COMPRESS[I] or LOCAL-COMPRESS[II] (see appendix) to compress the isolated chain locally. The choice of the above two procedure is up to the uniformity of the edge function which will be discussed later and the local compress is supported by composition closure property.

2.3 The General Algorithm

With the four operations defined in last subsection, the tree computation algorithm can be specified as following:

ALGORITHM tree computation
input computation tree $\mathcal{T}$;
output The values of all nodes in $\mathcal{T}$;
begin
repeat
clean edge; make leaf; tree forwarding;
clean edge; make leaf;
isolation; generate local compress process;
until the tree is reduced to its root;
run dynamic expansion procedure given in [16]
end.

Lemma 2.1 *At the end of the algorithm, the value associated with each node is the value of the node.*

[PROOF] It is straightforward consequence of the definition of composition, and tree forwarding properties, that clean edge, make leaf and tree forwarding support the RAKE operation. Thus this lemma is a consequence of the main theorem in [16] and [12]. □

[2] This algorithm was improved by Gazit, Miller and Teng [12], by using new load balancing techniques called SUPERRAKE, new list ranking algorithm [2], [7] and implementation techniques called ISOLATED COMPRESS on exclusive read and exclusive write model PRAMs, using $O(\log n)$ times and $O(n/\log n)$ processors deterministically as well as randomly. Two local compress procedures were given for different applications (see appendix), we will discuss their applications later in this paper. We will count the $\log n$ reduction of processor count in all our theorems without further discussion.

Claim 2.1 *Tree computation can be done in $O(T(n) * \log n)$ time, use $O(P(n) * n/\log n)$ processors in any EREW model parallel system.*

[PROOF] Since clean edge, make leaf, tree forwarding simulate the generic operation RAKE in $T(n)$ time, use $P(n)$ processors, and isolation and local compress are defined in the same way as those in [12]. Clearly this lemma (So is the first main theorem) is a consequence of the major theorems in [16] and [12]. □

2.4 General Methods

Methods for the design of processor-efficient and time-efficient parallel algorithms.

- **Input** a given problem P (on data D).
- Find the domain D of the problem and a set of operators OP over the domain.
- Transform the problem P into a computation tree problem $\mathcal{T}$ over the domain D and the operators set OP.
- Construct a set of unary functions $\mathcal{F}$ which satisfies the composition closure, combination closure, forwarding closure, and doubling closure properties.
- Construct a data structure to support the composition, combination, doubling, forwarding operations over the set of unary functions. Moreover, the data structure must be powerful enough to compute the unary functions fast in parallel.
- Prove that using the above data structure, the composition, combination, doubling, forwarding operations over the unary functions and the evaluation of the unary functions can be done fast in parallel with reasonable hardware complexity.
- Transform the tree $\mathcal{T}$ into a computation tree with unary function in order to reduce the tree complexity of the problem. (optional).
- Prove that the above two transforming procedure can be done fast in parallel without reasonable hardware complexity.
- Show the uniformity of the unary function during the parallel tree evaluation. If the size unary functions are not constant, show the uniformly additively bounded properties of the function class (for definition, see section 5). If the unary function class are uniformly additively bounded, then apply the prudent evaluation technique (see section 5) and the dynamic processor scheduling technique (see section 5) to balance the working load of each processor. Therefore, reduce the processor count.
- Apply the general tree evaluation algorithm presented in this section to constructure the high level coding and using our main theorem to show the complexity.

3 Reduction Lemma

In this section, we give a useful lemma for edge function construction, proof of closure properties and proof of minimality for many function class.

We have to solve two problems when a simple computation tree over domain D and operator set OP is given. First, we have to construct a set of unary functions over D and then prove it is (minimumly) closed over OP. Of course, we also have to study the uniformity of the class of functions we constructed.

It would be nice if we have a general theory and systematic method for edge function construction, closure property proof and minimality proof. We observe that many function classes over operator sets (denoted by OP) containing min and max operators have an elegant form: $min\{a, max[L(x), b]\}$. Where $L(x)$ is a unary function closed under composition and forwarding over OP-$\{min, max\}$. Hence, we can study the properties of two smaller operator sets $\{min, max\}$ and $OP - \{min, max\}$ and the properties of unary function class $\{L(x)\}$. This is the motivation behind our reduction lemma. Let us see an example first. Another example will be given in section 6.

Example 3.1 (Min-max-plus-times-tree) *Min-max-plus-times tree is a computation tree over domain $\mathcal{R}$, the set of real number, and operator set $\{min, max, +, \times\}$.*

Based on the theory above, we have the following lemma. We will use reduction lemma to prove this lemma at end of next subsection.

Lemma 3.1 *The unary function class $\mathcal{F} = \{min\{a, max[L(x), b]\} \mid a, b \in \mathcal{R}, L(x)$ is a linear fraction$\}$ is minimumly closed under composition and forwarding over $\{min, max, +, \times\}$.*

The construction and proof in this subsection are on domain $D \subseteq \mathcal{R}$, a subset of real numbers. This can be extended to other rings or fields by their algebraic relation.

Lemma 3.2 (Duality lemma) *For all unary functions $L(x)$, for all $a, b \in D$, there exist $c, d \in D$ such that:*
(1) $min\{a, max[L(x), b]\} = max\{c, min[L(x), a]\}$
(2) $max\{a, min[L(x), b]\} = min\{d, max[L(x), a]\}$

[PROOF] The lemma follows by the distributivity of *min* over *max* and *max* over *min*. □

Claim 3.1 *For all unary functions $L(x)$ over $\mathcal{D}$, for all $a, b \in \mathcal{D}$:*

- *If $L(x)$ is monotonic increasing over $\mathcal{D}$, then:*
  $L(max[a,b]) = max[L(a), L(b)]$
  $L(min[a,b]) = min[L(a), L(b)]$
- *If $L(x)$ is monotonic decreasing over $\mathcal{D}$, then:*
  $L(max[a,b]) = min[L(a), L(b)]$
  $L(min[a,b]) = max[L(a), L(b)]$

Lemma 3.3 *If a family of unary functions $\mathcal{F}_1$ is closed under tree forwarding over operator set OP_1, and all functions $L(x) \in \mathcal{F}_1$ are monotonic, then $\mathcal{F} = \{min[a, max(L(x), b)] \mid L(x) \in \mathcal{F}_1, a, b \in \mathcal{D}\}$ is closed under tree forwarding over $OP_1 \cup \{max, min\}$.*

[**PROOF**] For all $L(x) \in \mathcal{F}_1$, for all $a, b, c, d_1, \ldots, d_l \in \mathcal{D}$, for all $\odot \in OP_1$:

- Since $\mathcal{F}_1$ is closed under tree forwarding over OP_1, there exists $L_\odot(x) \in \mathcal{F}_1$ such that, $L_c(x) = L(\odot(d_1, ..., x, ...d_l))$, hence:

  $$min\{a, max[L(x \odot c), b]\} = min\{a, max[L_c(x), b]\}$$

- If $L(x)$ is monotonic increasing over $\mathcal{D}$, then:

  $$min\{a, max[L(max(c, x)), b]\} =$$
  $$min\{a, max[L(x), max(L(c), b)]\} \in \mathcal{F}$$

  By duality lemma, we have:

  $$min\{a, max[L(min(c, x)), b]\} \in \mathcal{F}$$

- If $L(x)$ is monotonic decreasing over $\mathcal{D}$, then:

  $$min\{a, max[L(min(c, x)), b]\} =$$
  $$min\{a, max[L(x), max(L(c), b)]\} \in \mathcal{F}$$

  By duality lemma, we have:

  $$min\{a, max[L(max(c, x)), b]\} \in \mathcal{F}$$

□

Lemma 3.4 *If a family of unary functions $\mathcal{F}_1$ is closed under composition, and all functions $L(x) \in \mathcal{F}_1$ are monotonic, then $\mathcal{F} = \{min[a, max(L(x), b)] \mid L(x) \in \mathcal{F}_1, a, b \in \mathcal{D}\}$ is closed under composition.*

[**PROOF**] For all $L(x), L'(x) \in \mathcal{F}_1$, for all $a, b, a', b' \in \mathcal{D}$, since $\mathcal{F}_1$ is closed under composition, there exists $L''(x) = L \circ L'(x)$.

- If $L(x)$ is monotonic increasing, then:

  $$\begin{aligned} & min\{a, max[L(min\{a', max[L'(x), b']\})]\} \\ = \; & min\{a, max[L(max\{b'_1, min[L'(x), a']\}), b]\} \\ = \; & min\{a, max[L(min[L'(x), a']), b_1]\} \\ = \; & max\{b_2, min[L(min[L'(x), a']), a]\} \\ = \; & max\{b_2, min[L(L'(x)), a_1]\} \\ = \; & min\{a_2, max[L''(x), b_2]\} \in \mathcal{F} \end{aligned}$$

- If $L(x)$ is monotonic decreasing, the by duality lemma, we have:

  $$min\{a, max[L(min\{a', max[L'(x), b']\})]\} \in \mathcal{F}$$

□

Lemma 3.5 (Reduction Lemma) *If a family of unary functions $\mathcal{F}_1$ is minimumly closed under composition and tree forwarding over operator set OP_1, and all functions $L(x) \in \mathcal{F}_1$ are monotonic, then $\mathcal{F} = \{min[a, max(L(x), b)] \mid L(x) \in \mathcal{F}_1, a, b \in \mathcal{D}\}$ is minimumly closed under composition and tree forwarding over $OP_1 \cup \{min, max\}$.*

[**PROOF**] Follow the above two lemmas, it is suffice to prove $\mathcal{F}$ is minimum. For all $L(x) \in \mathcal{F}_1, a, b \in \mathcal{D}$, since $\mathcal{F}_1$ is minimum, there exists a computation tree T_1 over OP_1 generates $L(x)$. Clearly, the tree in figure 2 generates $min\{a, max[L(x), b]\}$. □

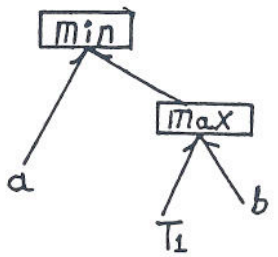

Figure 2: Tree for $min\{a, max[L(x), b]\}$

Claim 3.2 *The unary function class $\mathcal{F} = \{min\{a, max[L(x), b]\} \mid a, b \in \mathcal{R}, L(x)$ is a linear function$\}$ is minimumly closed under composition and forwarding over $\{min, max, +, \times\}$.*

[**PROOF**] Since $\mathcal{F}_1 = \{cx + d \mid c, d \in \mathcal{R}\}$, the set of linear functions are minimumly closed under composition $\{+, \times\}$. Moreover, every function in $\mathcal{F}_1$ is monotonic. Thus, as a consequence of reduction lemma, $\mathcal{F}$ is minimumly closed under composition and forwarding over $\{min, max, +, \times\}$. □

Corollary 3.1 *Min-max-plus-times tree can be computed in $O(\log n)$ deterministic parallel time and using $O(n/\log n)$ processors.*

Reduction lemma is beautiful in the sense that we can reduce the construction, closure properties proof and minimality proof of a given operator set which contains min, max

4 ⊕-tree, min-max-⊕-tree

Definition 4.1 (⊕-tree) *Let $\mathcal{T}$ an arbitrary computation tree over domain integer Z and operator set $\{\oplus\}$ which is defined as following: $\oplus : Z \times Z \to Z$ and for all $a, b \in Z$:*

$$\oplus(a,b) = \begin{cases} a+1 & a = b \\ max(a,b) & otherwise \end{cases}$$

This computation tree is very important in code automatic generation and optimization. (see [23] and [10]). In fact, it is easy to convert a expression tree to ⊕-tree by assign its left leaves the value 1 and right leaves the value 0, and label all internal nodes ⊕, which is a computation tree for label function defined in [23]. An $O(\log n)$ algorithm was given in [10] where their computation rule is complicated. Here we give an elegant proof of the existence of $O(\log n)$ parallel algorithm for ⊕-tree computation.

Let $\mathcal{F} = \{f_{(a,c)} \mid a \in Z \cup \{-\infty\}, c \in Z\}$ be the following set of unary functions: Where $f_{(a,c)}$ is defined as follows:

$$f_{(a,c)}(x) = \begin{cases} c & x < a \\ c+1 & a \le x \le c+1 \\ x & x > c+1 \end{cases}$$

Claim 4.1 *$\mathcal{F}$ is minimumly closed under composition and forwarding over ⊕.*

[PROOF] For all $f = f_{(a,c)}, g = f_{(a',c')}$:

- $\mathcal{F}$ is closed under composition, since:

$$(g \circ f)(x) = \begin{cases} g(c) & x < a \\ g(c+1) & a \le x \le c+1 \\ g(x) & x > c+1 \end{cases}$$
$$= \begin{cases} f_{(a',c')} & c < a' - 1 \\ f_{(a,c')} & c = a' - 1 \\ f_{(-\infty,c'+1)} & a' \le c < c'+1 \\ f_{(a,c)} & c \ge c'+1 \end{cases}$$

- $\mathcal{F}$ is minimumly closed over ⊕ since for all $a \in Z$, $f_{a,a} = \oplus(a,x)$ and for all $a, c \in \mathcal{D}$, $c \ge a$:

$$f_{(a,c)}(x) = (f_{(c,c)} \circ f_{(c-1,c-1)} \circ \ldots \circ f_{(a+1,a+1)} \circ f_{(a,a)})(x)$$

Therefore, all functions in $\mathcal{F}$ can be generated by an ⊕-tree.

Corollary 4.1 (Complexity of ⊕-tree) *Any ⊕-tree of size n (thus the label function) can be computed in time $O(\log n)$, using $O(n/\log n)$ processors.*

Since for all $a \in Z$, $max(a,x) = f_{(-\infty,a)}$, we get the following corollary for free.

Corollary 4.2 (Complexity of max-⊕-tree) *Any max-⊕-tree of size n can be evaluated in $O(\log n)$ time, with $O(n/\log n)$ processors.*

It is clear that $\mathcal{F} = \{f_{(a,c)} \mid a \in Z \cup \{-\infty\}, c \in Z\}$ is monotonic, thus as a consequence of reduction lemma, we have the following lemma and corollary.

Lemma 4.1 *The unary function class $\{min\{a, max[L(x), b]\} \mid a, b \in Z, L(x) \in \mathcal{F}\}$ is minimumly closed under composition and forwarding over $\{min, max, \oplus\}$.*

Corollary 4.3 (Complexity of min-max-⊕-tree) *Any min-max-⊕-tree of size n can be computed in $O(\log n)$ time, with $O(n/\log n)$ processors.*

5 Prudent Evaluation and Dynamic Processor Scheduling

In this section, we will show that for many tree computation problems, processor count can be reduced by a factor of $min(n, p(n))$ through load balancing, an important technique to develop cheap but fast parallel algorithms. This is, in fact, a partition problem as well as a scheduling problem [13]. How can we make full use the hardware in parallel system? The one of the first important issues of parallel processing, had unfortunately been proved as a $NP-hard$ problem for the general case. Many heuristic methods were proposed for variety problems and for some special structured systems. We will, in this section, prove that if the family of the edge functions for a computation tree is uniformly additively bounded, then the processor count can be reduced to $O(max(P(n), n))$ through load balancing and new scheduling technique.

Definition 5.1 (Uniformly additively bounded) *A family of unary functions $\mathcal{F}$ is uniformly additively bounded if we can assign size to each function in $\mathcal{F}$ and:*
(1) The time to compose two function $f, g \in \mathcal{F}$ of size m, n respectively is $Ti(m+n)$ and use $P(m+n)$ processors. Moreover, $g \circ f$ has size at most $m+n$.
(2) For all $a_1, \ldots, a_l \in \mathcal{D}$, $\odot \in OP$, we can find a function $f \in \mathcal{F}$ in $Ti(m)$ time, use $P(m)$ processors. Such that: $f(x) = \odot(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_l)$ And f has size at most l.

Theorem 5.1 (The second main theorem) *For any computation tree, if its edge functions is uniformly additively bounded, the time and processors count used to compute composition and forwarding operation on any functions f, g with size s, t is bounded by $Ti(s+t)$, $P(s+t)$ and $Ti(s), P(s)$ respectively, then $\mathcal{T}$ can be computed in $O(Ti(n) * \log n)$ time, use $O(max(P(n), n))$ processors in EREW model.*

[PROOF] Since, the size of unary function on each edge is constant[3] at beginning, we can assign $max(P(n), n)/n$ pro-

[3] In order to have a reduction of $\log n$ on processor count, we will use SUPERRAKE technique in [12]. SUPERRAKE and the removal of the nodes not in V_2 will reduce the size of tree by factor $\log n$. However the size of unary functions associated with the edges in the tree might increase and not be constant. Fortunately, the uniformly additively bounded property tell us that the summation of the sizes of all the unary functions in the tree will not increase. Hence, we can use prefix sums to assign the processors to the node according to the size of unary function associated with the edge form the node to its father. The rest argument is same as in the following proof.

cessors to each node. We have the following two cases:

1. The size of all the functions in $\mathcal{F}$ is less than certain constant. It is clear that the theorem holds, since in this case $Ti(n), P(n)$ are constant.

2. The size functions in $\mathcal{F}$ is not constant. Since, at beginning, we have n nodes and we assign $max(P(n), n)/n$ processors to each node. For forwarding operation on v, we use the processor associated with v to compute, and assign its leaf children's processors to v. We use LOCAL-COMPRESS[II] to support local compress. Since rank of nodes in chain with length m can be computed in $O(\log m)$ time with $O(m)$ processors, this can be done by using one processors associated with each node in the chain. The advantages of LOCAL-COMPRESS[II] over LOCAL-COMPRESS[I] is that no useless chain is produced after each application of compress. Any two consecutive nodes u, v are combined by using the processors associated with those two nodes and then we assign all the processors of u, v to the resulted node. Following the instruction of this scheduling method and the uniformly additively bounded property of the edge function, it is easy to prove by induction that at any time, the number of processors associated with any node v is not less than the size of function associated with edge $(v, father(v))$. That ends the prove of the theorem. □

6 Min-max-plus-times-division Tree

Min-max-plus-times-division tree play an important role in algebraic computation. The algorithm presented in this section can be easily extended to any field in which any pair of elements are comparable. As you can see, that min-max-plus-times-division tree contains all the general algebraic operators. Minus is removed since it can be easily implemented by times and plus without increasing the magnitude of tree size. The introducing of min, max makes the arithmetic tree more general, but on the other hand, makes the computation more complicated. Is it possible to use rake and compress operations to evaluate the min-max-plus-times-division tree? We will give a positive answer using the closure properties of edge functions.

6.1 Min-max-plus-times-division Tree on $\mathcal{R}^+$

In this subsection, we restrict the domain of min-max-plus-times-division tree to $\mathcal{R}^+$, the positive real number set. This restriction make the min-max-plus-times-division tree computation surprisingly easier than the general min-max-plus-times-division tree computation. Since the operator set contains $\{min, max\}$, therefore, we can use the deduction lemma to made the following conjecture and to prove the following lemma.

Lemma 6.1 (Unary functions class) *The family of unary functions $\mathcal{F}$ for min-max-plus-times-division tree over $\mathcal{R}^+$ is defined as following:*

$$\mathcal{F} = \{min\{a, max[LF(x), b]\} \mid a, b \in \mathcal{R}^+\}$$

Where $LF(x)$ is a linear fraction over $\mathcal{R}^+$ defined as $LF(x) = (cx + d)/(ex + f)$, and $c, d, e, f, \in \mathcal{R}^+$.

Lemma 6.2 *$\mathcal{F}$ is minimumly closed under composition and forwarding over $\{min, max, +, \times, \div\}$.*

[PROOF] The family of linear fractions is minimumly closed under composition and forwarding over $\{+, \times, \div\}$ over domain $\mathcal{R}^+$ can be proved in the similar way as in claim 4.3. Every linear fractions is monotonic since all the constants occur in the function are positive. Therefore, by applying reduction lemma, we have $\mathcal{F}$ is minimumly closed under composition and forwarding over $\{min, max, +, \times, \div\}$. □

Because all the functions in $\mathcal{F}$ is finitely represented and the composition, forwarding operations can be computed in constant time sequentially. The following theorem is a directed consequence of the previous lemma and the first main theorem.

Theorem 6.1 *A min-max-plus-times-division tree over $\mathcal{R}^+$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors deterministically on EREW PRAMs.*

6.2 Unary Functions of Min-max-plus-times-division Tree

We will extend the domain of min-max-plus-times-division tree from $\mathcal{R}^+$ to $\mathcal{R}$. However, since linear fractions are not monotonic over $\mathcal{R}$, the set of real number, we can not apply reduction lemma directly. We start with a short discussion of linear fractions.

Let $Lf(x) = ax + b/cx + d$ be a linear fraction. It is not hard to see that Lf has a simple pole at $x = -d/c$: In the case that $c = 0$ we say that Lf has a pole at infinity. It is one-one if and only if $ad - bc \neq 0$ and in this case the derivative is either always positive or always negative. We shall call a partial function f **monotone increasing with respect to its pole p** for all points x and y for which f is defined if $f(x) \leq f(y)$ then either $x \leq y \leq p$, $p \leq x \leq y$, or $y \leq p \leq x$. In a similar way we define monotone decreasing and monotone with respect to a pole. In general a pole is simply a point where the function is undefined, possibly ∞. Note that the linear fractions are monotone with respect to their poles. We next show that the monotone property is preserved under composition. The proof will appear in the final paper.

Lemma 6.3 *If f and g are monotone with respect to poles p and q, respectively, then $f \circ g$ is monotone with respect to p', where p' is any point between $max\{x|g(x) \leq p\}$ and $min\{x|g(x) \geq p\}$.*

We will prove that the following class of unary functions are large enough to evaluate the min-max-plus-times-division trees. We will assume that a tree that requires a division by zero to evaluate a subtree need not be evaluated, we can return undefined. Thus, functions from the following class will contain intervals for which they are not defined, called **U-intervals**. We will also need to maintain points where the function is continuous but not necessarily differentiable, called **breakpoints**. We will denote these two types of closed intervals by I_i, and let $x < I_i$ denote the fact that x is less than all elements in I_i.

Definition 6.1 (Unary functions class) *Let F denote the class of functions of the form*

$$f_{(Lf,I_1,\ldots,I_n,F_0,\ldots,F_{n+1},p)} = \begin{cases} F_0(x) & x < I_1 \\ F_{n+1}(x) & x > I_n \\ F_i(x) & I_i < x < I_{i+1}, 1 \leq i < n \end{cases}$$

Which satisfy the following conditions:

1. *The I_i form a disjoint ordered set of closed intervals of two types: intervals where f is undefined, U-intervals, and single point intervals which are breakpoints of f.*
2. *Each F_i either agrees with Lf on its interval or it is a constant function.*
3. *The function f is monotone with respect to its pole p.*

Note that the functions in $\mathcal{F}$ consist of a sequence of line segments or curve segments (of a linear fraction). Between each of these segments is an U-interval or a breakpoint. The size of a function in $\mathcal{F}$ is the number of U-interval and breakpoints in it. In Figure 3 we give an example of such a function.

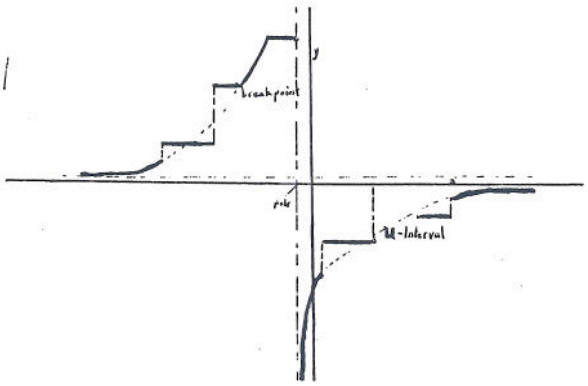

Figure 3: An Example of a Possible Unary Function for a Min-max-plus-times-division tree.

6.3 Closure Properties and Uniformly Additively Bounded Property

Claim 6.1 *$\mathcal{F}$ is closed under composition and projection over $\{min, max, +, \times, \div\}$.*

[PROOF] Let

$$f = f_{(Lf,I_1,\ldots,I_n,F_0,\ldots,F_{n+1},p)} \text{ and } g = g_{(Lf,I'_1,\ldots,I'_m,F'_0,\ldots,F'_{m+1},p')}$$

be two functions in $\mathcal{F}$. We must show that $\mathcal{F}$ is closed under composition, $f \circ g \in \mathcal{F}$ with some pole.

For the sake of succinctness and ease of understanding, we give the following informal argument. Since the functions f and g, where defined, are a sequence of horizontal line segments or segments of a fixed linear fraction, the composition $h = f \circ g$ must be, where defined, a collection of horizontal line segments and segments of the fixed linear fraction $Lf \circ Lf'$. To see this, observe that the composition of a linear function and constant is a constant, independent of the order. While, the composition of two linear fractions is a linear fraction. We must still describe the U-intervals and the breakpoints, and show that the composition is monotone with respect to its pole.

We start by computing the intervals for h. Clearly the intervals of g will be intervals of h. But the x in $\{x|g(x) \in I_i, 1 \leq i < n\}$ will also be in the interval of h. Note that if the preimage of a breakpoint p of f by g, i.e., $g_{-1}(p)$, is an interval then the breakpoint p will be contained in either an U-interval or a breakpoint of g. Thus breakpoints of h are singletons and not proper intervals. We have shown that h satisfies the first two condition of the class.

By the last lemma h is monotone with respect to some point q. Note that in general this point may not be the pole of $Lf \circ Lf'$. This is possible when the pole of h is not a "real" pole but just a point where h is undefined. Thus, the class of function is closed under composition.

To see that the class is closed under projection. We observe that following function are in the class where a is any element $\in \mathcal{R}$: $\min(x, a)$, $\max(x, a)$, $x + a$, $x \cdot a$, and a/x. □

We count the number of U-intervals of h, the size of h, in terms of the number of intervals of f and g. As we have observed above, each interval of g contributes at most one interval to h. We must determine the contribution from the intervals of f. Let I be some interval of f. It follows that I will generate a new interval for h only when some defined interval J of g is maped onto I by g. Since g is basically one-to-one there can be at most one such interval J. Using the fact that g is monotone on J we see that I generates at most one U-interval for h. Therefore, the size of $g \circ f$ is no more than the summation of the size of g and f. Note that the only projection to generate U-intervals is division. Since we need only one breakpoint between every pair of U-intervals we need only maintain two times the number of divisions intervals for each part of the subtree which we are working. In this representation we note that we can evaluate the function the $O(\log n)$ time

Claim 6.2 *For all $f \in \mathcal{F}$, size$(f) = n$, then for all $x \in \mathcal{R}$, $f(x)$ can be computed in $O(\log n)$ times using using $O(1)$ processors.*

[PROOF] Use binary search, in $O(\log n)$ time, we can determine the interval containing x and then evaluate $f(x)$ in constant time. □

We use this claim to determine the composition of two such functions:

Claim 6.3 *For all $f, g \in \mathcal{F}$ with size m, n respectively, $g \circ f$ can be computed in $O(\log(m+n))$ time, use $O(m+n)$ processors.*

[PROOF] The idea is that assign one processor to each interval of g and f. By reordering the intervals of g via cyclic shift starting at the pole of g we see that their images under g will be ordered intervals which we can determine in $O(\log n)$ time by the last Claim. Now, we simply need to merge the intervals of f with the image intervals of g. This can be done in $O(\log(n+m))$ time using $n+m$ processors. Using this information we can determine the intervals of h. We next decide whether each defined interval of h is a constant segment or linear fraction segment. This can be done in constant time, use $O(n+m)$ processors. □

Claim 6.4 *For all $c \in \mathcal{R}, f, g \in \mathcal{F}$, size$(f)$=$m$, size$(g)$=$n$, $\odot \in \{min, max, +, \times, \div\}$:*

1. *Let $h_\odot(x) = f(\odot(c, x))$, then size$(h_\odot) \leq m+1$.*
2. *Let $h_\circ = g \circ f$, then: size$(h_\circ) \leq m+n$.*

Theorem 6.2 *A min-max-plus-times-division tree of size n over $\mathcal{R}$ can be computed in $O(\log^2 n)$ time using $O(n)$ processors deterministically in EREW model.*

Actually, for simple min-max-plus-times-division-tree, there is tighter bound on the maximum size of the unary function during the computation time. Observe that the projection of any non division node does not generate an U-interval. While a division node generates only one U-interval. Therefore, the maximum size of the functions used during the evaluation of min-max-plus-times-division-tree is bounded by the two times the number of division-nodes. Hence, we have the following theorem.

Theorem 6.3 *Any min-max-plus-times-division tree over $\mathcal{R}$, which has n nodes and d division-nodes, can be computed in $O((\log n)(\log d))$ time using $O((n \log d)/\log n)$ processors.*

7 Dynamic Tree Complexity

We can view a computation tree as a dataflow graph [9], and compute the values of all nodes in a bottom-up manner (eager evaluation) as the execution of conventional data flow program. It is clear that the processor count to compute this dataflow graph equals to the number of the leaves in the tree and the computation time is proportional to the height of the tree. So, in the worst case, we use $O(n)$ processors, but compute the tree in $O(n)$ times. The simple data flow idea does not help anything! The reason is that the complexity of simple dataflow computation depend only on the static complexity of the dataflow graph. (Here, the computational tree) E.g the time complexity of a tree, from the dataflow point of view, is its height and the hardware complexity of a tree is the number of its leaves.

In this paper, we consider the complexity of computation tree dynamicly and study the relationship between the dynamic complexity of a tree and some special family of unary functions. We can define, for sufficient reasons, that the dynamic height of a computation tree is the time complexity to compute the tree, and we have shown in the main theorem, that the dynamic height of a tree can be much smaller than its static height. Therefore, for any problem which can be reduced to a tree computational problem, the following complexities are important.

- The complexity to reduce the problem to a tree problem. We call it tree uniformity of the problem. This contains the complexity for the selection of the operator set and the construction of the tree.
- The complexity of the tree. This is the static complexity of the tree. The height of the is defined to be the length of the longest path between a leaf to root, the size of the tree is the number of its nodes.
- The complexity to specify the family of edge functions over the domain of the problem and the operator set of the tree. We call it edge function uniformity of the problem.
- The complexity for computing composition, forwarding, of the edge functions.
- The complexity to contract the tree. The time to compute the tree is the dynamic height of the tree and the number of the processors used is defined as the dynamic size of the tree. We charge the time and the number of processors to compute the operations over the edge functions as constant.

8 Conclusion and Open Problems

In this paper, we present a systematic method for tree based parallel algorithm generation and prove some nontrivial sufficient condition for the existence of fast parallel algorithm for tree computation. This provides a powerful tool for designing parallel algorithms for many problems. (see part 2 of [16] for survey). Based on our tree computation algorithm and main

theorem, we give the definition of dynamic height of computation tree and study the dynamic complexity of computation tree. This shows, in some sense, our idea has its advantages over that of dataflow model.

Several problems are still interisting both for parallel computing theory and for practical application in parallel processing. We list some of them.

1. What is the necessary condition for a computation tree to be computed in poly-logarithm time, using only polynomial number of processors?
2. Is it possible for us to embed the idea in this paper in some parallel code generator, which will in turn to generate better parallel code from some sequential programming language like Fortran systematicly?
3. What is the systematic way to find the family of unary functions which is minimumly closed over a given operator set?

Appendix. Two Procedures for LOCAL-COMPRESS

PROCEDURE LOCAL-COMPRESS[I]
In Parallel For all node v is a chain do
(1) if v has token R, do
if v also has G token, delete its tokens
else do nothing
(2) if v has no token and its father has token R
then label v by token R;
point to its grandfather
(3) if v has no token
then point to its grandfather

PROCEDURE LOCAL-COMPRESS[II]
begin
(1) isolate chains in the tree
(2) compute the rank of node in the chain
(3) In Parallel
combine the nodes with rank $2i-1$ and $2i$
assign the rank to the generated node by i
end

Claim A.1[4] By using LOCAL-COMPRESS[I] or LOCAL-COMPRESS[II], tree contraction algorithm can be implemented on EREW PRAMs without increasing the time count and processor count.

Claim A.2[5] LOCAL-COMPRESS[II] at most two times as slow as LOCAL-COMPRESS[I]. However, no useless chain is produced during the application of tree contraction algorithm. We call it prudent compress.

[4] see LOCAL COMPRESS lemma in [12]
[5] see claim 4.1 in [12]

References

[1] M.Ajtai, J.Komlos, E.SZEMEREDI. Sorting in *clogn* Parallel Steps. *Combinatorica*, 3 (1) pp1-19, 1983.

[2] R.J.Anderson, G.L.Miller. *Optimal Parallel Algorithm for the List Ranking Problems.* Technical Report, USC, 1987.

[3] I.Bar-On, U.Vishkin. Optimal Parallel Generation of a Computation Tree Form. *ACM TOPLS*, pp348-357, July 1985.

[4] R,P.Brent. The Parallel Evaluation of General Arithmetic Expression. In *JACM*, pp201-208, vol21, April, 1974.

[5] S.A.Cook. Deterministic CFL's Are Accepted Simultaneous in Polynomial Time and log Squared Space. In *STOC79*, ACM, pp338-345, 1979.

[6] S.A.Cook. *The Classification of Problems Which Have Fast Parallel Algorithms.* Technical Report, University of Toronto, 1983.

[7] R.Cole, U.Vishkin. Deterministic Coin Tossing and Accelerating cascades: Micros and Macros Technique for Designing Parallel Algorithms. In *STOC86* pp206-219, 1986.

[8] R.Cole, U.Vishkin Approximate and Exact Parallel Scheduling with Applications to List Tree and Graph Problems In *IEEE FOCS86*, 1986.

[9] A.L.Davis, R.M.Keller. Data Flow Program Graphs. *IEEE Computer*, pp27-41, Feb. 1982.

[10] E.Dekel, S.Ntafos, S.Peng. *Parallel Tree Techniques and Code Optimization.* Technical Report, UTD 1986.

[11] P.W.Dymond, W.L.Ruzzo. Parallel RAMs with Owned Global Memory and Deterministic Context-Free Language Recognition. In *ICACP86*, 1986.

[12] H.Gazit, G.L.Miller, S-H.Teng. *Optimal Tree Contraction in EREW Model.* Technical Report, USC, 1986.

[13] D.D. Gajski, J.K.Peir. Essential Issues in Multiprocessor Systems. *IEEE Computer*, pp9-27, 1985.

[14] H.Gazit. An Optimal Randomized Parallel Algorithm for Finding Connected Components in a Graph. In *FOCS86*, 1986.

[15] P.Klein, J.Reif. An Efficient Parallel Algorithm for Planarity. In *FOCS86*, 1986.

[16] G.L.Miller, J.H.Reif. Parallel Tree Contraction and Its Application. In *FOCS85*, 1985.

[17] G.L.Miller, V.Ramachandran, E.Kaltofen. *Efficient Parallel Evaluation of Straight-line Code and Arithmetic Circuits.* Technical Report, USC, 1985.

[18] G.L.Miller, S-H.Teng. *Dynamic Parallel Complexity of Computational Circuits.* In ACM STOC87, 1987.

[19] J.H.Reif. Parallel Time $O(\log N)$ Acceptance of Deterministic DFL's. In *FOCS82*, 1982.

[20] J.H.Reif. An Optimal Parallel Algorithm for Integer Sorting. In *FOCS85*, IEEE, 1985.

[21] W.L.Ruzzo. Tree-Size Bounded Alternation. *JCSS*, 21(2): pp218-235, October 1980.

[22] W.L.Ruzzo. On Uniform Circuit Complexity. *JCSS*, pp365-383, June 1981.

[23] R.Sethi, J.D.Ullman. The Generation of Optimal Code for Arithmetic Expressions. In *JACM*, 17, 4, pp715-728, Oct. 1970.

[24] L.J.Stockmeyer, U.Vishkin.. Simulation of Random Access Machine by Circuits. *SIAM J.Computing*, July, 1984.

[25] Y.Siloach, U.Vishkin. An O(log n) Parallel Connectivity Algorithm. In *J. of Algorithms* 3, 1, pp57-67, 1983.

[26] R.E.Tarjan, U.Vishkin. Finding Biconnected Components and Computing Tree Functions in Logarithmic Parallel Time. In *IEEE FOCS84*, pp12-22, 1984.

[27] S-H.Teng. *Semantics-Oriented Dataflow Schemes.* Technical Report, USC, 1987.

[28] S-H.Teng, B.Wang. Parallel Algorithms for Message Decomposition. *In Journal of Parallel and Distributed Computing*, June, 1987.

[29] J.D.Ullman, A.V.Gelder. *Parallel Complexity of Logical Query Programs.* TR, Stanford University 1985.

[30] U.Vishkin. *Synchronous Parallel Computation- a Survey.* TR-71, Courant Institute NYU, 1983.

[31] L.G.Valiant S.Skyum S.Berkowitz C.Rackoff. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM J.Computing*, pp641-644, Nov, 1983.

[32] J.C.Wyllie. *The Complexity of Parallel Computation.* Cent. for Comp. Res. TR-79-387, Cornell Univ. 1979.