Broadcasting with Selective Reduction:

An Alternative Implementation and New Algorithms

by

CATHERINE MAXCY CHOW

A thesis submitted to the Department of Computing and Information Science in conformity with the requirements for the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
November, 1997

copyright © Catherine Maxcy Chow, 1997
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-22290-X
Abstract

Broadcasting with Selective Reduction (BSR) is a model of parallel computation. It is an extension of the Parallel Random Access Machine (PRAM) with an additional broadcasting instruction which allows all processors to gain access to all memory locations simultaneously. At each memory location, a subset of incoming data are selected according to a given selection criterion, and reduced to a single value using an appropriate reduction operator.

A generalization of BSR in which several criteria can be applied in the selection process was proposed earlier. We compare the proposed implementation of k-criteria BSR using a distributed memory parallel architecture with another design using a shared memory parallel architecture. This thesis also gives a feasible implementation of the multiple criteria BSR, which is more expandable and gives a better memory access time than the existing designs.

A number of computational problems are investigated and constant time algorithms are proposed. These include the problem of finding non-intersecting straight line connections of grid points to the boundary, variations of the knapsack problem, the longest consecutively-identical subsequence problem, the all point-pair distance problem, and the all point-pair inscribing problem. All the algorithms given are more efficient than the best known PRAM algorithm for the same problem.
We also revisit the convex hull problem for a set of points in the plane, and present for the first time an algorithm which runs in constant time using a number of processors linear in the size of the given set. By applying the same technique used for solving the convex hull problem, we are also able to produce a constant-time linear-cost solution to the convex polygon intersection problem. Finally, an algorithm is suggested for computing the shortest distance between two convex polygons using the multiple-criteria BSR model.
Acknowledgements

I would like to thank my supervisor, Dr. Selim Akl, who gave me generous support and advice during this work. Many thanks to the faculty and staff members of the Department of Computing and Information Science, for making the department an ideal place for doing research. Special thanks to my parents and sisters whose support and encouragement made this possible.
Contents

1 Introduction 1
   1.1 Parallel computation 1
   1.2 The BSR model 1
   1.3 Contributions 2
   1.4 Outline 3

2 Parallel Models of Computation 5
   2.1 The parallel random access machine (PRAM) model 5
   2.2 The broadcasting with selective reduction (BSR) model 8
   2.3 Implementation of BSR 12
      2.3.1 Hypothetical implementation of BSR 12
      2.3.2 An optimal implementation of BSR 15

3 Generalization of BSR 20
   3.1 The k-criteria BSR model 20
   3.2 Implementation of k-criteria BSR:
      Distributed memory parallel architecture 22
   3.3 Implementation of k-criteria BSR:
      Shared memory parallel architecture 25
   3.4 Distributed memory design versus shared memory design
      of the k-criteria BSR model 29
## CONTENTS

4 **An Alternative Implementation of the k-criteria BSR** 30

4.1 Filter circuit 30

4.2 Equal circuit 32

4.3 The full k-criteria BSR circuit 34

4.4 Cost analysis of the implementation of the k-criteria BSR circuit 39

4.5 Example of a k-criteria BSR problem 42

5 **BSR Algorithms** 48

5.1 Finding non-intersecting straight line connections of grid points to the boundary problem 53

5.2 Variations of the knapsack problem 67

5.2.1 One dimensional variation of the knapsack problem 68

5.2.2 Two dimensional variation of the knapsack problem 70

5.3 Longest consecutively-identical subsequence 71

5.4 All point-pair distance problem 74

5.5 All point-pair inscribing problem 76

6 **Multiple-Criteria BSR Algorithms** 79

6.1 Convex hull 79

6.2 Intersection of two convex polygons 84

6.3 Shortest distance between two convex polygons 88
CONTENTS

7 Conclusions and Future Work 92

Bibliography 94

Vita 98
List of Figures

1  The PRAM architecture 5
2  Three phases BROADCAST instruction operation 9
3  Hypothetical implementation of BSR using memory buses 13
4  Hypothetical implementation of BSR using a mesh of trees 14
5  An optimal implementation of BSR 15
6  A network for two-way prefix computation 16
7  Implementation of k-criteria BSR network:
   Distributed memory parallel architecture 23
8  A k-criteria BSR switch 24
9  Implementation of k-criteria BSR network:
   Shared memory parallel architecture with memory buses 26
10 Implementation of k-criteria BSR network:
    Shared memory parallel architecture with a mesh of trees 28
11 Implementation of the FILTER circuit 31
12 Implementation of the EQUAL circuit with $N = 8$ and $DNum = 3$ 33
13 An alternative implementation of k-criteria BSR 35
14 An example of k-criteria BSR (Steps 1 to 3) 45
15 An example of k-criteria BSR (Steps 3 to 6) 47
LIST OF FIGURES

16 Non-intersecting straight line connections of grid points to the boundary problem 54
17 Examples of intersecting and non-intersecting polygons 85
Chapter 1

Introduction

1.1 Parallel computation

As the demand for faster machines increases in today’s dynamic world, parallel computation has emerged as an important field in computer science. Parallel computers are considered more attractive than their sequential counterparts because of their ability to speed up computation. A great deal of research has been carried out with the purpose of developing different models and algorithms for parallel computers with multiple processors. Such parallel computers have been made feasible by the enabling technology of very large scale integrated circuits. Many different models of parallel computation have been proposed. One such model is the popular Parallel Random Access Machine (PRAM). This thesis is devoted to studying an enhancement of the PRAM, namely, the broadcasting with selective reduction (BSR) model. We propose an extension and an implementation of BSR, as well as fast algorithms for solving several computational problems on it. These algorithms are more efficient than the best known PRAM algorithms for the same problems.

1.2 The BSR model

BSR is a model of parallel computation first proposed in [10]. It is an extension of the
CHAPTER 1  INTRODUCTION

PRAM, the most widely used model of parallel computation\textsuperscript{[7]}: It supports all forms of memory access allowed by the PRAM, that is, exclusive read (ER), exclusive write (EW), concurrent read (CR), and concurrent write (CW), with an additional BROADCAST instruction which allows all \( N \) processors to write to all \( M \) memory locations simultaneously. It is shown that while an optimal implementation of BSR requires no more resources than the weakest variant of the PRAM, namely, the EREW PRAM, BSR is more powerful than the strongest variant of the PRAM, namely, the CRCW PRAM\textsuperscript{[11]}. The BROADCAST instruction consists of three phases, the broadcast phase, the selection phase and the reduction phase. During the selection phase, the BROADCAST instruction (as described in\textsuperscript{[10]}) required that only one selection criterion be applied to the processor records. A generalization of BSR allows \( k \)-criteria selection to take place at each memory location. An implementation of the \( k \)-criteria BSR using a distributed memory architecture is suggested in\textsuperscript{[14]}. Many BSR algorithms have been designed to solve a variety of traditional problems\textsuperscript{[9,10,13,14,18,22,26]}. These algorithms provide constant-time solutions using a number of processors linear or quadratic in the size of each problem.

1.3 Contributions

To date, all implementations of the generalized BSR model are hypothetical, i.e. those are conceptual designs only. An alternative implementation of the multiple-criteria BSR using well-known circuits is presented in the thesis. This design is more expandable and
CHAPTER 1  INTRODUCTION

gives a better access time than the existing designs. Constant-time linear-cost BSR algorithms for a number of computational problems are suggested. These include the problem of finding non-intersecting straight line connections of grid points to the boundary, variations of the knapsack problem, the longest consecutively-identical subsequence problem, the all point-pair distance problem, and the all point-pair inscribing problem. All the algorithms given are more efficient than the best known PRAM algorithms for the same problems. Constant-time algorithms for solving the convex hull problem, the shortest distance between two convex polygons problem, and the convex polygon intersection problem are also given. Each of these algorithms requires a number of processors that is linear in the size of the problem being solved.

1.4 Outline

The remaining chapters are organized as follows. Chapter 2 reviews the PRAM, and provides a detailed description of the BSR model, its operation and implementation. In chapter 3, we consider a generalization of the BSR model. The existing distributed memory implementation and an alternative shared memory design are discussed and compared. In chapter 4, an alternative implementation of the generalized BSR model is given. Chapter 5 presents constant-time linear-cost BSR solutions to different computational problems. Chapter 6 addresses the convex hull problem and other related
convex polygon problems. We conclude our work and give an overview of open
problems in chapter 7.
Chapter 2
Parallel Models of Computation

2.1 The parallel random access machine (PRAM) model

The Parallel random access machine (PRAM) is the most widely accepted parallel computational model [7]. It consists of $N$ processors, each with its own local memory, $M$ shared memory locations, and a memory access unit (MAU) which allows processors to gain access to the shared memory. The architecture of the PRAM is shown in figure 1.

Figure 1: The PRAM architecture
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

Several variants of the PRAM exist. They differ from one another by the way the processors gain access to the shared memory. The instructions for PRAM processors to read from or write to memory locations include:

ER (Exclusive read): Processors can read from memory locations simultaneously, but no two processors are permitted to read from the same memory location at the same time.

EW (Exclusive write): Processors can write to memory locations simultaneously, but no two processors are permitted to write into the same memory location at the same time.

CR (Concurrent read): Two or more processors are allowed to read from the same memory location at the same time.

CW (Concurrent write): Two or more processors are allowed to write into the same memory location at the same time.

The CW instruction is further classified into COMMON WRITE, PRIORITY WRITE, ARBITRARY WRITE, COMBINING WRITE and RANDOM WRITE, by the type of conflict resolution rule applied. Such a rule determines the final datum stored in a memory location when more than one processor attempts to write simultaneously into that memory location. For example, with a COMMON WRITE, a write to a particular shared variable succeeds only if all of the values being written are identical; otherwise the variable is unaffected by the attempted writes. The most powerful of these CW rules is COMBINING WRITE, where the values that the processors wish to write in a given memory location are combined (arithmetically or logically) into a single value. Note,
however, that at any given time, at most $N$ memory locations allow $N$ processors to read data from them, or write data to them. Each step of a PRAM algorithm consists of a READ phase, a COMPUTE phase and a WRITE phase. During the READ and WRITE phases, processors read data from and write data to the shared memory respectively. During the COMPUTE phase, processors perform arithmetic or logical operations on the data. How long does a PRAM step take? The time required by the READ and WRITE phases is dictated by the MAU. The latter is typically implemented as a combinational circuit, that is a device consisting of simple components arranged in stages. The components in each stage receive their inputs from previous stages (via direct connections) and deliver their outputs to subsequent stages (via direct connections). Each component can do a basic operation, such as adding two inputs, in constant time, and is used only once. The depth of a combinational circuit is the maximum number of components on a path, over all paths from input to output. The width of a combinational circuit is the maximum number of components in a stage, over all stages. The size of a combinational circuit is the total number of components used by the circuit. A combinational circuit implementing a PRAM's MAU has a depth of $O(\log M)$, a width of $O(M)$, and a size of $O(M\log M)$, and these values are optimal. This implies that the time taken by the READ and WRITE phases, denoted by $\tau_s(N,M)$, is $O(\log M)$. The time taken by the COMPUTE phase depends on how a basic operation such as adding or comparing two numbers is implemented. Typically, such an operation takes time that is logarithmic in the number of bits used to represent each of the two numbers.
CHAPTER 2 PARALLEL MODELS OF COMPUTATION

Assuming that the PRAM’s word size is \( O(\log M) \), the time taken by the COMPUTE phase, denoted by \( \tau_c (N, M) \), is \( O(\log \log M) \). In the PRAM literature, for simplicity, both \( \tau_s (N, M) \) and \( \tau_c (N, M) \) are taken to be constant. Therefore a PRAM steps requires \( \tau_s (N, M) + \tau_c (N, M) = O(1) \) time.

2.2 The broadcasting with selective reduction (BSR) model

Broadcasting with selective reduction (BSR) is an extension of the PRAM \(^{[10]}\). In addition to allowing all forms of memory access of the PRAM, namely, ER, EW, CR and CW, the BSR model also permits a BROADCAST operation, which provides all processors access to all shared memory locations simultaneously for the purpose of writing. A BROADCAST instruction consists of three phases. The broadcast phase allows all of the \( N \) processors to write concurrently to all of the \( M \) memory locations. Processor \( P_i \), \( 1 \leq i \leq N \), produces a record containing two fields, a tag \( g_i \) and a datum \( d_i \), where the tag will help identify those locations in which the datum is to be stored. After the data are received at each memory location \( U_j \), \( 1 \leq j \leq M \), a switch \( S_j \) associated with that memory location will select a subset of the receiving data \( d_i \) by comparing the tag value \( g_i \) of the datum \( d_i \) with a limit value \( l_j \), using a selection rule \( \sigma \). In the last phase, the selected data are then reduced to a single value using a binary associative reduction rule \( R \). Each of the three phases is performed simultaneously by \( N \) processors and \( M \) switches.
CHAPTER 2  \textit{PARALLEL MODELS OF COMPUTATION}

at the \( M \) memory locations. The three phases of the \textsc{broadcast} instruction operation are described in figure 2.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{broadcast_diagram}
\caption{Three phase \textsc{broadcast} instruction operation}
\end{figure}

The selection rule, \( \sigma \), can be chosen from the following set of relational operators:

\[ \{ <, \leq, =, \geq, >, \neq \} \]
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

The reduction rule, $R$, can be selected from the following set of binary associative operators, which denote sum, product, and, or, exclusive-or, max and min:

$$\{ \sum, \prod, \land, \lor, \oplus, \cap, \cup \}$$

The BROADCAST instruction is written as follows:

$$U_j \leftarrow R_{\sigma} d, \quad 1 \leq j \leq M$$

This notation means that at each memory location $U_j$, we compare the tag $g_i$ broadcast by processor $P_i$, with the limit $l_j$ associated with that particular memory location $U_j$. If the selection rule $\sigma$ is satisfied for the pair of $g_i$ and $l_j$, then the datum $d_i$ which is associated with the tag $g_i$ will be selected. For all the selected data, the reduction rule $R$ will be used to reduce them into one single value. This single value will then be stored in memory location $U_j$.

The BROADCAST instruction can be implemented using a combinational circuit of depth $O(\log M)$, width $O(M)$, and size $O(M \log M)$ \cite{footnote}. It therefore takes $O(\log M)$ time to be executed. Consistent with the assumption regarding memory access time on the PRAM (see end of Section 2.1), this time is also considered to be constant.
CHAPTER 2 PARALLEL MODELS OF COMPUTATION

The BSR model incorporates all forms of memory access allowed on PRAM, namely, ER, EW, CR, and CW, plus the BROADCAST operation. The latter is equivalent to \(NM\) concurrent write operations performed simultaneously by all \(N\) processors at all \(M\) memory locations. Since the PRAM does not allow \(M\) simultaneous writes by each of the \(N\) processors, it follows that the BSR model is at least as powerful as the PRAM. To illustrate this, consider the computation of the prefix sums of a sequence \(x_0, x_1, x_2, \ldots, x_{N-1}\). That is, we would like to find \(s_i = x_0 + x_1 + \ldots + x_i\) for \(0 \leq i \leq N-1\). On the PRAM, the problem can be solved in \(O(\log N)\) time using \(N\) processors. Initially, let \(s_i = x_i\) for \(i = 0, 1, 2, \ldots, N-1\). During the \(j\)th step, where \(j = 0, 1, \ldots, (\log N) - 1\), the following operation is performed simultaneously for all \(i, 2^j \leq i \leq N-1:\)

\[
s_i \leftarrow s_i \cdot 2^j + s_i
\]

During each step, two numbers are added whose indices are twice the distance in the previous step. At the end of the \(\log N\) steps, each of \(s_i\) holds the required prefix sum value.

On the PRAM, it is clear that no algorithm can solve this problem in constant time using \(N\) processors. In fact, the \(N\) processors can compute each one of the \(s_i\) in \(O(\tau_d(N, M))\), that is \(O(1)\) time, but not all of \(s_0, s_1, s_2, \ldots, s_{N-1}\) at the same time. By contrast, the same problem can be solved on BSR using just one BROADCAST instruction:

1. Processor \(P_i\), \(0 \leq i \leq N - 1\), broadcasts \((i, x_i)\) as the tag and datum pair.
2. Memory location \(U_j\) selects those data \(x_i\) with tag \(i\) less than or equal to \(j\), \(0 \leq j \leq N - 1\).
3. Those $x_i$ selected by $U_j$ are added up to obtain $s_j$ at each memory location, $0 \leq j \leq N - 1$.

Using the notation given earlier, we have:

$$s_j \leftarrow \sum_{0 \leq i \leq N - 1} x_i$$

This requires $O(1)$ time and $N$ processors.

2.3 Implementation of BSR

The MAU for BSR can be implemented using a combinational circuit of depth $O(\log M)$, width $O(M)$, and size $O(M \log M)$; this implementation is optimal and thus BSR requires asymptotically no more resources than the PRAM. In fact, a number of implementations of BSR exist; all these implementations allow BSR to execute the BROADCAST instruction as well as all forms of memory access permitted on different variants of the PRAM (i.e., ER, EW, CR, CW).

2.3.1 Hypothetical implementation of BSR

A hypothetical implementation of BSR has been suggested by using memory buses. Here, each processor possesses a bus to which all memory locations are connected. This permits broadcasting of records from each processor to all memory locations simultaneously. There are $N$ switches associated with each memory location, the switches select a subset of the incoming data which satisfy the selection criterion $\sigma$. Finally, at
each memory location, a binary tree, which is called a concurrent access tree (CAT), reduces the selected data into one single value, and stores that value in the associated memory location. The implementation of BSR using memory buses is illustrated in figure 3.

Figure 3: Hypothetical implementation of BSR using memory buses

Another hypothetical implementation of BSR uses a mesh of trees \(^{11}\) is shown in figure 4. Each of the \(N\) processors is connected to the root of a binary tree with \(M\) leaves, and each of the \(M\) memory locations is connected to the root of a binary tree with \(N\) leaves. Each of the \(M\) leaves of a processor tree is connected to a memory location tree through
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

a switch. This allows the processors to broadcast their data to all the $M$ memory locations at the same time. When the data reach the switches at the leaves of the binary tree at each memory location, they are tested. Those data which satisfy the selection rule will be allowed to go through, and reduced into a single value that is stored in that memory location.

Figure 4: Hypothetical implementation of BSR using a mesh of trees
2.3.2 An optimal implementation of BSR

An optimal implementation of BSR [21] using a combination of sorting, merging and prefix sum circuits is shown in figure 5. The circuit consists of six components, of which three are sorting circuits, two are prefix circuits and one is a merging circuit. Any optimal size sorting/merging circuit can be chosen. The depth of the chosen circuit will then determine the memory access time of the implementation. Batcher's Odd-Even or Bitonic sorting circuits [15], and the AKS circuit [2][23] have been proposed as the building blocks of BSR.

Figure 5: An optimal implementation of BSR
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

The prefix circuits here are designed to handle two-way prefix computation, that is, both prefix and suffix computation. They are prefix sum circuits augmented with duplicated connections between distinct rows to allow information to pass in both forward and backward directions. A circuit for two-way prefix computation is shown in figure 6. For an input of size $k$, the circuit has a width of $O(k)$, a depth of $O(\log k)$, and a size of $O(k \log k)$.

For $1 \leq i \leq N$, processor $P_i$ produces a processor record $(i, g_i, d_i)$, where $i$ is the processor's index, $g_i$ its tag, and $d_i$ its datum. These $N$ records are received as input by the MAU. For $1 \leq j \leq M$, memory location $U_j$ produces a memory record $(j + N, l_j, v_j)$, where $j$ is the index of the memory location, $l_j$ its limit, and $v_j$ the variable that holds the datum to be stored in $U_j$. The majority of the computational work is actually done in the
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

PREFIX circuit B where temporary combined results are computed. The distribution of results from the processors to the memory locations is done in PREFIX circuit E, where an interval broadcasting method is employed to pass results from processor records to memory records. Finally, the memory records are separated from the processor records using the SORT circuit F.

This is how the BSR BROADCAST instruction is executed at the memory access unit:

- The SORT circuit A accepts records \((i, g_i, d_i)\) from each of the \(N\) processors, \(P_i\), and sorts them according to the tag \(g_i\). If the tags are equal, datum \(d_i\) will be used to order the records. A rank field \(r_i\) is added to indicate the sorting order. The SORT circuit outputs records in the form \((i, r_i, g_i, d_i)\).

- The PREFIX circuit B accepts records output from SORT circuit A, performs a prefix computation on the datum \(d_i\) according to the selection and reduction rules, \(\sigma\) and \(\mathcal{R}\), of the particular computational problem.

- The SORT circuit C accepts records \((a, l_j, v_a)\) from \(M\) memory locations (where \(a\) is the address of memory location \(U_j\)), and sorts them according to the limit \(l_j\) associated with the memory location \(U_j\).

- At MERGE circuit D, the sorted processor records output from PREFIX circuit B, are merged with the sorted memory records output from the SORT circuit C, according to the tag \(g_i\) and limit \(l_j\).
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

CHAPTER 2  PARALLEL MODELS OF COMPUTATION

- PREFIX circuit E accepts outputs from MERGE circuit D, and performs interval broadcasting to distribute the temporary computational results, that is the modified datum \( a_i \), from processor records to appropriate memory records.

- SORT circuit F receives output from PREFIX circuit E, replaces the address fields \( a \) of each memory record by \( (a + N) \), and sorts the records according to the index or address field of the processor and memory records. Hence, at the end of the circuit, memory records are separated from the processor records.

Suppose that the AKS circuit is used as the sorting/merging circuit of BSR. For an input of size \( K \), this circuit has a width of \( O(K) \), a depth of \( O(\log K) \) and a size of \( O(K\log K) \) \cite{2}\cite{23}. Therefore, each building block will have the following cost:

- SORT circuit A: \( O(N) \) width, \( O(\log N) \) depth
- SORT circuit C: \( O(M) \) width, \( O(\log M) \) depth
- MERGE circuit D: \( O(M + N) \) width, \( O(\log (M + N)) \) depth
- SORT circuit F: \( O(M + N) \) width, \( O(\log (M + N)) \) depth

Since the modified prefix sum circuit used also has the same order of implementation cost, so we have:

- PREFIX circuit B: \( O(N) \) width, \( O(\log N) \) depth
- PREFIX circuit E: \( O(M + N) \) width, \( O(\log (M + N)) \) depth
CHAPTER 2  PARALLEL MODELS OF COMPUTATION

Therefore, assuming $N = O(M)$, the combinational circuit implementing BSR’s memory access unit will have an overall width of $O(M)$, depth of $O(\log M)$ and size of $O(M \log M)$. These values are optimal since any combinational circuit for the PRAM (and hence BSR) must have a width of $\Omega(M)$, a depth of $\Omega(\log M)$, and a size of $\Omega(M \log M)$. [5].
Chapter 3

Generalization of BSR

3.1 The k-criteria BSR model

So far, during a BSR BROADCAST instruction, a datum $d_i$, broadcast by a processor $P_i$, was selected by a memory location $U_j$ for reduction if $d_i$ (or, more precisely, its associated tag $g_i$) satisfied one criterion. In some computational problems, it is necessary for the data to satisfy more than one criterion before they can be selected and reduced in a memory location. A generalization of BSR allows the data to be tested for satisfaction of $k$ criteria, where $k \geq 1^{[13][14]}$. Here, we let $\sigma_h$ be the $hth$ selection rule, for $1 \leq h \leq k$, where each of the $k$ selection rules can be selected from the relational operator set defined in section 2.2. The notation for the tag and limit fields of the processor and memory records are now denoted by $g_{i,h}$ and $l_{j,h}$ respectively. Tag $g_{i,h}$ represents the tag broadcast by processor $P_i$ to be used with selection criterion $\sigma_h$. Limit $l_{j,h}$ represents the limit associated with memory location $U_j$ to be used with selection criterion $\sigma_h$. Now, each record of processor $P_i$ consists of tags $g_{i,1}$, $g_{i,2}$, ..., $g_{i,k}$ and datum $d_i$, and each memory location $U_j$ consists of limits $l_{j,1}$, $l_{j,2}$, ..., $l_{j,k}$. Note that the $k$ tags and $k$ limits may all be different.
CHAPTER 3  GENERALIZATION OF BSR

The k-criteria BSR BROADCAST instruction is denoted as follows:

\[ U_j \leftarrow \bigwedge_{1 \leq i \leq N_{SM}} \bigwedge_{1 \leq h \leq k} g_{i,h} \sigma_h l_{i,h}, \text{ for } 1 \leq j \leq M \]

This notation means that at each memory location \( U_j \), for each criterion \( \sigma_h, 1 \leq h \leq k \), we test the tag \( g_{i,h} \) of the record from processor \( P_i \) with the limit \( l_{i,h} \) of the record from memory location \( U_j \). If all \( k \) selection rules \( \sigma_h, 1 \leq h \leq k \), are satisfied, then the datum \( d_i \) will be selected. All the selected data will then be combined and reduced into one single value according to the reduction rule, \( \mathcal{R} \), of that computational problem. If no datum is accepted by memory location \( U_j \), then an identity element \( e(\mathcal{R}) \), which is unique to the specific reduction rule \( \mathcal{R} \), will be received by \( U_j \). The identity elements of sum, product, AND, OR, exclusive-OR, max, and min operations are 0, 1, true, false, false, \(-\infty\) and \( \infty \) respectively.

For example, consider the following General Prefix Computation (GPC) problem. Let \( f(1), f(2), \ldots, f(n) \), and \( y(1), y(2), \ldots, y(n) \) be two sequences of elements with a binary associative operator "*" defined on the \( f \) elements, and a linear order "<" defined on the \( y \)-elements. It is required to compute the sequence:

\[ D(m) = f(j_1) \ast f(j_2) \ast \ldots \ast f(j_k) \quad \text{for } m = 1, 2, \ldots, n, \]

where \( j_1 < j_2 < \ldots < j_k \), and \( \{j_1, j_2, \ldots, j_k\} \) is the set of indices \( j_i < m \) for which \( y(j_i) < y(m) \). In order to solve GPC in constant time on the BSR model, double selection is required, the first selection rule being \( j_i < m \), and the second \( y(j_i) < y(m) \) \([5]\).
CHAPTER 3  GENERALIZATION OF BSR

3.2 Implementation of k-criteria BSR: Distributed memory parallel architecture

A circuit for implementing selection on a k-criteria BSR is proposed in [14]. Here, $M$ memory locations are distributed among the processors, such that each processor has a local memory of its own of size $M/N$, which is accessible indirectly by other processors. The memory access unit of BSR consists of $N^2$ switches arranged as a mesh, where $N$ is the number of processors. Each switch $S(i,j)$ is connected with processor $i$ and processor $j$ via a horizontal and vertical bus. For each $i$, $1 \leq i \leq N$, and fixed $j$, the $N$ switches are constructed such that they form the leaves of a binary tree, called the Concurrent Access Tree (CAT). There are in total $N$ horizontal and $N$ vertical buses, each bus contains $O(k)$ separate lines which allow transmission of $k$ sets of data from each processor to the switches. For each selection criterion $\sigma_h$, $1 \leq h \leq k$, each switch $S(i,j)$ accepts datum $d_i$ and tag $g_{i,h}$ from the processor $P_i$, and limit $l_{j,h}$ from memory location $U_j$. If all $k$ criteria are satisfied for all pairs of $g_{i,h}$ and $l_{j,h}$, then the datum $d_i$ will be allowed to go through. The concurrent access tree CAT$_j$, which is constructed at the top of the $j$th vertical bus, receives data through the switches, combines all the selected data, and reduces them into a single value. Here, each processor is assigned $O(k)$ memory, where all data (tags, limits, datum and selection criteria) can be stored and sent directly to the switches via buses. Using buses, switches can receive all the necessary data from the processors and compute them all at once. The implementation is shown in figure 7, with the processors drawn twice for the sake of simplicity.
CHAPTER 3 GENERALIZATION OF BSR

Figure 7: Implementation of k-criteria BSR network: Distributed memory parallel architecture

Each switch consists of k gates (figure 8), whose k output lines go into an AND gate, to which a multiplexer is connected at the other end. At each gate h, 1 ≤ h ≤ k, the tag $g_{ih}$ is compared with limit $l_{ih}$ using the selection rule $\sigma_h$. If criterion h is satisfied, gate h will send a true signal to the AND gate. If all k criteria are satisfied, another true signal will be produced from the AND gate. Upon receiving the true signal from the AND gate, the multiplexer will then allow the datum $d_i$ to go through. In the case where not all the k
CHAPTER 3 GENERALIZATION OF BSR

Figure 8: A $k$-criteria BSR switch

criteria are satisfied, a false signal will be produced from the AND gate, after receiving this signal, the multiplexer will output an identity value, $e(\mathcal{R})$, instead.

However, an output of an identity value resulting from the failure to accept a datum has to be distinguished from the output value of a successful reduction. To do this, we can
either introduce a new failure symbol, or modify the operation at the algorithmic level. The former method is too complicated to implement because extra hardware has to be added, so modification at the algorithmic level is suggested. For example, to distinguish a successful AND reduction leading to a "true" from a "true" caused by data acceptance failure, we can replace the instruction:

\[ U_j \leftarrow \bigwedge d_i \text{ 1 criteria} \]

by the following instruction:

\[ U_j \leftarrow \sum 1 \text{ 1 criteria} \]

Now, a successful "true" can be recognized by a non-zero sum, while a failure "true" will be indicated by a zero value. The algorithmic designs of other selection rules are modified using similar ideas.

3.3 Implementation of k-criteria BSR:

Shared memory parallel architecture

An alternative way to implement k-criteria BSR is to exploit the shared memory architecture. Unlike the previous implementation where memory is distributed among the processors, here memory is directly shared by processors. The first implementation we suggest here is similar to the hypothetical implementation of the 1-criterion BSR using memory buses (see figure 9). Each processor \( P_i \) possesses a memory bus to which all
memory locations are connected. Datum $d_i$ and tags $g_{i,1}, g_{i,2}, \ldots, g_{i,k}$ are sent simultaneously from each processor through these memory buses to all memory locations at the switches in unit time. Each switch $S_{j,1}$ also receives $k$ selection criteria $\sigma_1, \sigma_2, \ldots, \sigma_k$. 

Chapter 3 Generalization of BSR
through another memory bus from a special memory location containing the k selection criteria, and a stream of limits \( l_{j,1}, l_{j,2}, \ldots, l_{j,k} \) from memory location \( U_j \). The switches are identical to the ones introduced in the previous section. Each switch contains k gates (see figure 8), each of which will compare the tag \( g_{i,h} \) with the limit \( l_{j,h} \) using the selection criterion \( \sigma_h \). If all k gates return a true signal after the comparison, a true signal will be returned from the AND gate at the switch, then the data \( d_i \) will be allowed to go through. Finally, the concurrent access tree \( \text{CAT}_j \), which is constructed at each memory location connecting all the switches, will reduce the selected data into one single value using the reduction rule \( \mathcal{R} \).

The second implementation we suggest here which also exploits the shared memory architecture, is similar to the hypothetical implementation of the 1-criterion BSR using a mesh of trees. Each of the \( N \) processors is connected to the root of a binary tree with M leaves. Each of these M leaves is connected to different memory locations through a switch. Datum \( d_i \) and tags \( g_{i,1}, g_{i,2}, \ldots, g_{i,k} \) are sent simultaneously from each processor through these leaves to all memory locations at the switches in unit time. Like the previous shared memory architecture using memory buses, each switch \( S_{j,i} \) receives k selection criteria \( \sigma_1, \sigma_2, \ldots, \sigma_k \) and a stream of limits \( l_{j,1}, l_{j,2}, \ldots, l_{j,k} \) from memory location \( U_j \); it performs a similar selection operation through the k gates and passes the selected data to the concurrent access tree \( \text{CAT}_j \) where selected data are reduced into one single
value. The implementation of BSR using shared memory architecture with a mesh of trees is illustrated in figure 10.

![Diagram of k-criteria BSR network](image_url)

**Figure 10:** Implementation of k-criteria BSR network: Shared memory parallel architecture with a mesh of trees
CHAPTER 3  GENERALIZATION OF BSR

3.4 Distributed memory design versus shared memory design of the k-criteria BSR model

In the design of Section 3.2, the $M$ locations of memory are distributed among the $N$ processors. The circuit has $N^2$ switches. However, although each processor can access its local memory in unit time, additional communication time is required to transfer data or messages among processors during a computation. Since the distance separating the furthest two processors is $O(N)$, the data communication time is $O(N)$ in the worst case. By contrast, the design of Section 3.3 uses $NM$ switches; this number is larger than $N^2$ when $M > N$ (as the number of memory locations is usually greater than the number of processors). Nonetheless, because the memory is shared, all communications are through the memory and require $O(\log M)$ time at worst. When $\log M < N$ (which one would expect in typical circumstances as the number of processors and the number of memory locations are usually comparable) this is an improvement over the distributed memory design.
Chapter 4

An Alternative Implementation of the k-criteria BSR

In this chapter, we propose an alternative implementation of k-criteria BSR. This implementation is easy to implement and expand as it uses common circuits as building blocks. The building blocks include the SORT, MERGE and PREFIX circuits that are also used in the optimal implementation of one-criterion BSR. Additional circuits used are the FILTER circuit and the EQUAL circuit. By using any SORT, MERGE and PREFIX circuits whose width is $O(N)$ and depth $O(\log N)$, together with the new FILTER circuit which also has a width of $O(N)$ and a depth of $O(\log N)$, and the EQUAL circuit of width $O(N)$ and depth $O(k)$, a k-criteria BSR can be implemented with a circuit of total size $O(kN^2\log N)$.

4.1 FILTER circuit

The FILTER circuit accepts $N$ records, $R_1, R_2, \ldots R_N$, each of which carries a field called $t_i$. This field $t_i$ can hold data of one of two data types, namely, memory location, and either numeric, string, or others depending on the computational problem. Once $N$ records are input to the circuit in order, the circuit will search for the location of the first record $R_k$ with datatype memory location, then output the first $k$ records: $R_1, R_2, \ldots R_k$. 
The main objective of the FILTER circuit is to extract the set of input records from the first one up to the one which represents a memory location record.

The implementation of FILTER circuit is shown in figure 11. The FILTER circuit is implemented using a PREFIX circuit with \( N - 1 \) extra switches. Each switch receives the field \( t_{i,j} \) from record \( R_{i-1} \). It will then check the datatype of \( t_{i,j} \). The value False will be assigned internally to memory register \( MR_i \) if \( t_{i,j} \) is of datatype memory location, and the value True will be written otherwise. Prefix computation on the values of \( MR_i \) is then
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

performed by an AND operator at the PREFIX circuit. At the end of the prefix computation, those records with MR, equal to TRUE will be output. Here in our example, our record layout is \((i, \text{tag}_1, \text{tag}_2, \ldots, \text{tag}_y, d_i)\), all \(d_i\) are non-memory location type while \(V_x\) is memory location type. Since the FILTER circuit uses the PREFIX circuit as a building block, it follows that by using a PREFIX circuit with a width of \(O(N)\) and a depth of \(O(\log N)\), we can have the FILTER circuit implemented with a width of \(O(N)\), a depth of \(O(\log N)\), and an overall size of \(O(M \log N)\).

4.2 EQUAL circuit

The EQUAL circuit is used to return consecutively duplicate records as one single record with the number of duplicates preset as DNum. In other words, a unique copy of those records which have DNum copies input consecutively will be returned from the circuit.

The implementation of the EQUAL circuit is shown in figure 12. The EQUAL circuit accepts \(N\) records, \(R_1, R_2, \ldots, R_N\), each of which carries an index field \(i\). At the first stage upon receiving \(N\) records, each switch \(S_i\) accepts records \(R_i\) and \(R_{i+1}\). It checks to see if they are equal; if so, then the first of these records will be allowed to go through. At the \(jth\) stage, where \(1 \leq j \leq DNum - 1\), switch \(S_i\) accepts records output from the previous stage, specifically, from the \(i\)th and the \((i+1)\)th switch. In this example, \(N\) is equal to 8, \(DNum\) is set to 3, and the record layout is \((i, \text{tag}_1, \text{tag}_2, \ldots, \text{tag}_y, d_i)\).
It is clear that $DNum - 1$ comparisons (and hence stages) are sufficient to test for consecutive duplication of $DNum$ records. Therefore, with an input size of $N$, the EQUAL circuit has an overall width of $O(N)$ and a depth of $O(DNum)$, resulting in a total size of $O(N \times DNum)$. For the purpose of k-criteria BSR implementation, $DNum$ of the EQUAL circuit will be set to $k$, the number of selection criteria allowed. Therefore, the circuit has a total width of $O(N)$, depth of $O(k)$ and an overall size of $O(kN)$. 

Figure 12: Implementation of the EQUAL circuit with $N = 8$ and $DNum = 3$
CHAPTER 4 A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

4.3 The full k-criteria BSR circuit

The generalization of BSR which allows k-criteria selection accepts \( N \) records from the processors, each of these records \( PR_i \) consists of tags \( g_{i,1}, g_{i,2}, \ldots, g_{i,k} \) and datum \( d_i \). These tags which are used with selection criteria \( \sigma_n, 1 \leq n \leq k \), are then compared with \( M \) sets of limits \( l_{j,1}, l_{j,2}, \ldots, l_{j,k} \) provided by memory record \( MR_j \). The set of data which satisfy all \( k \) selection criteria will then be combined and reduced to one single value according to the reduction rule \( \mathcal{R} \). The result will be stored in the memory field \( v_j \) provided by memory record \( MR_j \). These \( M \) memory records \( MR_j \) will then be returned. The BSR Broadcasting instruction is as follows:

\[
U_j \leftarrow \bigcap_{1 \leq i \leq N} \bigcap_{l \in S_k} \bigcap_{1 \leq s \leq M} \mathcal{R} d_i \land g_{i,h} \land \sigma_h \land l_{j,h}, \text{ for } 1 \leq j \leq M
\]

The alternative full k-criteria BSR memory access unit can be implemented using the SORT, MERGE, PREFIX, FILTER and EQUAL circuits. A k-criteria BSR problem with \( N \) inputs requires \( k \) SORT circuits, \( (k+1)N \) MERGE circuits, \( kN \) FILTER circuits, \( N \) PREFIX circuits and \( N \) EQUAL circuits. The implementation diagram is shown in figure 13. The idea of the circuit design is as follows: \( k \) copies of the records are first sorted on \( k \) different tags concurrently, each of these separate sets of sorted records are then compared with one of the \( M \) memory records (on the tag and limit fields) to produce a set of eligible records for that memory record that satisfy exactly one of the \( k \) criteria. These different eligible sets of records are then combined, and only those records that satisfy
Figure 13: An alternative implementation of k-criteria BSR
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

all k criteria will be selected. Finally, these selected records are reduced to a single
value according to the reduction rule $R$. The BSR selection actually takes place in the
MERGE and FILTER circuits in the second and third step of the process, while the
reduction step takes place at the PREFIX circuits in the last step. The execution of the
k-criteria BSR BROADCAST instruction is done with the full k-criteria BSR circuit as
follows:

Step 1: SORT circuit
There are in total of $k$ SORT circuits arranged in parallel in the full k-criteria BSR
model. Upon receiving $N$ records with record layout $(i, g_{i,1}, g_{i,2}, \ldots, g_{i,k}, d_i)$ from $N$
processors, each of the SORT circuits sorts the records on one of the $k$ tags, with
SORT$_h$ responsible for sorting records on the tag $g_{i,h}$. If the tags are equal, datum $d_i$ will
be used to order the records. The SORT$_h$ circuit outputs records in the form $(i, g_{i,1}, g_{i,2},
\ldots, g_{i,k}, d_i)$ in sorted order of $g_{i,h}$.

Step 2: MERGE circuit
There are $k$ levels of $M$ MERGE circuits in total arranged in parallel in the full k-criteria
BSR. Memory location $U_j$ sends memory record $MR_j$ $(j, l_{j,1}, l_{j,2}, \ldots, l_{j,k}, v_j)$ to the set of
k MERGE circuits MERGE$_{j,h}$, where $1 \leq j \leq M$ and $1 \leq h \leq k$. At each of the $k$ levels,
MERGE circuit MERGE$_{j,h}$ receives the outputs of sorted records from the SORT
circuit (sorted on $g_{i,h}$), it then merges these records with the memory record $MR_j$ on tag
g_{i,h} and l_{j,h}. The merge order depends on the selection operator of the algorithmic
problem given. Selection operators $<, \leq, =, \geq, >, \neq$ decide the order of the merge as
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

ascending, ascending, equal, descending, descending and non-equal respectively. The
ascending and descending orders are obvious. Equal order means that at the end of the
merge, the records will be ordered such that all the records with tag \( g_{i,h} \) equal to limit
\( l_{j,h} \) will be placed in front of the memory record holding the limit \( l_{j,h} \). Non-equal order
means that all the records with tag \( g_{i,h} \) not equal to limit \( l_{j,h} \) will be placed in front of the
memory record holding the limit \( l_{j,h} \). In the case where tags \( g_{i,h} \) are equal, datum \( d_i \) will
be used to order the records. If the tags \( g_{i,h} \) and limit \( l_{j,h} \) are equal, then we will order the
equal records according to the selection operator of the problem given. We will sort the
equal processor records (the ones with tag \( g_{i,h} \)) in front of the memory record (the ones
with limit \( l_{j,h} \)) in the case when the selection operator is \( \leq \) or \( \geq \), and have the memory
record placed in front of the processor records if the selection operator is \( < \) or \( > \).

Step 3 : FILTER circuit

There are \( k \) levels of \( M \) FILTER circuits in total arranged in parallel in the full \( k \)-criteria
BSR, with each FILTER circuit connected to each of the MERGE circuits from the
previous stage. FILTER circuit \( \text{FILTER}_{j,h} \) receives the outputs of sorted records (\( N \)
processor records and 1 memory record) from MERGE circuit \( \text{MERGE}_{j,h} \). It then filters
the records on \( d_i \) and \( v_j \) to extract the set of records from the first one up to the memory
record in order of input. This set of output records are the set of eligible records that
satisfy the \( h \) th criterion for the \( j \) th record, by using the limit \( l_{j,h} \) provided by memory
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

location \( MR_j \).

**Step 4 : MERGE circuit**

The \( M \) MERGE circuits here receive records output from all levels of FILTER circuits and merge them on index \( i \) and \( j \), with \( \text{MERGE}_j \) responsible for merging the outputs from \( \text{FILTER}_{j,h} \) for \( 1 \leq h \leq k \). Conceptually, each MERGE circuit produces a union of \( k \) set of records in sorted order, each of which satisfies exactly one of the \( k \) criteria for the \( j \)th record.

**Step 5 : EQUAL circuit**

There are \( M \) EQUAL circuits in total, each EQUAL circuit is connected to each of the MERGE circuits from the previous stage. The \( DNum \) of the EQUAL circuits are first preset to \( k \) in this case (see section 4.2). The EQUAL circuit \( \text{EQUAL}_j \) receives records from MERGE circuit \( \text{MERGE}_j \), checks the record index field \( i \) or \( j \) to find those records which have \( DNum \) copies as input, and then returns all the first copies of these duplicate records as output. Here, EQUAL circuit \( \text{EQUAL}_j \) is used to produce the set of records that satisfy all \( k \) criteria for the \( j \)th record.

**Step 6 : PREFIX circuit**

There are \( M \) PREFIX circuits in total, each PREFIX circuit is connected to each of the EQUAL circuits from the previous stage. The PREFIX circuit \( \text{PREFIX}_j \) receives records
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

from EQUAL circuit EQUAL\textsubscript{j}. It then does a prefix computation on datum d\textsubscript{i} according to the reduction rule \textsubscript{R} of the computational problem given. Here, the reduction result is sent at the same time to the memory record MR\textsubscript{j} which is located last among the inputs. Finally, the memory records from each PREFIX circuit are returned, with the data fields v\textsubscript{j} overwritten by the BSR result for the j\textsubscript{th} record.

4.4 Cost analysis of the implementation of the k-criteria BSR circuit

Since the FILTER circuit has a width of \( O(N) \) and a depth of \( O(\log N) \), and the EQUAL circuit has a width of \( O(N) \) and depth of \( O(k) \), by using any SORT, MERGE and PREFIX circuits that have a width of \( O(N) \) and depth of \( O(\log N) \) \([2]\,[21]\,[23]\), we obtain the following cost of implementation for each of the following steps:

**Step 1**: SORT circuit

Since there are a total of k SORT circuits, with each receiving \( N \) records as input, we have an overall width of \( O(kN) \), and a depth of \( O(\log N) \).

**Step 2**: MERGE circuit

Since there are a total of \( M \) MERGE circuits for each of the k levels, with each receiving \( N \) records as input, we have an overall width of \( O(kMN) \), and a depth of \( O(\log N) \).
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

Step 3: FILTER circuit

The cost here is the same as the previous MERGE circuit step. Since there are a total of $M$ FILTER circuits for each of the $k$ levels, with each receiving $N$ records as input, we have an overall width of $O(kMN)$, and a depth of $O(\log N)$.

Step 4: MERGE circuit

Since there are a total of $M$ MERGE circuits, with each receiving at most $kN$ records as input, we have an overall width $O(kMN)$, and depth of $O(\log N)$.

Step 5: EQUAL circuit

Since there are a total of $M$ EQUAL circuits, with each receiving at most $kN$ records as input, we have an overall width of $O(kMN)$, and a depth of $O(k)$.

Step 6: PREFIX circuit

Since there are a total of $M$ PREFIX circuits, with each receiving at most $N$ records as input, we have an overall width of $O(MN)$, and a depth of $O(\log N)$.

Therefore, the total width and depth of the implementation of the alternative k-criteria BSR is $O(kMN)$ and $O(\log N)$ respectively (consider that in most cases, $\log N$ is generally greater than the constant $k$), resulting in an overall size of $O(kMN\log N)$.

Assuming $N = O(M)$, the overall size of our alternative implementation of k-criteria BSR model becomes $O(kM^2\log M)$.
Although the size of our implementation is larger by a factor of \( \log N \) than that of the implementations described in chapter 3, the present implementation is easier and more expandable than the previous one because of its modular nature. For example, if we add one more selection criterion to a problem, our implementation will only need one more layer of circuits in the first three steps (i.e. one more SORT circuit in step 1, \( M \) more MERGE circuits in step 2, and \( M \) more FILTER circuits in step 3), whereas the implementations of chapter 3 require the addition of gates to each individual switch. In the case where there is an increase in memory size, here only one more set of circuits is necessary to add below the existing circuits in steps 2–6 (i.e. one more set of \( k \) MERGE circuits in step 2, one more set of \( k \) FILTER circuits in step 3, one more MERGE circuit in step 4, one more EQUAL circuit in step 5, and one more PREFIX circuits in step 6), whereas in the implementations of chapter 3, the CATs have to be redesigned and buses connecting each processor to the new memory location have to be constructed. In addition, the memory access time for the design proposed in this chapter is much better than the one suggested previously. Here, the depth of the circuit is \( O(\log N) \) which is more efficient than that of the distributed memory access model (with depth of \( O(N) \)), or that of the shared memory access model (with a depth of \( O(\log M) \)) described in chapter 3.
CHAPTER 4  A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

4.5 Example of a k-criteria BSR problem

To illustrate how the alternative k-criteria BSR implementation works, we consider a problem which can be solved by a two-criteria BSR model. The Empirical Cumulative Distribution Function (ECDF) searching problem consists of computing for each point \( p \) on the cartesian plane, \( p \in S \), the number \( E(p, S) \) of points in \( S \) dominated by \( p \). Let \( x[p] \) and \( y[p] \) be the x and y coordinates of point \( p \). Point \( p_1 \) dominates point \( p_2 \) if \( x[p_2] < x[p_1] \) and \( y[p_2] < y[p_1] \).

A BSR solution to this problem is given in \(^{13} \) as follows:

\[
E(p, S) \leftarrow \sum_{p \in S} 1 \mid (x[p_1] < x[p_2] \land y[p_1] < y[p_2])
\]

Suppose there are 5 points in \( S \), so in this case, \( N = M = 5 \). The five points and their respective processor records with layout \( (i, g_{i,1}, g_{i,2}, d_i) \) are:

Point 1: \((6, 5)\) record: \( (1, 6, 5, 1) \)
Point 2: \((7, 2)\) record: \( (2, 7, 2, 1) \)
Point 3: \((9, 6)\) record: \( (3, 9, 6, 1) \)
Point 4: \((1, 1)\) record: \( (4, 1, 1, 1) \)
Point 5: \((4, 3)\) record: \( (5, 4, 3, 1) \)

while the five memory records with layout \( (j + N, l_{j,1}, l_{j,2}, v_{j+N}) \) are:

\((6, 6, 5, v_6), (7, 7, 2, v_7), (8, 9, 6, v_8), (9, 1, 1, v_9), (10, 4, 3, v_{10})\)
\textit{CHAPTER 4 \ A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR}

Step 1: SORT circuit

There are two layers of SORT circuit at the first step (because \( k = 2 \)). Sort circuit \text{SORT}_1 \) receives the 5 records from the processors and sorts the records on tag \( g_{i,1} \), resulting in the order of \( (4, 1, 1, 1), (5, 4, 3, 1), (1, 6, 5, 1), (2, 7, 2, 1), (3, 9, 6, 1). \)

Sort circuit \text{SORT}_2 \) receives the 5 records from the processors and sorts the records on tag \( g_{i,2} \), resulting in the order of \( (4, 1, 1, 1), (2, 7, 2, 1), (5, 4, 3, 1), (1, 6, 5, 1), (3, 9, 6, 1). \)

Step 2: MERGE circuit

There are two layers of 5 MERGE circuits at this second step (because \( k = 2 \) and \( M = 5 \)). MERGE circuits \text{MERGE}_1,1 \) and \text{MERGE}_1,2 \) receive memory record \text{MR}_1 \) \((6, 6, 5, v_6)\) from memory location \( U_1 \). MERGE circuits \text{MERGE}_2,1 \) and \text{MERGE}_2,2 \) receive memory record \text{MR}_2 \) \((7, 7, 2, v_7)\) from memory location \( U_2 \), ..., MERGE circuits \text{MERGE}_5,1 \) and \text{MERGE}_5,2 \) receive memory record \text{MR}_5 \) \((10, 4, 3, v_{10})\) from memory location \( U_5 \).

At the same time, on both layers, all 5 MERGE circuits receive all sorted records from the output of the previous SORT circuits, with \text{MERGE}_{j,1} \) \((1 \leq j \leq 5)\) receiving sorted records from \text{SORT}_1, \) and \text{MERGE}_{j,2} \) \((1 \leq j \leq 5)\) receiving sorted records from \text{SORT}_2, \)

At \text{MERGE}_{1,1}, \) by merging on tag \( g_{i,1} \) for all sorted input records from \text{SORT}_1 \) circuit and the memory record \((6, 6, 5, v_6)\), we obtain the following records as returned from the circuit:
CHAPTER 4 A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

(4, 1, 1, 1), (5, 4, 3, 1), (6, 6, 5, v_6), (1, 6, 5, 1), (2, 7, 2, 1), (3, 9, 6, 1)

At MERGE_2, by merging on tag g_{i,1} for all sorted input records from the SORT_1 circuit and the memory record (7, 7, 2, v_7), we obtain the following records as returned from the circuit:

(4, 1, 1, 1), (5, 4, 3, 1), (1, 6, 5, 1), (7, 7, 2, v_7), (2, 7, 2, 1), (3, 9, 6, 1)

At MERGE_1, by merging on tag g_{i,2} for all sorted input records from the SORT_2 circuits and the memory record (6, 6, 5, v_6), we obtain the following records as returned from the circuit:

(4, 1, 1, 1), (2, 7, 2, 1), (5, 4, 3, 1), (6, 6, 5, v_6), (1, 6, 5, 1), (3, 9, 6, 1).

This process is similarly done in all the other MERGE circuits in both layers.

Step 3: FILTER circuit

Similar to the MERGE circuits in step 2, there are two layers of 5 FILTER circuits at this step. FILTER circuits FILTER_{j,h} (1 \leq j \leq 5, 1 \leq h \leq 2) receive records from the previous step MERGE_{j,h}, and then filter the record on the data field by checking the location of the memory record. At the end of this step, FILTER_{1,1} returns the records:

(4, 1, 1, 1), (5, 4, 3, 1), (6, 6, 5, v_6)

FILTER_{2,1} returns the records:

(4, 1, 1, 1), (5, 4, 3, 1), (1, 6, 5, 1), (7, 7, 2, v_7)

and FILTER_{1,2} returns the records:
(4, 1, 1, 1), (2, 7, 2, 1), (5, 4, 3, 1), (6, 6, 5, v₆)

This process is similarly done in all other FILTER circuits in both layers. Steps 1 to 3 are illustrated in figure 14.

Figure 14: An example of k-criteria BSR (Steps 1 to 3)
CHAPTER 4   A ALTERNATIVE IMPLEMENTATION OF THE K-CRITERIA BSR

Step 4: MERGE circuit
At this step, records output from both layers of five FILTER circuits are merged at the five MERGE circuits. For example, MERGE circuit MERGEi receives both sets of output records from FILTERi,1 and FILTERi,2 from the previous step, merges them on index i or j to form the records in the following order:

(2, 7, 2, 1), (4, 1, 1, 1), (4, 1, 1, 1), (5, 4, 3, 1), (5, 4, 3, 1), (6, 6, 5, v6), (6, 6, 5, v6)

Step 5: EQUAL circuit
At this step, EQUAL circuit EQUALj receives records from the previous step MERGE circuit MERGEj, 1 ≤ j ≤ 5, and returns the records which have exactly two copies (because k = 2). For example, EQUALi receives output records from the previous step and returns the following records:

(4, 1, 1, 1), (5, 4, 3, 1), (6, 6, 5, v6)

Step 6: PREFIX circuit
Finally, the PREFIX circuits perform a prefix sum on the datum field di for all the records received, and return only the memory records. Therefore PREFIX1 returns the memory record (6, 6, 5, 2). The completed set of the output records from the five PREFIX circuits are: (6, 6, 5, 2), (7, 7, 2, 1), (8, 9, 6, 4), (9, 1, 1, 0), (10, 4, 3, 1).
The final three steps of the example are shown in figure 15.

Figure 15: An example of k-criteria BSR (Steps 3 to 6)
Chapter 5

BSR Algorithms

BSR algorithms have been designed to solve different computational problems in
constant time\(^9\)\(^{10}\)\(^{13}\)\(^{14}\)\(^{18}\)\(^{23}\)\(^{26}\). To design a BSR algorithm for a given problem, we
have to decide which variables to choose as the tag \(g_i\), datum \(d_i\), and limit \(l_j\) of the records.

Selection rule(s) \(\sigma\) and reduction rule(s) \(\mathcal{R}\) should also be decided for the computational
problem addressed. The efficiency of an algorithm is measured by its cost, which is the
upper bound on the total number of elementary steps executed by the algorithm, defined
as the product of its running time and the number of processors it uses. In order to
illustrate the power and elegance of the BSR model in solving computational problems, a
sample BSR algorithm for the classical Sorting Problem is outlined below\(^{10}\). The
algorithm uses a number of processors that is linear in the size of the computational
problem and yields a constant time solution.

**Sorting Problem** :

Given an array of elements \(d_1, d_2, \ldots, d_N\), it is required to rearrange the elements into a
sequence \(s_1, s_2, \ldots, s_N\) such that these elements are sorted in non-decreasing order. The
BSR algorithm below uses \(N\) processors and \(N\) memory locations. It first counts for each
CHAPTER 5  BSR ALGORITHMS

d_i how many numbers are smaller in order to determine the rank of d_i. It then transfers
each element d_i to its final position, taking care of the non-unique element case.

Algorithm BSR SORT

Step 1: Compute the rank for each element d_k, store the rank in r[k].

for k = 1 to N do in parallel
    r[k] ← 0
    for i = 1 to N do in parallel
        r[k] ← \sum_{d_i<d_k} 1
    end for.
end for.

Step 2: Transfer each element to its final position, the final sequence output as s_1, s_2, ..., s_N.

for k = 1 to N do in parallel
    r[k] ← r[k] + 1

    for i = 1 to N do in parallel
        s_k ← \bigcap_{d_i \in r[i]=k} d_i
    end for.
end for.  ■
CHAPTER 5  BSR ALGORITHMS

This algorithm uses $O(N)$ processors. Since both step 1 and step 2 are $O(1)$ time steps, this algorithm runs in constant time, and has an overall running cost (i.e. number of processors multiplied by running time) of $O(N)$. Notice that since the time required for memory access $\tau_a(N, M)$ on BSR is considered to be $O(1)$, instead of $O(\log N)$, in keeping with the assumption made for the PRAM, the cost is $O(N)$ instead of $O(N \log N)$. Hence, the lower bound of $\Omega(N \log N)$ on the number of comparisons required in the worst case to solve the sorting problem is not violated here.

Other computational problems solved on 1-criterion BSR using a number of processors that is linear in the size of each problem include:

- Computational Geometry Problems $^{[14]}$:
  
  Interval Problem, and Convex Polygon Intersection Problem.

- Tree Problems $^{[13]}$ :
  
  Binary Tree Decoding, Generating and Reconstruction Problem.

- Image Processing/ Digital Geometry Problems $^{[18][22]}$ :
  
  Line Drawing, Histogramming, and Distance Transformation Problem.

- Other Computational Problems $^{[9][10][18]}$ :
  
  Prefix Sum, Element Uniqueness, Sieve of Eratosthenes, Maximal Subsequences, Maximal Vectors Finding, Job Sequencing Problem, a variant of the Knapsack Problem and, Maximal Sum Subsegment Problem.
CHAPTER 5  BSR ALGORITHMS

There are also algorithms designed on 1-criterion BSR which use a number of processors quadratic in the size of each problem; the problems solved by these algorithms include:

- **Computational Geometry Problems**\(^{[10][18]}\):
  - Convex Hull, Triangulation Of Points in the Plane, Closest Pair in the Plane.

- **Number Sequence Problems**\(^{[26]}\):
  - Longest Common Substring, and Maximum Sum Subarray Problem.

As mentioned earlier, apart from 1-criterion selection, BSR can also solve problems based on multiple criteria selection as well. Let us look at an example of BSR solving the **Counting Inversions in a Permutation** problem\(^{[13]}\) which uses a 2-selection criteria BSR instruction to achieve a constant-time solution.

**Counting Inversions in a Permutation Problem:**

Given a permutation \(\pi(1), \pi(2), \ldots, \pi(n)\) of the numbers 1, 2, ..., n, the problem is to count the number of inversions, which is the number of pairs \((i, j)\) with \(i < j\) but \(\pi(i) > \pi(j)\).

**Algorithm** COUNTING INVERSIONS IN A PERMUTATION

**Step 1:** We compute the number of inversions for each element, and store the number in \(y[k]\).

\[
\text{for } k = 1 \text{ to } N \text{ do in parallel} \\
y[k] \leftarrow \sum_{1 \leq i < N} 1 \mid k < i \land \pi(k) > \pi(i)
\]

end for.
CHAPTER 5  BSR ALGORITHMS

Step 2: Sum up the total number of inversions.

\[
\text{TotalInversions} \leftarrow \sum_{1 \leq i \leq N} y[i]
\]

Other algorithms designed on 2-criteria BSR using processors whose number is linear in the size of problems include the construction of the Voronoi diagram \(^{[14]}\), and all nearest and farthest neighbour in L₁ metric computation problem \(^{[22]}\). As for the 3-criteria BSR model, algorithms designed for this model include the vertical segment visibility problem \(^{[14]}\), all nearest and furthest foreign neighbour problem \(^{[13]}\), and medial axis transformation problem \(^{[22]}\). These algorithms also achieve constant time complexity using processors whose number is linear in the size of the problem. A number of algorithms have also been designed to solve computational problems on the k-criteria BSR model \(^{[14]}\) in constant time; these problems include:

- Maximal vector with in k dimensions
- Empirical cumulative distributive function ECDF searching in R^k space
- Two set dominance counting in R^k space
- Isothetic Line/Rectangle intersection counting in R^k space
- Rectangle enclosure counting in R^k space
- Rectangle containment counting in R^k space
5.1 Finding non-intersecting straight line connections of grid points to the boundary problem

The Finding Non-intersecting Straightline Connections of Grid Points to the Boundary Problem is defined as follows: Consider an mxn rectangular grid (i.e. a rectangular grid formed by m+1 vertical lines and n+1 horizontal lines), and N data points on the grid, where \( N \leq 2(m+n-2) \), determine whether it is possible to connect each data point to the grid boundary using a straight line (horizontal or vertical) such that no two straight lines intersect; if this is possible, provide such set of connections. Figure 16 shows a legal assignment satisfying such requirement. Sequential solutions to the problem with \( O(m+n) \) and \( O(N\log N) \) time complexity are given in [{16}]. Applications of this problem include network reliability [{16}] and solving the single-track switches Rectangular Array problem [{20}].

Each boundary exit is labelled in the clockwise direction starting from the top left corner as \( e_1, e_2, \ldots, e_{2(m+n-2)} \). A grid point is defined as a boundary grid point with respect to a particular boundary edge if it can be connected to the grid boundary using a straight line without crossing any other grid points. For example, in figure 16, grid points Q, A, O, B, N, D, L, F, G, and T are boundary grid points with respect to boundary A; and F, D, A, G, H, T, M, J, R, K, and L are boundary grid points with respect to boundary B. A grid
point is left-internal with respect to a particular boundary edge if it is a boundary grid point of that particular boundary edge and not a boundary grid point with respect to its counterclockwise neighbor boundary.

Figure 16: Non-intersecting straight line connections of grid points to the boundary problem
In figure 16, grid point D is left-internal with respect to boundary edge A, and grid points H, M, J, R, and K are left-internal with respect to boundary edge B. The leftmost-internal grid point of boundary edge X is the left-internal grid point of boundary X which has the smallest index i among all left-internal grid points of the same boundary which correspond to grid boundary exit e_i. In our example, grid point D is the leftmost-internal grid point of boundary A and grid point H is the leftmost-internal grid point of boundary B. A grid point is right-internal with respect to a particular boundary edge if it is a boundary grid point of that particular boundary edge and not a boundary grid point with respect to its clockwise neighbor boundary. In figure 13, grid points N, B, O and Q are right-internal with respect to boundary edge A, and grid points R, T, H, A, D, and F are right-internal with respect to boundary edge B. The rightmost-internal grid point of boundary edge X is the right-internal grid point of boundary X which has the largest index i among all right-internal grid points of the same boundary which correspond to grid boundary exit e_i. In our example, grid point N is the rightmost-internal grid point of boundary A and grid point R is the rightmost internal grid point of boundary B.

The basic idea behind the solution to the problem is as follows: for each boundary X, we look at all the points on the grid which can connect to X without crossing any point; we define the collection of these points as set S. For simplicity, suppose X is the top boundary of the grid. We identify if possible a top left corner rectangular region with diagonal defined by the top left corner grid point and one of the points from S such that
the points lying within this region cannot all legitimately connect to the right neighboring boundary of $X$ as exit, i.e. if we connect these points to the right neighboring boundary as exit connections, intersection of connections will definitely occur no matter how the assignments of connections are defined. We also identify if possible a similar region for the top right corner. Such regions are termed forbidden regions. Once the locations of the forbidden regions are known, we assign to those points located within the regions the boundary area where they can legitimately connect as exit. Finally, each point is forced to find a solution within the area in a unidirectional manner by looking for the closest allowable exit located to its right where it can connect. If each point can find a unique boundary exit, then a solution to the problem exists.

To solve this problem using BSR, after we number each boundary exit $e_1, e_2, \ldots, e_{2(m+n-2)}$ in the clockwise direction starting from the top left corner, for each boundary exit $e_i$, we select the possible grid point that can connect to $e_i$ as an exit. We do it by letting each boundary exit $e_i$ select all the points which appear on its grid line, and choose the one which is closest to itself. Then for each boundary edge $X$, we find the leftmost-internal grid point, store it in $\text{Lpoint}_X$ and set $i$, the index of the boundary exit $e_i$ it correspond to, as the left boundary limit of boundary edge $X$, store it in $\text{lblp}[X]$. For each boundary, we then locate the top right corner region described by $\text{Lpoint}_X$ for each boundary(shaded areas in figure 13), and set the left-allowable limit $l\text{alim}[p_i]$ of each point $p_i$ within the region to $\text{lblp}[X]$. For each boundary edge $X$, we then find the rightmost internal grid
point, store it in \( \text{Rpoint}_X \) and set index \( i \), the index of the boundary exit \( e_i \) it correspond to, as the right boundary limit of boundary edge \( X \), store it in \( \text{rblp}[X] \). For each boundary, we then locate the top left corner region described by \( \text{Rpoint}_X \) for each boundary, and set the right allowable limit \( \text{ralim}[p_i] \) of each point \( p_i \) within the region to \( \text{rblp}[X] \). Finally, for all grid points \( p_i \), we choose the boundary exit \( e_i \) (which has chosen itself as boundary grid point of boundary \( X \) in the first step) with smallest index \( i \) greater than \( \text{lblp}[X] \) and smaller than \( \text{rblp}[X] \) as the exit point. If all points are able to find a boundary exit, then such a set of connections is possible.

Let us summarize the BSR algorithm using the BSR notation:

**Algorithm** BSR CONNECTION OF GRID POINT TO BOUNDARY PROBLEM

**Step 1:** Each boundary exit \( e_k \) selects the closest grid point that can connect to \( e_k \) as an exit. This is done by first selecting all those grid points which have the same \( x \)-coordinate (or \( y \)-coordinate), then select the one which is closest. Store the point in \( (\text{closx}[k], \text{closy}[k]) \).

\[
\text{for } k = 1 \text{ to } 2(m+n-2) \text{ do in parallel} \\
\text{for } i = 1 \text{ to } N \text{ do in parallel} \\
\text{select case} \\
\text{case } y[e_k] = n \\
\text{closy}[k] \leftarrow \bigcap y[p_i] \mid x[p_i] = x[e_k]
\]
CHAPTER 5  BSR ALGORITHMS

\[
closx[k] \leftarrow x[e_k]
\]

\[
\text{case } x[e_k] = m
\]

\[
closx[k] \leftarrow \bigcap x[p_i] \mid y[p_i] = y[e_k]
\]

\[
closy[k] \leftarrow y[e_k]
\]

\[
\text{case } y[e_k] = 0
\]

\[
closy[k] \leftarrow \bigcup y[p_i] \mid x[p_i] = x[e_k]
\]

\[
closx[k] \leftarrow x[e_k]
\]

\[
\text{case } x[e_k] = 0
\]

\[
closx[k] \leftarrow \bigcup x[p_i] \mid y[p_i] = y[e_k]
\]

\[
closy[k] \leftarrow y[e_k]
\]

end select.

end for.

end for.

Step 2: Each boundary edge \( X \) selects the \textit{leftmost-internal boundary grid point}. This is done by first checking to see if each boundary grid point (from the previous step) is a \textit{left-internal boundary grid point}. Store TRUE to \text{leftflag}[k] if the grid point \((closx[k], closy[k])\) is a \textit{left-internal boundary grid point}, and FALSE otherwise. Then select the \textit{left-internal boundary grid point} by finding the one which corresponds to boundary exit \( e_i \) with the smallest index \( i \). Store the \textit{left}
boundary limit point in \((x[LPointX], y[LPointX])\) and its index \(i\) in \(lbp[X]\) for boundary \(X = \{A, B, C, D\}\).

for \(k = 1\) to \(2(n+m-2)\) do in parallel

for \(i = 1\) to \(N\) do in parallel

select case

case \(1 \leq k \leq m-1\)

\[\text{leftcount}[k] \leftarrow \sum 1 \mid \text{closy}[i] = \text{closy}[k] \land \text{closx}[i] < \text{closx}[k] \land 1 \leq i \land i \leq (m-1)\]

case \(m \leq k \leq (m+n-2)\)

\[\text{leftcount}[k] \leftarrow \sum 1 \mid \text{closx}[i] = \text{closx}[k] \land \text{closy}[i] > \text{closy}[k] \land m \leq i \land i \leq (m+n-2)\]

case \((m+n-1) \leq k \leq (2m+n-3)\)

\[\text{leftcount}[k] \leftarrow \sum 1 \mid \text{closy}[i] = \text{closy}[k] \land \text{closx}[i] > \text{closx}[k] \land (m+n-1) \leq i \land i \leq (2m+n-3)\]

case \((2m+n-2) \leq k \leq 2(m+n-2)\)

\[\text{leftcount}[k] \leftarrow \sum 1 \mid \text{closx}[i] = \text{closx}[k] \land \text{closy}[i] < \text{closy}[k] \land (2m+n-2) \leq i \land i \leq 2(m+n-2)\]

end select.

if \(\text{leftcount}[k] \neq 0\)

then \(\text{leftflag}[k] \leftarrow \text{TRUE}\)

else \(\text{leftflag}[k] \leftarrow \text{FALSE}\)
CHAPTER 5  BSR ALGORITHMS

end if.
end for.
end for.

for i = 1 to N do in parallel

X[Lpoint_a] ← ∪ closx[i] | 1 ≤ i ∧ i ≤ (m-1) ∧ leftflag[i] = TRUE

y[Lpoint_a] ← ∩ closy[i] | closx[i] = x[Lpoint_a]

lblp[A] ← i | closx[i] = x[Lpoint_a] ∧ closy[i] = y[Lpoint_a]

y[Lpoint_B] ← ∩ closy[i] | m ≤ i ∧ i ≤ (m+n-2) ∧ leftflag[i] = TRUE

x[Lpoint_B] ← ∩ closx[i] | closy[i] = y[Lpoint_B]

lblp[B] ← i | closx[i] = x[Lpoint_B] ∧ closy[i] = y[Lpoint_B]

x[Lpoint_C] ← ∩ closx[i] | (m+n-1) ≤ i ∧ i ≤ (2m+n-3)

∧ leftflag[i] = TRUE

y[Lpoint_C] ← ∪ closy[i] | closx[i] = x[Lpoint_C]

lblp[C] ← i | closx[i] = x[Lpoint_C] ∧ closy[i] = y[Lpoint_C]

y[Lpoint_D] ← ∪ closy[i] | (2m+n-2) ≤ i ∧ i ≤ (m+n-2)

∧ leftflag[i] = TRUE

x[Lpoint_D] ← ∪ closx[i] | closy[i] = y[Lpoint_D]
\textbf{CHAPTER 5} \textit{BSR ALGORITHMS}

\begin{equation}
\text{lbp}[D] \leftarrow i \mid \text{closx}[i] = x[\text{Lpoint}_D] \land \text{closy}[i] = y[\text{Lpoint}_D]
\end{equation}

\text{end for.}

\textbf{Step 3:} Each boundary edge $X$ selects the \textit{rightmost internal boundary grid point}. This is done by first checking to see if each boundary grid point (from step 1) is a \textit{right internal boundary grid point}. Store TRUE to rightflag[$k$] if the grid point ($\text{closx}[$k$], \text{closy}[$k$]) is a \textit{right internal boundary grid point}, and FALSE otherwise. Then select the \textit{right-internal boundary grid point} by finding the one which corresponds to boundary exit $e_i$ with the largest index $i$. Store the \textit{right boundary limit point} in ($x[R\text{point}_X], y[R\text{point}_X]$) and its index $i$ in $\text{rblp}[X]$ for boundary $X = \{A, B, C, D\}$.

\textbf{for} $k = 1$ to $2(n+m-2)$ \textbf{do in parallel}

\textbf{for} $i = 1$ to $N$ \textbf{do in parallel}

\textbf{select case}

\textbf{case} $1 \leq k \leq m-1$

\begin{equation}
\text{rightcount}[k] \leftarrow \sum 1 \mid \text{closy}[i] = \text{closy}[k] \land \text{closx}[i] > \text{closx}[k] \\
\land 1 \leq i \land i \leq (m-1)
\end{equation}

\textbf{case} $m \leq k \leq (m+n-2)$

\begin{equation}
\text{rightcount}[k] \leftarrow \sum 1 \mid \text{closx}[i] = \text{closx}[k] \land \text{closy}[i] < \text{closy}[k] \\
\land m \leq i \land i \leq (m+n-2)
\end{equation}

\textbf{case} $(m+n-1) \leq k \leq (2m+n-3)$
rightcount[k] ← \sum 1 | \text{closy}[i] = \text{closy}[k] \land \text{closx}[i] < \text{closx}[k] \\
\land (m+n-1) \leq i \land i \leq (2m+n-3)

\text{case } (2m+n-2) \leq k \leq 2(m+n-2) \\
rightcount[k] ← \sum 1 | \text{closx}[i] = \text{closx}[k] \land \text{closy}[i] > \text{closy}[k] \\
\land (2m+n-2) \leq i \land i \leq 2(m+n-2)

\text{end select.}

\text{if } \text{rightcount}[k] \neq 0 \\
\text{then } \text{rightflag}[k] ← \text{TRUE} \\
\text{else } \text{rightflag}[k] ← \text{FALSE} \\
\text{end if.}

\text{end for.}

\text{end for.}

\text{For } i = 1 \text{ to } N \text{ do in parallel}

\text{x}[\text{Rpoint}_A] ← \bigcap \text{closx}[i] | 1 \leq i \land i \leq (m-1) \land \text{leftflag}[i] = \text{TRUE}

\text{y}[\text{Rpoint}_A] ← \bigcap \text{closy}[i] | \text{closx}[i] = \text{x}[\text{Rpoint}_A]

\text{rblp}[A] ← i \mid \text{closx}[i] = \text{x}[\text{Rpoint}_A] \land \text{closy}[i] = \text{y}[\text{Rpoint}_A]

\text{y}[\text{Rpoint}_B] ← \bigcup \text{closy}[i] | m \leq i \land i \leq (m+n-2) \land \text{leftflag}[i] = \text{TRUE}

\text{x}[\text{Rpoint}_B] ← \bigcap \text{closx}[i] | \text{closy}[i] = \text{y}[\text{Rpoint}_B]

\text{rblp}[B] ← i \mid \text{closx}[i] = \text{x}[\text{Rpoint}_B] \land \text{closy}[i] = \text{y}[\text{Rpoint}_B]
\begin{align*}
x[Rpoint_{C}] & \leftarrow \bigcup \left\{ closx[i] \mid (m+n-1) \leq i \wedge i \leq (2m+n-3) \right\} \\
\text{letflag}[i] & = \text{TRUE} \\
y[Rpoint_{C}] & \leftarrow \bigcup \left\{ closy[i] \mid closx[i] = x[Rpoint_{C}] \right\} \\
\text{rblp}[C] & \leftarrow \{ i \mid closx[i] = x[Rpoint_{C}] \wedge closy[i] = y[Rpoint_{C}] \} \\
y[Rpoint_{D}] & \leftarrow \bigcap \left\{ closy[i] \mid (2m+n-2) \leq i \wedge i \leq 2(m+n-2) \right\} \\
\text{letflag}[i] & = \text{TRUE} \\
x[Rpoint_{D}] & \leftarrow \bigcup \left\{ closx[i] \mid closy[i] = y[Rpoint_{D}] \right\} \\
\text{rblp}[D] & \leftarrow \{ i \mid closx[i] = x[Rpoint_{D}] \wedge closy[i] = y[Rpoint_{D}] \} \\
\end{align*}

\textbf{Step 4}: Set left allowable limit \( \text{lalim}[p_i] \) for all grid point \( p_i \) if they fall into the top right corner region described by any \emph{leftmost internal grid point} \( (x[Lpoint_{x}], y[Lpoint_{y}]) \) of a boundary. If the point belongs to more than one corner region, then choose the one with the largest \( \text{lblp}[x] \), which is the index \( i \) of the boundary exit \( e_i \) to which the \emph{leftmost internal grid point} corresponds.

\begin{algorithm}
\textbf{for} \( k = 1 \) \textbf{to} \( N \) \textbf{do in parallel}

\begin{align*}
\text{lamin}[p_k] & \leftarrow 0 \\
\text{if} \ x[Lpoint_{A}] & \leq x[p_k] \wedge y[Lpoint_{A}] \leq y[p_k] \\
\text{then lamin}[p_k] & \leftarrow \text{lblp}[A] \\
\end{align*}

\textbf{end if.}
CHAPTER 5  BSR ALGORITHMS

if \( x[\text{Lpoink}_B] \leq x[pk] \land y[\text{Lpoink}_B] \geq y[pk] \)
then \( \text{lamin}[p_k] \leftarrow \text{rlbp}[B] \)
end if.

if \( x[\text{Lpoink}_C] \geq x[pk] \land y[\text{Lpoink}_C] \geq y[pk] \)
then \( \text{lamin}[p_k] \leftarrow \text{rlbp}[C] \)
end if.

if \( x[\text{Lpoink}_D] \geq x[pk] \land y[\text{Lpoink}_D] \leq y[pk] \)
then \( \text{lamin}[p_k] \leftarrow \text{rlbp}[D] \)
end if.

end for.

Step 5: Set right allowable limit \( \text{ralim}[p_i] \) for all grid point \( p_i \) if they fall into the top left corner region described by any *rightmost internal grid point point* \( (x[\text{Rpoint}_x], y[\text{Rpoint}_x]) \) of a boundary. If the point belongs to more than one corner region, then choose the one with the smallest \( \text{rlbp}[x] \), which is the index \( i \) of the boundary exit \( e_i \) to which the *rightmost internal grid point* corresponds.

for \( k = 1 \) to \( N \) do in parallel

\( \text{ramin}[p_k] \leftarrow 2(m+n-2)+1 \)

if \( x[\text{Lpoink}_D] \geq x[pk] \land y[\text{Lpoink}_D] \geq y[pk] \)
then \( \text{ramin}[p_k] \leftarrow \text{rlbp}[D] \)
end if.

if \( x[\text{Lpoink}_C] \leq x[pk] \land y[\text{Lpoink}_C] \geq y[pk] \)
CHAPTER 5  BSR ALGORITHMS

then \( \text{ramin}[p_k] \leftarrow \text{rblp}[C] \)

end if.

if \( x[\text{Lpoint}_B] \leq x[p_k] \land y[\text{Lpoint}_B] \leq y[p_k] \)
then \( \text{ramin}[p_k] \leftarrow \text{rblp}[B] \)
end if.

if \( x[\text{Lpoint}_A] \geq x[p_k] \land y[\text{Lpoint}_A] \leq y[p_k] \)
then \( \text{ramin}[p_k] \leftarrow \text{rblp}[A] \)
end if.

end for.

Step 6: Each grid point \( p_k \) selects a proper exit \( e_i \), which has chosen itself as boundary
grid point in Step 1, with \( i \) within the left and right allowable limit, \( \text{lalim}[p_k] \) and
\( \text{ralim}[p_k] \). Store \( i \) in \( \text{exit}[k] \).

for \( k = 1 \) to \( N \) do in parallel
for \( i = 1 \) to \( 2(n+m-2) \) do in parallel
\( \text{exit}[k] \leftarrow 0 \)

\( \text{exit}[k] \leftarrow \bigcup i \mid \text{ramin}[p_k] \leq i \land i \leq \text{ramin}[p_k] \)

\( \land \text{closx}[i] = x[p_k] \land \text{closy}[i] = y[p_k] \)

end for.
end for.

Step 7: If all points are able to find a boundary exit \( e_i \), then such a set of non-intersecting
connections is possible, with the boundary exit index of each point \( p_k \) stored in
CHAPTER 5  BSR ALGORITHMS

exit[k].

for $i = 1$ to $N$ do in parallel

\[ \text{TotalExit} \leftarrow \sum 1 \mid \text{exit}[i] \neq 0 \]

end for.

if \( \text{TotalExit} = N \)

then CONNECTION SOLUTION EXIST

end if.  

Step 6 can ensure one to one mapping of each grid point in choosing a boundary exit because we only allow each grid point to choose a boundary exit which has chosen itself earlier in step 1. Therefore, it is not possible for a boundary exit to be chosen by two different grid points at the same time.

The left allowable limit $l\text{lim}[p_i]$ and right allowable limit $r\text{lim}[p_i]$ are used to restrict the boundary area where the grid point $p_i$ can exit. In using these limits, when we are dealing with the grid points located in the top right corner region described by leftmost boundary point of boundary D, we actually mean that these points can choose any boundary exit with index greater than $l\text{bp}[D]$ AND possibly on boundary A. However, since the index $i$ of boundary exit $e_i$ on boundary A is less than that of $l\text{bp}[D]$, in our selection criteria of Step 6, we fail to allow the latter possibility from happening. To deal with this wrap-around case, we can either set a equivalent index of the boundary exits $e_1, e_2, \ldots, e_{m-1}$ of
boundary $A$ to $e_{2(m+n-2)+1}$, $e_{2(m+n-2)+2}$, ..., $e_{2(m+n-2)+m-1}$, or set a flag to indicate that these points can select exits with indices between 1 and $m-1$ as well. The same solution can be applied to the setting of the right allowable limit $r_{lim}[p_i]$ to grid points $p_i$ located in the top left region described by the rightmost internal point of boundary $A$. This algorithm can prevent straight line connections from intersecting each other because we are setting the allowable region of exit for each grid point in a one directional manner, we force the grid point to always find an exit to the right of the internal grid points, so if each grid point can find a boundary exit at the last step, then a solution to this problem exists.

**Analysis**  BSR CONNECTION OF GRID POINT TO BOUNDARY PROBLEM

The BSR algorithm suggested above uses $O(m+n)$ processors, since $N \leq 2(m+n-2)$. Constant time multiple criteria BSR operations apply in all steps 1 to 6. Step 7 is a combining write operation which takes constant time as well. Hence, the algorithm runs in $O(1)$ time with $O(m+n)$ processors, resulting in an overall cost of $O(m+n)$.

### 5.2 Variations of the knapsack problem

The Knapsack Problem is usually defined as follows: Given a capacity $K$, a collection $C$ of items and their associated weights and costs, find a subset of items from $C$ such that the total cost will not exceed $K$, and the total weight is maximized. Such problems can be solved using our BSR model\(^{18}\).
5.2.1 One dimensional variation of the knapsack problem

We now consider a one dimensional variation of the knapsack problem. Given a length of time $T$, a collection $C$ of items, their associated weights (possibly negative) $d_1, d_2, \ldots, d_N$, and the specific time stamp $t_1, t_2, \ldots, t_N$ at which each job is to be processed on an assembly line, select a subsequence of elements which maximize the weight sum of the elements it contains, with the total time required not exceeding $T$.

The solution to this problem is similar to the BSR solution to the maximal sum subsegment problem\(^9\) with an additional sorting step and a time length selection constraint imposed. Here, we first sort all the elements according to the time stamp $t_i$ at which the job needs to be processed, then we calculate the prefix sum $S_i$ for each element on the weights. For each $i^{th}$ element, we then find the largest prefix sum $M_j$ and its index to the right of the element which is within the next $T$ interval of time. Finally, for each element $i$, we calculate the sum of each subset $e[i] = M_j - S_i + d_i$. The subsequence with the largest weight not exceeding length $T$ is the sequence which has the largest $e[i]$, with starting position $i$ and ending position $j$.

To solve this problem using the PRAM, a similar algorithm can be used. However, since prefix computation with PRAM is performed in $O(\log N)$ time instead of $O(1)$, a total time of $O(\log N)$ and a cost of $O(M\log N)$ are required. Let us summarize the BSR algorithm using the BSR notation:
Algorithm BSR VARIATION OF ONE DIMENSIONAL KNAPSACK PROBLEM

Step 1: Sort all the items according to their time stamp $t_k$ using BSR SORT.

Step 2: Compute the prefix sum on the weights $d_i$, assign this to $a[k]$.

for $k = 1$ to $N$ do in parallel
    for $i = 1$ to $N$ do in parallel
        $a[k] \leftarrow \sum d_i \mid i \leq k$
    end for.
end for.

Step 3: Find the largest prefix sum and its index to the right of each element which is within a time length of $T$, assign this to $b[k]$ and $c[k]$ respectively.

for $k = 1$ to $N$ do in parallel
    for $i = 1$ to $N$ do in parallel
        $b[k] \leftarrow \cap a[i] \mid t_i \leq (t_k + T) \land i \geq k$
        $c[k] \leftarrow i \mid b[k] = a[i]$
    end for.
end for.

Step 4: Calculate the sum of each subset, assign it to $e[k]$.

for $k = 1$ to $N$ do in parallel
    $e[k] \leftarrow b[k] - a[k] + d_k$
end for.
**CHAPTER 5  BSR ALGORITHMS**

**Step 5:** Find the largest value among all subset weight sums, store this maximal weight value in MAX. The selected subsequence is represented by having the index of the starting element of the sequence identified by StartPos, and the index of the last element of the sequence identified by EndPos.

\[
\text{MAX } \leftarrow \cap e[i] \\
\text{StartPos } \leftarrow i \mid e[i] = \text{MAX} \\
\text{EndPos } \leftarrow c[i] \mid e[i] = \text{MAX} \]

**Analysis BSR VARIATION OF ONE DIMENSIONAL KNAPSACK PROBLEM**

The BSR algorithm here uses \( O(N) \) processors. Step 1 is a BSR sorting procedure which takes constant time \(^{[10]}\). Steps 2 and 3 are single criterion BSR instructions which take constant time. Step 4 is a constant time operation. Step 5 is a constant time maximum concurrent write instruction. Hence, the algorithm runs in \( O(1) \) time with a total cost of \( O(N) \). This is optimal because a lower bound of \( \Omega(N) \) is required to process all the given elements.

**5.2.2 Two dimensional variation of the knapsack problem**

A two dimensional version of the same problem can be solved using BSR. The problem is stated as follows: Given a square area \( A \), and a collection \( C \) of items and their associated weights (possibly negative) on a plane, select a subset of elements which maximize the sum of the weight of the elements it contains, with the area not exceeding square area \( A \). The solution of this problem follows closely with that of the maximal sum
CHAPTER 5  BSR ALGORITHMS

subrectangle problem [9], with additional steps which are similar to the ones modified for this algorithm.

5.3 Longest consecutively-identical subsequence

Given a sequence, \(d_1, d_2, \ldots, d_N\) we are asked to locate the longest consecutively-identical subsequence. To solve this problem using BSR, we can first compute for each element the difference \(\text{diff}\) between itself and the element preceding it. We then classify an element as a left point if \(\text{diff}\) is not equal to zero and \(\text{diff}\) of the succeeding element is equal to zero, and define an element as a right point if \(\text{diff}\) is equal to zero and \(\text{diff}\) of the succeeding element is not equal to zero. For each of the left points, by choosing the next nearest right point, we can retrieve the consecutive sequence starting from that left point. By using a similar approach, longest descending/ascending subsequences can be located using BSR. Here, we will keep track of the parity of the difference between each element and its preceding neighbor.

To solve the same longest consecutively-identical subsequence problem using the PRAM, we have to use a different approach, after classifying all the left points and right points, we set the records of each points as \((i, d_i, a_i, v_i)\) where \(i\) is the index of each element, \(d_i\) is the element, \(a_i\) is 1, and \(v_i\) is 1 if it is a left points, -1 if it is a right point and 0 otherwise. We then use interval prefix computation with operation \(0\) on the records \((i, d_i, a_i, v_i)\) \(0 (j, d_j, a_j, v_j)\) defined as follows:
CHAPTER 5  BSR ALGORITHMS

if \( v_j = 1 \) then \( a_j = 1 \)

else if \(( v_j < v_i \) ) then \(( a_j \leftarrow a_i + a_j \))

At the end of this computation, the length of all consecutive identical subsequences are stored in \( a_j \) of all right point records. Finally, by using maximum concurrent write, we can find the longest consecutively-identical subsequence. This algorithm takes \( O(\log N) \) time, which is optimal in view of the \( \Omega(\log N) \) lower bound on the time required to solve the prefix sum computation. Of course the same algorithm using prefix computation can be performed using BSR as well; in this case the BSR solution will only have a running time of \( O(1) \). Let us summarize the BSR algorithm using BSR notation:

Algorithm: BSR LONGEST CONSECUTIVELY-IDENTICAL SUBSEQUENCE

Step 1: Calculate the difference between each element and its preceding element, store the difference in \( \text{diff}[k] \).

for \( k = 1 \) to \( N \) do in parallel

if \( k = 1 \)

then \( \text{diff}[k] \leftarrow 1 \)

else \( \text{diff}[k] \leftarrow d_k - d_{k-1} \)

end for.

Step 2: Classify each element as either left or right point. We call the \( k \)th element a left point if \( \text{diff}[k] \) is not equal to zero and \( \text{diff}[k + 1] \) is equal to zero, and call it a right point if \( \text{diff}[k] \) is equal to zero and \( \text{diff}[k + 1] \) is not equal to zero. Store left point as value 1 and right point as value 2 in \( \text{ptype}[k] \).
CHAPTER 5  BSR ALGORITHMS

for $k = 1$ to $N$ do in parallel
    if $\text{diff}[k] \neq 0 \land \text{diff}[k + 1] = 0$
        then $\text{ptype}[k] \leftarrow 1$
    end if.
    if $\text{diff}[k] = 0 \land \text{diff}[k + 1] \neq 0$
        then $\text{ptype}[k] \leftarrow 2$
    end if.
end for.

Step 3: For all left points, find the closest right point to its right, store the index in $rseq[k]$, and the distance of this element from the left point in $len[k]$, which is also the length of each identical sequence.

for $k = 1$ to $N$ do in parallel
    for $i = 1$ to $N$ do in parallel
        if $\text{ptype}[k] = 1$ then
            $rseq[k] \leftarrow \bigcup i \mid \text{ptype}[i] = 2 \land i > k$
            $\text{len}[k] \leftarrow \text{req}[k] - k + 1$
        end if.
    end for.
end for.

73


**CHAPTER 5  BSR ALGORITHMS**

**Step 4:** Find the longest consecutively-identical subsequence, with the length of this subsequence stored in MaxLen, the index of the starting element of this sequence stored in StartPos and the index of the ending element stored in EndPos.

```
for i = 1 to N do in parallel
    MaxLen ← ∩ len[i] | ptype[i] = 1
    StartPos ← i | MaxLen = len[i]
    EndPos ← rseq[i] | MaxLen = len[i]
end for. ■
```

**Analysis BSR LONGEST CONSECUTIVELY-IDENTICAL SUBSEQUENCE**

The BSR algorithm here uses $O(N)$ processors. Steps 1 and 2 are constant time instructions. Step 3 is a 2-criteria BSR step which takes constant time. Step 4 is a maximum combining write constant time instruction. Hence, the algorithm has an $O(1)$ time complexity and a total cost of $O(N)$, which is optimal, because $\Omega(N)$ time is required to go through all the elements at least once.

**5.4 All point-pair distance problem**

Given a set of $N$ points on the plane, for all points $x_a, a \in \{1..N\}$, find the number of points which are closer to $x_a$ than $x_b, b \in \{1..N\}$. A parallel algorithm using BSR proceeds as follows. We first compute the distance between all pairs of points using
CHAPTER 5 BSR ALGORITHMS

$O(N^2)$ number of processors. Then we select all the points which form a shorter distance to $x_a$ than the distance between $x_a$ and $x_b$. Finally we reduce the selected data by counting them.

To solve the same problem using the PRAM, we first compute the distance between all pairs of points using $O(N^2)$ processors as in the case of the BSR algorithm. Then for each point $x_a$, we sort all the distances associated with it, i.e. the length $x_a x_i$, $i \in \{1..N\}$. The number of points which are closer to $x_a$ than $x_b$ would then be given by the rank of the length $x_a x_b$ in the $x_a$ set minus 1, this PRAM algorithm runs in $O(\log N)$ time due to the sorting procedure. Let us summarize the BSR algorithm using the BSR notation. For simplicity, let Distance($x, y$) be the procedure used to calculate the distance between point $x$ and point $y$.

Algorithm BSR POINT PAIR DISTANCE PROBLEM

Step 1: Each processor $P_k$ computes the distance between $x_i$ and $x_j$. Store the distance in $len[k]$.

for $k = 0$ to $N^2 - 1$ do in parallel

\[ len[k] \leftarrow \text{Distance}(x_k \text{DIV} N, x_k \text{MOD} N) \]

end for.

Step 2: Count the number of points which are closer to the point $x_r$ than to point $x_s$, and store it in $cnum[k]$. 

75
for k = 0 to $N^2 - 1$ do in parallel
for i = 0 to $N^2 - 1$ do in parallel

\[ cn[k] \leftarrow \sum 1 \mid len[i] < len[k] \land k \text{ DIV } N = i \text{ DIV } N \]

end for.
end for.

Analysis BSR POINT PAIR DISTANCE PROBLEM

The BSR algorithm uses $O(N^2)$ processors. Step 1 is a constant time instruction. The BSR instruction in Step 2 is a constant time step as well. So, the total cost of the algorithm is $O(N^2)$, which is optimal in the view of the lower bound $\Omega(N^2)$ to answer the query for all pairs of points given in the plane.

5.5 All point-pair inscribing problem

Given a set of $N$ points on the plane, for every point pair $p$ and $q$, find the total number of points which is inscribed by the rectangle which has $p$ and $q$ as the diagonal corners. To solve this problem using BSR, we can use $O(N^2)$ processors and memory locations, with each memory location representing each of the point pairs. Define x-coordinate and y-coordinate of a point $p$ as $x[p]$ and $y[p]$ respectively. For all the point pairs $p$ and $q$, by selecting all the points $r$ which satisfy the condition: $(\min(x[p], x[q]) \leq x[r] \leq \max(x[p], x[q]))$ and $(\min(y[p], y[q]) \leq y[r] \leq \max(y[p], y[q]))$, and counting them, we can get the total number of points which are inscribed by the rectangle which has $p$ and $q$ as the diagonal corners.
CHAPTER 5  BSR ALGORITHMS

To solve the same problem using the PRAM, we can use $O(N^2)$ processors. We first sort all the points in the horizontal direction. Then for each given pair of points ($N^2$ of them), we go through all the $N$ points and count the ones which are between them in the $x$ direction. This is an $O(N)$ time algorithm which uses $O(N^2)$ processors; therefore, the overall cost of this algorithm is $O(N^3)$.

Let us summarize the BSR algorithm using BSR notation:

Algorithm  BSR ALL POINT PAIRS INSCRIBING PROBLEM

Step 1: Each processor $P_k$, representing a pair of points $p_{k \text{ div } N}$ and $p_{k \text{ mod } N}$, finds the upper and lower bounds of $x$ and $y$ coordinates describing the rectangle formed by the two points. Store the upper bound and the lower bound of the $x$ and $y$ coordinates in $\text{UpperX}[k]$, $\text{LowerX}[k]$, $\text{UpperY}[k]$, and $\text{LowerY}[k]$ respectively.

for $k = 0$ to $N^2 - 1$ do in parallel

\[
\begin{align*}
\text{UpperX}[k] & \leftarrow \max(x[p_{k \text{ div } N}], x[p_{k \text{ mod } N}]) \\
\text{LowerX}[k] & \leftarrow \min(x[p_{k \text{ div } N}], x[p_{k \text{ mod } N}]) \\
\text{UpperY}[k] & \leftarrow \max(y[p_{k \text{ div } N}], y[p_{k \text{ mod } N}]) \\
\text{LowerY}[k] & \leftarrow \min(y[p_{k \text{ div } N}], y[p_{k \text{ mod } N}])
\end{align*}
\]

end for.

Step 2: Count the number of points which are inscribed by the rectangle formed by the two points each processor represents. Store the number in $\text{count}[k]$.

for $k = 0$ to $N^2 - 1$ do in parallel

for $i = 0$ to $N^2 - 1$ do in parallel
\textbf{CHAPTER 5  BSR ALGORITHMS}

\[
\text{count}[k] \leftarrow \sum 1 \mid \text{LowerX}[k] \leq x[p_i] \land x[p_i] \leq \text{UpperX}[k] \\
\land \text{LowerY}[k] \leq y[p_i] \land y[p_i] \leq \text{UpperY}[k]
\]
end for.
end for. 

\textbf{Analysis BSR ALL POINT PAIR INSCRIBING PROBLEM}

The BSR algorithm uses $O(N^2)$ processors. Step 1 is a constant time instruction. Step 2 is a multiple criteria BSR instruction running in constant time. Therefore, the total cost of the algorithm is $O(N^2)$, which is favorable in the view of the lower bound of $\Omega(N^2)$ on the time required to answer the queries for all pairs of points given in the plane.
6 Multiple-Criteria BSR Algorithms

In this section, we show how to solve several geometric problems in constant time using a multiple selection criteria BSR instruction. The common technique in solving the problems here is as follows: Upon selection of processor records, computations are performed separately on each of the selected records at the memory location, results of these computations are then combined to a final reduced value using the reduction rule \( R \). Unlike the previously discussed algorithms where reductions are done strictly on data provided directly from processor records, here, an extra step is processed on the selected records, and reductions are done on the results produced from this extra step. Problems solved using this technique include the Convex Hull problem, the Convex Polygon Intersection detection problem and the Shortest Distance between two Convex Polygons problem.

6.1 Convex Hull

The Convex Hull of a set of points \( S \) in the plane is the smallest convex polygon \( P \) that encloses \( S \). The Convex Hull problem with \( N \) vertices \( p_1, p_2, \ldots, p_N \) has been solved using
BSR with $O(N^2)$ processors in constant time \cite{10}. The main idea of this algorithm is based on the fact that a vertex $p$ is a convex hull vertex if the largest angle formed between the rays connecting $p$ and any other two adjacent vertices in the set is greater than $\pi$. The BSR algorithm is as follows:

**Algorithm BSR CONVEX HULL**

**Step 1:** Each processor, $P_k$, computes the angle made by the ray $p_ip_j$ with the positive $x$ -axis (where $i = k \text{ DIV } N$ and $j = k \text{ MOD } N$). Store the angle in $a[k]$ and store $i$ in $s[k]$.

for $k = 0$ to $N^2 - 1$ do in parallel

\[a[k] \leftarrow \text{AngleWithHorizontal}(p_k \text{ DIV } N, p_k \text{ MOD } N)\]

\[s[k] \leftarrow k \text{ DIV } N\]

end for.

**Step 2:** Sort the angles in order, first on the angles stored in $a[k]$, then on $s[k]$, using the **BSR SORT** algorithm \cite{10}.

**Step 3:** Compute the angles between adjacent rays from $p$ to other vertices in the set.

for $k = 0$ to $N^2 - 1$ do in parallel

if $(k \text{ MOD } N) \neq (N-1)$

then $a[k] := a[k+1] - a[k]$.

else $a[k] := a[k-N+1] - a[k] + 2 \pi$

end if.

end for.
CHAPTER 6  MULTIPLE-CRITERIA BSR ALGORITHMS

Step 4: Use a maximizing concurrent-write to determine the largest adjacent angle $m[k]$ for each point $p_k$, where $0 \leq k \text{ DIV } N \leq N - 1$. The point $p_i$ is a hull point if and only if the largest adjacent angle formed by joining $p_i$ to other vertices is greater than $\pi$.

for $k = 0$ to $N^2 - 1$ do in parallel
  if $k \text{ MOD } N = 1$
    then $m[k] \leftarrow \cap a[i] \mid s[i] = s[k]$
      if $m[k] \geq \pi$
        then $p_{s[k]}$ is a hull point.
      end if.
  end if.
end for. ■

This algorithm uses $O(N^2)$ processors and runs in constant time. However, the same performance can be obtained on a CRCW PRAM $^{[4]}$. Now, we propose another BSR solution to the same problem using a different approach. This time, we will use the multiple criteria BSR method, only a linear number of processors are required and the algorithm runs in constant time. The idea follows the solution suggested in $^{[6]}$ with modification to achieve constant time.

Let $P$ be the convex hull of a set $S$ of points in the plane, and let $L$ be the line connecting the leftmost and rightmost points of $S$. The upper convex hull of $S$ is that part of $P$ that lies above $L$. Similarly, the lower convex hull of $S$ is that part of $P$ that lies below $L$. 

81
CHAPTER 6 MULTIPLE-CRITERIA BSR ALGORITHMS

We compute the convex hull of a set of vertices \( p_1, p_2, \ldots, p_N \) by first finding the upper hull, followed by its lower hull computed in the same manner. The idea is as follows: for each point \( p_i \) of set \( S \), we find the point \( p_j \) to the right of \( p_i \) which makes the largest angle with the horizontal, and conclude that any point of \( S \) which does not fall below any segment \( (p_i, p_j) \) is a corner of the upper hull.

Algorithm BSR UPPER CONVEX HULL

Step 1: Sort the set \( S \) of \( N \) vertices \( p_1, p_2, \ldots, p_N \) by their x-coordinate using BSR SORT.

Step 2: For each point \( p_i \) of set \( S \), we look at all the points to the right of \( p_i \), find the point \( p_j \) such that the line supporting the segment \( (p_i, p_j) \) forms the largest angle with the horizontal. Store the largest angle in \( a[k] \) and the index of \( p_j \) in \( j[k] \).

\[
\text{for } k = 1 \text{ to } N \text{ do in parallel} \\
\quad \text{for } i = 1 \text{ to } N \text{ do in parallel} \\
\quad \quad a[k] \leftarrow \bigcap \text{AngleWithHorizontal}(p_k p_i) \quad x[p_i] > x[p_k] \\
\quad \quad j[k] \leftarrow i \quad \text{AngleWithHorizontal}(p_k p_i) = a[k] \\
\quad \text{end for.} \\
\text{end for.}
\]
**CHAPTER 6  MULTIPLE-CRITERIA BSR ALGORITHMS**

**Step 3:** For each point $p_k$ of $S$, we check to see if it is below any line segment $(p_i, p_{j(i)})$, for $1 \leq i \leq N$; if not, then it is an upper hull point. We store the status of $p_k$ being an upper hull point in $b[k]$: $b[k]$ is initially set to zero for all points; if $p_k$ falls below a segment $(p_i, p_{j(i)})$, then $b[k]$ will be increased by 1. Finally, all points $p_k$ which have $b[k] = 0$ are upper hull points.

```plaintext
for k = 1 to N do in parallel
  for i = 1 to N do in parallel
    b[k] ← 0
    b[k] ← ∑ 1 | x[k] > x[p_i] ∨ x[k] < x[p_{j(i)}]
    if b[k] = 0
      then $p_k$ is a upper hull point
    end if.
  end for.
end for.
```

It is necessary to explain why Step 3 works. In Step 2, for each point $p_i$ of set $S$, we look at all the points to the right of $p_i$ and find the point $p_j$ such that the line supporting the segment $(p_i, p_j)$ forms the largest angle with the horizontal. Since the segment $(p_i, p_j)$ forms the highest elevation with the horizontal, any point $p$ which is between $p_i$ and $p_j$ in the $x$ axis direction (i.e. $x[p] > x[p_i] \land x[p] < x[p_j]$) must be located under the segment $(p_i, p_j)$. because if otherwise, segment $(p_i, p)$ would have been the one forming the largest angle with the horizontal.
CHAPTER 6  MULTIPLE-CRITERIA BSR ALGORITHMS

Analysis BSR UPPER CONVEX HULL

The BSR algorithm here uses $O(N)$ processors. BSR SORT in step 1 runs in constant time. The BSR instructions in steps 2 and 3 run in constant time as well. Therefore the BSR Upper Convex Hull algorithm achieves a constant time solution with a total cost of $O(N)$.

As a conclusion, with the additional step of computing the lower convex hull in a similar manner, we can find the convex hull of a set of $N$ vertices with an overall cost of $O(N)$ in $O(1)$ time.

6.2 Intersection of two convex polygons

Two polygons intersect if any edge of one polygon crosses an edge of the other polygon. In figure 17, the pair of polygons in case A and case B are considered non-intersecting, and the pair of polygons in case C is considered to intersect. The problem of determining whether two convex polygons intersect has been solved by BSR using $O(N)$ number of processors in constant time [14]. The idea is based on the slab method [25] modified with a parallel algorithm suggested for the hypercube model [27].
Here, we suggest an alternative solution to the problem that uses a multiple criteria BSR with a linear number of processors in the size of the input points and runs in constant time. To do this, we use a technique similar to that used to solve the Convex Hull problem in constant time with linear cost. This time, for each upper edge of polygon R on the upper hull, we mark all those vertices of polygon S which are underneath it. We repeat this check with each edge of polygon R on the lower hull, and mark all those vertices of polygon S which are above it. If there are some, but not all, vertices of polygon S marked twice, then we conclude that polygon R intersects polygon S. We
repeat the above procedure for polygon S on vertices of polygon R to ensure Case C of figure 17 is detected. Case C of figure 17 demonstrates a situation where a polygon can have all its vertices external to all the vertices of another polygon, yet the two polygons intersect.

Algorithm BSR CONVEX POLYGON INTERSECTION PROBLEM

Step1: Compute the leftmost $r_1$ and rightmost $r_w$ extreme point of polygon R by maximum concurrent write. We then locate and order the points on the upper edges of polygon R from left to right, $r_1$, $r_2$, ..., $r_w$. This is done as follows: for all the points $r_i$ of polygon R, we first compute the angle $A_i$ that the line $(r_1, r_i)$ made with the horizontal. Let the angle that the line $(r_i, r_w)$ made with the horizontal be $A_w$. Then we sort all the angle $A_i$ with value greater than or equal to $A_w$ using BSR SORT and store its corresponding points in $r_1$, $r_2$, ..., $r_w$.

Step2: Compute the angle $a[i]$ that each edge $(r_i, r_{i+1})$, on the upper hull of polygon R makes with the horizontal.

for $k = 1$ to $w - 1$ do in parallel

$a[k] \leftarrow \text{AngleWithHorizontal}(r_k, r_{k+1})$

end for.
Step 3: For each vertices $s_k$ of polygon $S$, we check to see if it is below any line segment $(r_i, r_{i+1})$. We store the status of $s$ being under any edge in $d[k]$: $d[k]$ is initially set to 0; if $s_k$ falls below any segment $(r_i, r_{i+1})$, then $d[k]$ will be increased by 1.

for $k = 1$ to $N$ do in parallel

for $i = 1$ to $N$ do in parallel

\[ d[k] \leftarrow 0 \]

\[ d[k] \leftarrow \sum 1 \mid x[s_k] > x[r_i] \land x[s_k] < x[r_{i+1}] \land \]

\[ \land \ \text{AngleWithHorizontal}(r, s_k) < a[i] \]

end for.
end for.

Step 4: Repeat Step 1 to Step 3 for the vertices on the lower hull of polygon $R$. This time, we sort them from right to left; for each vertex $s_k$ of polygon $S$, we check to see whether it is above any line segment $(r_i, r_{i+1})$, and store the status of $s_k$ being above any edge in $e[k]$.

Step 5: We check to see how many vertices $s_k$ of polygon $S$ have both $d[k]$ and $e[k]$ valued non-zero. If there are some, but not all $N$ of them, then we conclude that Polygon $R$ intersects polygon $S$.

for $i = 1$ to $N$ do in parallel

\[ \text{Total} \leftarrow \sum 1 \mid d[i] <> 0 \land e[i] <> 0 \]

if $\text{Total} <> 0 \land \text{Total} <> N$

then Polygon $R$ intersects Polygon $S$.  

87
end if.
end for.

Step 6: Repeat Steps 1 – 5 for polygon S on vertices of polygon R, to ensure Case C of figure 17 is detected.

Analysis BSR CONVEX POLYGON INTERSECTION PROBLEM

The BSR algorithm uses $O(N)$ processors. BSR SORT in Step1 runs in constant time. Step 2 is a constant time instruction. The multiple criteria BSR instruction used in Step 3 runs in constant time as well. Step 5 is a one step combining write instruction. Therefore the BSR Convex Polygon Intersection algorithm achieves a constant time solution with a total cost of $O(N)$.

In fact the same idea can be used to detect the containment of polygon like Case A in figure 17. This time, for each upper edge of polygon R on the upper hull, we mark all those vertices of polygon S which are underneath it. We repeat this check with each edge of polygon R on the lower hull, and mark all those vertices of polygon S which are above it. If all vertices of polygon S are marked twice, then we conclude that polygon R contains polygon S. We repeat the above procedure for polygon S on vertices of polygon R to detect the case where polygon S contains polygon R.
6.3 Shortest distance between two convex polygons

Given two non-intersecting convex polygons R and S, the shortest distance between them is defined as the shortest length of any segment joining vertex r of polygon R and a vertex s of polygon S. Two polygons intersect if any edge of one polygon crosses an edge of the other polygon. This problem can be solved in linear time using multiple criteria BSR with number of processors linear in the number of vertices. Let \( r_1, r_2, \ldots, r_N \) be the vertices of polygon R, and \( s_1, s_2, \ldots, s_N \) be the vertices of polygon S. The idea is as follows: For each vertex \( r \) of polygon R, we look at all the vertices of polygon S to its right, and select the one which form the shortest distances \( d[i] \) with it. We do this again with each vertex \( s_i \) of polygon S on vertices of polygon R and store the shortest distances in \( e[i] \). The shortest distance between the two polygons is the shortest length among all the \( d[i] \) and \( e[i] \).

Algorithm SHORTEST DISTANCE BETWEEN TWO CONVEX POLYGONS

Step 1: For each vertex \( r_k \) of polygon R, we look at all the vertices of polygon S to its right, select the one with the shortest distance, and store the value in \( d[k] \) and its index in \( g[k] \).

\[
\text{for } k = 1 \text{ to } N \text{ do in parallel}
\]
\[
\text{for } i = 1 \text{ to } N \text{ do in parallel}
\]

\[
d[k] \leftarrow \cup \text{ Distance}(r_k, s_i) \mid x[r_k] < x[s_i]
\]

\[
g[k] \leftarrow i \mid d[k] = \text{Distance}(r_k, s_i)
\]
CHAPTER 6  MULTIPLE-CRITERIA BSR ALGORITHMS

end for.
end for.

Step 2: For each vertex $s_k$ of polygon S, we look at all the vertices of polygon R to its right, select the one with the shortest distance, and store the value in $e[k]$ and its index in $h[k]$.  

for $k = 1$ to $N$ do in parallel
for $i = 1$ to $N$ do in parallel

\[
e[k] \leftarrow \bigcup \text{Distance}(s_k, r_i) \mid x[s_k] < x[r_i]
\]

\[
h[k] \leftarrow i \mid e[k] = \text{Distance}(s_k, r_i)
\]
end for.
end for.

Step 3: Find the shortest length among all the shortest distances. We first find the smallest value among the $d[i]$, and store it in $S_d$; we then find the smallest value among $e[i]$, and store it in $S_e$. The shortest distance between the two convex polygon is the smaller of $S_d$ and $S_e$.  

for $i = 1$ to $N$ do in parallel

\[
S_d \leftarrow \bigcup d[i]
\]

\[
S_e \leftarrow \bigcup e[i]
\]
end for.

ShortestDistance $\leftarrow \text{Shortest}(S_d, S_e)$
CHAPTER 6  MULTIPLE-CRITERIA BSR ALGORITHMS

Analysis BSR SHORTEST DISTANCE BETWEEN TWO CONVEX POLYGON

The BSR algorithm here uses $O(N)$ processors. The 2-criteria BSR instructions in Step 1 and Step 2 run in constant time. Step 3 is a constant time minimum concurrent write instruction. Therefore the BSR Shortest Distance between two Convex Polygons algorithm achieves a constant time solution with a total cost of $O(N)$. 
Chapter 7

Conclusions and Future Work

In this thesis, a survey was presented of various aspects concerning a powerful model of parallel computation, the Broadcasting with Selective Reduction (BSR) model, and a number of extensions of the model. The existing hypothetical implementations of the k-criteria BSR model using different memory architectures were considered. An alternative implementation of the model was proposed. In addition, we described new algorithms for solving different computational problems on BSR in constant time. These problems include the problem of finding non-intersecting straight line connections of grid points to the boundary, variations of the knapsack problem, the longest consecutively-identical subsequence problem, the all point-pair distance problem, and the all point-pair inscribing problem. All the algorithms given are more efficient than the best known PRAM algorithms for the same problems. A constant time algorithm was presented for solving the planar convex hull problem using a number of processors linear to the size of the input set of points. To our knowledge, all these algorithms are original and improve on the best previously known algorithms for the same problems.

So far, BSR can only handle problems whose data obey a linear order and consequently can be sorted. If the selection rule is not an order relation, then the memory access units
designed are not appropriate. The possibility of having an alternative design of the MAU which is capable of handling operators other than \{ <, \leq, =, >, \geq, \neq \} is one of the future directions to expand the model. As pointed out in \cite{10,13,14,21}, there are several other questions in connection with the BSR model which remain open. Apart from the selection and reduction operators defined, are there any other selection or reduction rules that we should add to the set? Can we design more BSR algorithms using the 1-criterion or k-criteria BSR model for other computational problems not addressed so far? To date, all the computational problems that are solvable efficiently on BSR are those that have all the information, required to produce each output, available at the start of the computation. Those problems which require intermediate results do not fall into this category; for example, graph problems such as coloring, determining a dominating set, computing minimum spanning trees, shortest paths, or connected components, and recursive matrix multiplication problems. Another research direction is to see whether these problems can be solved efficiently using BSR by either re-implementing the model in a different way, or re-structuring the model at the algorithmic level.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


