THREE ALGORITHMS FOR SELECTION ON THE RECONFIGURABLE MESH

Dipak Pravin Doctor and Danny Krizanc

TR-219, FEBRUARY 1993

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

Three Algorithms for Selection on the Reconfigurable Mesh

Dipak Pravin Doctor*
Computer Science
The University of Texas at Dallas
Richardson, TX 75083-0688
U.S.A.

Danny Krizanc†
School of Computer Science
Carleton University
Ottawa, Ontario K15 5B6
Canada.

Abstract

We present three algorithms for selection on the reconfigurable mesh (RMESH) model. The first, deterministic, algorithm runs in $O(blog^*N)$ time on an N processor RMESH, for N integers *each of length $\leq$ b-bits. The second, deterministic, algorithm runs in expected $O(logN)$ time on an N processor RMESH for N* elements *drawn from a linearly ordered universe. The third, randomized, algorithm runs in $O(log^2N)$ time, with high probability on an N processor RMESH again for N* elements *drawn from a linearly ordered universe. Our algorithms address three different scenarios and are important in view of a previous $O(log^2(N))$ algorithm [EW91] requiring complicated data movements operations.*

*e-mail: dipak@utdallas.edu, Phone: 716-626-5340, Post: 232 Belvoir Rd., Williamsville, NY-14221

†e-mail: krizanc@scs.carleton.ca, Phone: 613-788-2600 X 4359

1 Introduction

The problem of selection is to find the $k^{th}(1 \leq k \leq N)$ smallest element from a given set S of N elements drawn from a linearly ordered set [AHU74]. The problem of selecting k^{th}smallest element from a given set of elements can be solved by sorting the given set in non-decreasing order and then picking the k^{th}element from the top of the ordered list. The problem of selecting the middle element (k = $\lceil n/2 \rceil$) is knows as the *median* selection problem. The problem of finding the k^{th}largest element can be solved by picking the element starting from the bottom of the list which is sorted in the non-decreasing order. The selection problem has been widely studied in literature on several models of computation [CCC92, BFP+73, KN92, Raj90, EW91]. The reconfigurable mesh model of computation is becoming popular not only because of the absence of diameter consideration [BAPRS91], but also because of the existence of the experimental and commercial implementations [MKS89, TCS89, LS91]. One can solve the problem of selection in O(1) time by first sorting the given N elements in constant time [JPP92] and then picking the desired element. However, this approach will require an RMESH of $N \times N$ processors. In this paper, we present three algorithms which require an RMESH of only N processors for the given set of N elements.

In the next section we present the RMESH model and some previous results which will be used in our algorithms. Our deterministic, $O(blog^*N)$, algorithm is presented in section 3. In section 4, we present a second deterministic algorithm which runs in expected $O(logN)$ time. In section 5, we present a randomized algorithm which runs in $O(log^2N)$ time, with high probability. All three of our algorithms are simple to implement and require little if any data movement. Finally, in section 6 we give some concluding remarks as well as open questions raised by our results.

2 Preliminaries

2.1 Model Definition

A reconfigurable mesh (RMESH) of size N consists of N processing elements (p.e. for short), which are connected in a $\sqrt{N} \times \sqrt{N}$ grid with a p.e. at each grid point. Each p.e. is indexed by (row, column) number and each p.e.

knows it's own index. An RMESH can be thought of as a standard mesh with a programmable switch as part of each grid processor. We assume that, in one time unit, each p.e. is capable of (1) setting it's own local switch connection (for re-configuration), (2) sending (optionally) a message on the bus, (3) receiving (optionally) a message from the bus, and (4) executing $O(1)$ standard arithmetic/logic operations. A message is assumed to be of constant length based on the bandwidth of the network. Also each p.e. has $O(1)$ local memory registers.

Each p.e. is directly connected to it's four neighbors with the exception of the boundary processors. Each p.e. in an RMESH can connect up to three of it's neighbors, which makes a *bus* spanning across and through a p.e. Also each p.e. can be a part of up to two different buses. Note that a standard mesh can not make such a *bus* connection. For an RMESH one can easily conceive of all rows or all columns buses. If we denote the four neighbors of a p.e. as North(**N**), East(**E**), South(**S**) and West(**W**) then we can, for example, have connections like {NS, EW}, {WN, SE}, {WNE, S} as in [JPP92]. A standard mesh can be created by making {N, E, S, W}(no bus connection) switch setting. Even though a bus can have many receiving processors only one sender is allowed in a single time step. However, if there is more than one sender then all the p.e.s on that bus will detect the error condition.

The *unit-time delay model* assumes that a single data broadcast on the reconfigurable mesh takes $\Theta(1)$ time *regardless* of the size of the bus. In this paper, we only consider the unit-time delay model.

2.2 Previous Results

For the algorithms presented in this paper we will use the following variables, input characteristic and data operations;

S denotes the given set of elements from which we are required to select the k^{th}smallest element. $N = |S|$ is the number of elements in the set S. The first algorithm operates on b-bits long *distinct* integers, where $b \leq wordlength$ of the RMESH. The second and third, comparison based, algorithms operate on *any* elements which can be compared and manipulated by each p.e. of the RMESH in a constant number of time steps. We will use an RMESH of size N for both the algorithms. Since $N = |S|$, we will initially have one element of the set S stored per p.e. of the RMESH.

For all three algorithms, we assume the input elements to be distinct. This requirement can be obtained, as is customary, by representing an element $< e_i >$ located at j^{th} p.e. as $< e_i, j >$, if the inputs are not known to be distinct.

The following lemmas will be required in the proofs of our results. The problem of counting number of '1' bits in a sequence of N 0/1 bits–one bit per processor, has been addressed by Jang, Park and Prasanna [JPP92]. We state their conclusion in the following lemma;

Lemma 1 *The number of 1's in a $\sqrt{N} \times \sqrt{N}$ 0/1 table can be computed in $O(log^* N)$ time on an N processor RMESH.*

We note here that in the absence of above fast counting technique due to Jang, Park and Prasanna, the counting by usual summing would take $O(logN)$.

Random Access Read (RAR) and *Random Access Write* (RAW) [NS81] operations on a reconfigurable mesh have been studied by Miller, Prasanna-Kumar, Reisis and Stout [MPKRS87]. The following lemma is the consequence of one of their results;

Lemma 2 *$m \leq N$ data items may be routed in an arbitrary permutation in $O(\sqrt{m})$ steps on a N processor RMESH.*

Miller et al. [MPKRS87] also study the parallel prefix problem. They show;

Lemma 3 *Given a set $S = \{a_i\}$ of N values, distributed one per processor on a RMESH of size N so that P_i contains a_i, $0 \leq i \leq N-1$, and an $O(1)$ time binary associative operation $\otimes$, in $O(\log N)$ time, the parallel prefix problem can be solved so that each processor P_i knows $a_0 \otimes a_1 \otimes \cdots \otimes a_i$, $O \leq i \leq N-1$.*

Jang and Prasanna [JP92] have developed an efficient algorithm for sorting. We state the following simple corollary to their result without proof:

Lemma 4 *Elements stored in the first $r > 0$ rows of an N processor RMESH may be sorted in $O(r)$ time.*

We have the following approximations for a binomial distribution due to Chernoff [Che52]. A binomial distribution with n independent Bernoulli trials and with p probability of success in each trial is denoted by $B(n,p)$. (A *Bernoulli trial* is an experiment with only two possible outcomes, for example TRUE and FALSE.)

Lemma 5 *IF $B(n,p)$ is a binomial distribution and $\epsilon > 0$ then we have*
(1) *Probability* $(B \leq \lfloor (1-\epsilon)np \rfloor) \leq \exp(-\epsilon^2 np/2)$
(2) *Probability* $(B \geq \lceil (1+\epsilon)np \rceil) \leq \exp(-\epsilon^2 np/3)$

By high probability, we mean with probability $p \geq (1 - N^{-\alpha})$ for the given input size N and any $\alpha \geq 1$ [Raj90].

3 Deterministic Algorithm

This algorithm works on N integers each of which is b-bits long. The basic idea is similar to radix sorting [AHU74], where the sorting progresses in rounds from the *most significant* bit position to the *least significant* bit position and each sorting round considers one bit position. For b-bits long keys, the number of sorting rounds will be exactly b. Similarly, one can undertake binary radix selection, by, first, counting the number of '0's and '1's in the designated bit position and then selecting one of the two groups for further counting and selection on one less significant bit position. Since all the input integers are assumed to be distinct, the desired selection will be found within b rounds. It is easy to see that since, our b-bits long elements are drawn from the interval $[0, 2^b - 1]$, our binary selection procedure can be completed in $log(2^b) = b$ rounds. Details of our algorithm follow. (Comments are given in the curly brackets.)

Algorithm 1

Step 1: Proc SELECTION_1(S, N, b, k)

Step 2: {Initialize} Mark all elements active, let number of active elements E:=N and let j:= k.

Step 3: {For Loop} for i:= b *downto* 1 do

Step 4: {Extract a bit, Least Significant bit is 0^{th} bit.} Let CB:= i^{th} bit from each active element. For all inactive p.e. CB:=0.

Step 5: {Count '1' Bits} Let Y:=be the count of active elements with CB=1. Let X:= (E-Y) be the count of the rest of the active elements.

Step 6: {Mark Elements Inactive} If $j \leq X$ then mark all elements with CB=1 as inactive and E:= X; otherwise mark all elements with CB=0 as inactive, E:= Y; and j:=(j - X);

Step 7: {Check Termination.} If (E = 1) then mark the only active element as the desired selection and stop. Otherwise continue.

Step 8: endfor.

Step 9: endproc.

Since we assume distinct input elements, the correctness of the above algorithm can be easily proved by induction on the bit-length b'.

Theorem 1 *Selection of k^{th}, $1 \leq k \leq N$, smallest/largest element, from a set of distinct-integers, each of length $\leq$ b-bits, can done in $O(blog^*N)$ parallel time on an RMESH of size N.*

Proof. The for-loop in SELECTION_1 is executed $\leq b$ times. By lemma 1, it follows that each iteration in the for-loop takes at most $O(log^*(\sqrt{N}))$ $\leq O(log^*(N))$ steps. All other steps within the loop can be executed in $O(1)$ time on the RMESH. Thus we have an algorithm for selection with the claimed upper bound. □

We note here that (1) the above algorithm does not require *any* data movements operations and (2) it is practical and easy to implement.

4 Average-Case Algorithm

The deterministic algorithm presented here is a *comparison* based algorithm unlike the previous *radix* based algorithm. For this algorithm, we allow the input elements to be chosen from any set so long as two elements can be compared in a constant number of steps on a single p.e. of an RMESH. We

will show that the algorithm is efficient on *average*, i.e., for an input chosen uniformly at random among all possible input orders, the algorithm runs in $O(\log N)$ with high probability. However, in the worst case, the algorithm may required $O(\sqrt{N})$ time.

The basic idea of the algorithm is to, first, choose a sample R of $o(N)$ elements from the input set S and sort it. Now, let e be the element of rank $l = \lceil k(|R|/N) \rceil$ in R. For randomly chosen R, e is expected to be *close to* the element of rank k in set S. To eliminate most of the elements of S from further computation, we designate two elements e_1 and e_2 with ranks $l - \delta$ and $l + \delta$ in R respectively, where δ is a *small* integer, such that the ranks of e_1 and e_2, in S, are expected to be $< k$ and $> k$, respectively, with high probability. Next, we mark all the elements as being less than e_1, between e_1 and e_2 or greater than e_2. Depending on the sizes of the three sets we sort the set containing the element of rank k and we are done. Details of our algorithm follow.

Algorithm 2

Step 1: Proc SELECTION_2(S, N, k)

Step 2: { Sort a sample R } Sort the elements stored in the *first* $\log N$ rows of the RMESH.

Step 3: {Designate e_1 and e_2 } Designate as e_1 and e_2, respectively, the elements with ranks $(\lceil k \log N/\sqrt{N} \rceil - \log N)$ and $(\lceil k \log N/\sqrt{N} \rceil + \log N)$.

Step 4: {Mark, count and eliminate }Broadcast e_1 and e_2. Let A be the set of all elements strictly less than e_1, B, the set of all elements between e_1 and e_2 inclusive, and C, the set of all elements strictly greater than e_2. Using a prefix operation compute the sizes of sets A and B, denoted a and b, respectively. If $a > k$, mark elements in B and C as inactive. If $a \leq k \leq a + b$ mark elements in A and C as inactive. Otherwise, mark the elements in A and B as inactive.

Step 5: { Concentrate the active elements in the first rows } Using a prefix operation, rank the active elements in each column i of the RMESH and denote the element of maximum rank by c_i for $1 \leq i \leq$

$\sqrt{N}$. Move the elements one-by-one to the row of their rank using broadcasting in the column. Using another prefix operation, compute the maximum of all c_i's and call it c. Broadcast c. All processors in the first c rows not having an active element, create one with value $+\infty$ (i.e., a value greater than all possible values under consideration).

Step 6: {Sort and Select} Sort the first c rows of the RMESH. If $k \leq a$ then return the k^{th}element. If $a < k \leq a + b$ then return the $(k-a)^{th}$ element. Otherwise, return the $(k-a-b)^{th}$th element.

Step 7: endproc.

Since we assume distinct input elements, it is easy to see that the above algorithm performs the required selection.

Theorem 2 *Selection of k^{th}, $1 \leq k \leq N$, smallest/largest element, from a set of* distinct *elements, can be done in $O(logN)$ expected time on an RMESH of size N.*

Proof. The running time of SELECTION_2 is dominated by steps 5 and 6. Using lemmas 3 and 4 these can be shown to require $O(c + \log N)$ and $O(c)$ steps, respectively. All other steps are easily shown to require $O(\log N)$ steps (again using lemmas 3 and 4). Using lemma 5 we can show that if the input to the algorithm is random then $c = O(\log N)$ with high probability and the result follows. (Note that $c \leq \sqrt{N}$.) □

5 Randomized Algorithm

In this section we present our most general algorithm. We allow any input elements from a linearly ordered set so long as two elements can be compared in a constant number of steps on an RMESH. It is clear from the previous algorithm that the data movement operation is a *time consuming* operation on an RMESH, while broadcast and counting are very fast operations. We propose here a very simple and natural partitioning strategy which takes advantage of fast broadcast and counting operations on an RMESH. Details of our algorithm follow.

Algorithm 3

Step 1: Proc SELECTION_3(S, N, k)

Step 2: {Initialize} Mark all elements active, let number of active elements $E := N$ and let $j := k$.

Step 3: {Assign Rank} Assign rank to each active element in row-major order thru a prefix operation.

Step 4: {Pick a Partitioning Element.} Processor 1 selects random number, r, between 1 and E and broadcasts it. The element, m, of rank r is chosen to be the pivot element and the processor containing it broadcasts it.

Step 5: {Count and Eliminate} Count the number, c, of all elements $\leq m$. If $(j < c)$ then mark elements just counted as *active* and let $E := c$ else if $(j > c)$ then mark elements just counted as *in-active* and let $j := (E - c)$ and $E := E - c$ else declare m as the required element and stop.

Step 6: {Repeat} Repeat previous steps starting with step 3 until $E < \log^2 N$.

Step 7: {Concentrate and Sort} Concentrate the remaining elements on the first row of the mesh and sort them. Return the element of rank j.

Step 8: endproc.

Theorem 3 *Selection of k^{th}, $1 \leq k \leq N$, smallest/largest element, from a set of* distinct *elements, can be in done $O(log^2 N)$ parallel time, with high probability, on an RMESH of size N.*

Proof. It is easy to see that the partitioning algorithm correctly produces the k^{th}smallest element. Using lemma 5 it is easy to show the main loop of the algorithm will be performed $O(log(N))$ times with high probability. The most expensive component in the main loop is the prefix based renumbering of the remaining active elements, which requires $O(\log N)$ steps (lemma 3). By lemmas 2 and 4, step 7 can be completed in $O(\log N)$ time which leads to the run time claimed. □

6 Conclusion

In this paper, we presented three algorithms for the selection of k^{th}smallest or largest element on the Reconfigurable Mesh machine. All of our algorithms are simple and practical and they improve upon the best previous algorithms for the case of integers with fewer than $\log^2 N/\log^* N$ bits in the case of our first algorithm and on average in the case of our second algorithm. Further research is needed to address the issue of how to scale a selection algorithm to match the given combination of input set size and RMESH size. Another interesting issue is to find the least number of extra processors required to run the selection algorithm in $O(1)$ time. Also of interest are lower bounds for selection among N elements on a N processors RMESH.

References

[AHU74] Aho. A.V., J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[BAPRS91] A. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13:139–153, 1991.

[BFP+73] M. Blum, R. Floyd, V. R. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *Journal of Computer and System Science*, 7(4):448–461, 1973.

[CCC92] Y. Chen, W. Chen, and G. Chen. Application to two-variable linear programming on mesh-connected computers with multiple broadcasting. *Journal of Parallel and Distributed Computing*, 15:79–84, 1992.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, pages 493–507, 23(1952).

[EW91] H. ElGindy and P. Wegrowicz. Selection on the reconfigurable mesh. In *International Conference on Parallel Processing*, pages III, 26–33, Chicago, 1991.

[JP92] J. Jang and V. K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. In *International Parallel Processing Symposium*, pages 130–137, Beverly Hills, California, 1992.

[JPP92] J. Jang, H. Park, and V. K. Prasanna. A fast algorithm for computing histogram on reconfigurable mesh. Technical Report IRIS No. 290, Institute for Robotics and Intelligent Systems, Univ. of Southern California, Los Angeles, 1992.

[KN92] D. Krizanc and L. Narayanan. Multi-packet selection on mesh-connected processor arrays. In *International Parallel Processing Symposium*, pages 602–605, Beverly Hills, California, 1992.

[LS91] H. Li and Q. Stout. *Reconfigurable Massively Parallel Computers.* Prentice-Hall, Englewood Cliffs, New Jersy 07632, 1991.

[MKS89] O. Menzilcioglu, H. T. Kung, and S. W. Song. Comprehensive evaluation of a two-dimensional configurable array. In *Proceedings of 19th Symposium on Fault Tolerent Computing*, pages 93–100, Chicago, Illinois, June 1989.

[MPKRS87] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, and Q.F. Stout. Parallel computations on reconfigurable meshes. Technical Report IRIS No. 229, Institute for Robotics and Intelligent Systems Univ. of Southern California, Los Angeles, 1987.

[NS81] D. Nassimi and S. Sahni. Data broadcasting in simd computers. *IEEE Transactions on Computers*, C-30, No. 2:101–107, Feb. 1981.

[Raj90] S. Rajasekaran. Randomized parallel selection. In *Foundations of Software Technology and Theory of Computation*, pages 176–184, 1990.

[TCS89] X. Thibault, D. Comte, and P. Siron. A reconfigurable optical interconnection network for highly parallel architecture. In *Proceedings of 2nd Symposium on the Frontiers of Massively Parallel Computation*, 1989.

School of Computer Science, Carleton University
Recent Technical Reports

TR-178 **Determining Stochastic Dependence for Normally Distributed Vectors Using the Chi-squared Metric**
R.S. Valiveti and B.J. Oommen, July 1990

TR-179 **Parallel Algorithms for Determining K-width-Connectivity in Binary Images**
Frank Dehne and Susanne E. Hambrusch, September 1990

TR-180 **A Workbench for Computational Geometry (WOCG)**
P. Epstein, A. Knight, J. May, T. Nguyen, and J.-R. Sack, September 1990

TR-181 **Adaptive Linear List Reorganization under a Generalized Query System**
R.S. Valiveti, B.J. Oommen and J.R. Zgierski, October 1990

TR-182 **Breaking Substitution Cyphers using Stochastic Automata**
B.J. Oommen and J.R. Zgierski, October 1990

TR-183 **A New Algorithm for Testing the Regularity of a Permutation Group**
V. Acciaro and M.D. Atkinson, November 1990

TR-184 **Generating Binary Trees at Random**
M.D. Atkinson and J.-R. Sack, December 1990

TR-185 **Uniform Generation of Combinatorial Objects in Parallel**
M.D. Atkinson and J.-R. Sack, January 1991

TR-186 **Reduced Constants for Simple Cycle Graph Separation**
Hristo N. Djidjev and Shankar M. Venkatesan, February 1991

TR-187 **Multisearch Techniques for Implementing Data Structures on a Mesh-Connected Computer**
Mikhail J. Atallah, Frank Dehne, Russ Miller, Andrew Rau-Chaplin, and Jyh-Jong Tsay, February 1991

TR-188 **Generating and Sorting Jordan Sequences**
Alan Knight and Jörg-Rüdiger Sack, March 1991

TR-189 **Probabilistic Estimation of Damage from Fire Spread**
Charles C. Colbourn, Louis D. Nel, T.B. Boffey and D.F. Yates, April 1991

TR-190 **Coordinators: A Mechanism for Monitoring and Controlling Interactions Between Groups of Objects**
Wilf R. LaLonde, Paul White, and Kevin McGuire, April 1991

TR-191 **Towards Decomposable, Reusable Smalltalk Windows**
Kevin McGuire, Paul White, and Wilf R. LaLonde, April 1991

TR-192 **PARASOL: A Simulator for Distributed and/or Parallel Systems**
John E. Neilson, May 1991

TR-193 **Realizing a Spatial Topological Data Model in a Relational Database Management System**
Ekow J. Otoo and M.M. Allam, August 1991

TR-194 **String Editing with Substitution, Insertion, Deletion, Squashing and Expansion Operations**
B John Oommen, September 1991

TR-195 **The Expressiveness of Silence: Optimal Algorithms for Synchronous Communication of Information**
Una-May O'Reilly and Nicola Santoro, October 1991

TR-196 **Lights, Walls and Bricks**
J. Czyzowicz, E. Rivera-Campo, N. Santoro, J. Urrutia and J. Zaks, October 1991

TR-197 **A Brief Survey of Art Gallery Problems in Integer Lattice Systems**
Evangelos Kranakis and Michel Pocchiola, November 1991

TR-198 **On Reconfigurability of Systolic Arrays**
Amiya Nayak, Nicola Santoro, and Richard Tan, November 1991

TR-199 **Constrained Tree Editing**
B. John Oommen and William Lee, December 1991

TR-200 **Industry and Academic Links in Local Economic Development: A Tale of Two Cities**
Helen Lawton Smith and Michael Atkinson, January 1992

TR-201 **Computational Geometry on Analog Neural Circuits**
Frank Dehne, Boris Flach, Jörg-Rüdiger Sack, Natana Valiveti, January 1992

TR-202 **Efficient Construction of Catastrophic Patterns for VLSI Reconfigurable Arrays**
Amiya Nayak, Linda Pagli, Nicola Santoro, February 1992

TR-203 **Numeric Similarity and Dissimilarity Measures Between Two Trees**
B. John Oommen and William Lee, February 1992

TR-204 **Recognition of Catastrophic Faults in Reconfigurable Arrays with Arbitrary Link Redundancy**
Amiya Nayak, Linda Pagli, Nicola Santoro, March 1992

TR-205 **The Permutational Power of a Priority Queue**
M.D. Atkinson and Murali Thiyagarajah, April 1992

TR-206 **Enumeration Problems Relating to Dirichlet's Theorem**
Evangelos Kranakis and Michel Pocchiola, April 1992

TR-207 **Distributed Computing on Anonymous Hypercubes with Faulty Components**
Evangelos Kranakis and Nicola Santoro, April 1992

TR-208 **Fast Learning Automaton-Based Image Examination and Retrieval**
B. John Oommen and Chris Fothergill, June 1992

TR-209 **On Generating Random Intervals and Hyperrectangles**
Luc Devroye, Peter Epstein and Jörg-Rüdiger Sack, July 1992

TR-210 **Sorting Permutations with Networks of Stacks**
M.D. Atkinson, August 1992

TR-211 **Generating Triangulations at Random**
Peter Epstein and Jörg-Rüdiger Sack, August 1992

TR-212 **Algorithms for Asymptotically Optimal Contained Rectangles and Triangles**
Evangelos Kranakis and Emran Rafique, September 1992

TR-213 **Parallel Algorithms for Rectilinear Link Distance Problems**
Andrzej Lingas, Anil Maheshwari and Jörg-Rüdiger Sack, September 1992

TR-214 **Camera Placement in Integer Lattices**
Evangelos Kranakis and Michel Pocchiola, October 1992

TR-215 **Labeled Versus Unlabeled Distributed Cayley Networks**
Evangelos Kranakis and Danny Krizanc, November 1992

TR-216 **Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers**
Frank Dehne, Andreas Fabri and Andrew Rau-Chaplin, November 1992

TR-217 **Indexing on Spherical Surfaces Using Semi-Quadcodes**
Ekow J. Otoo and Hongwen Zhu, December 1992

TR-218 **A Time-Randomness Tradeoff for Selection in Parallel**
Danny Krizanc, February 1993

TR-219 **Three Algorithms for Selection on the Reconfigurable Mesh**
Dipak Pravin Doctor and Danny Krizanc, February 1993