The Smart Access Memory: An Intelligent RAM for Nearest Neighbor Database Searching

Aaron Lipman and Woodward Yang

Abstract

The nearest neighbor algorithm is well suited to an IRAM implementation, given its high bandwidth requirement between memory and processing. We have designed and implemented the Smart Access Memory (SAM) chip to retrieve the k nearest neighbors to a query point in a database of example data vectors by integrating 64 bit-serial processors, 4096 sorting units, and 16Mb of memory using a 0.35μm commercial DRAM process (the 1.6mm $\times$ 1.6mm chip is currently in fabrication.) The 800K transistor processor array can be clocked at speeds between 100-200MHz providing an internal memory bandwidth of 6.4-12.8Gb/s. This approach also allows for multiple chips to be used in parallel, and a system of 16 SAM chips can accelerate a brute-force nearest neighbor search by over three orders of magnitude compared to a highly optimized software implementation on a 100MHz PA-RISC workstation.

1 Introduction

Nearest neighbor database searching is used intensively for tasks such as pattern recognition and content-based querying. A set of data vectors are stored in an example database, and a query is presented. The vectors in the example database are compared to the query using a programmable distance metric. The data vectors with the smallest distance to the query point are called nearest neighbor matches. This is an important step in several learning algorithms which use this information to tune locally weighted units, or perform other types of local learning or compression schemes. Furthermore, this can be used directly to perform object recognition tasks, and other memory based computing tasks such as robot kinematics and control. This can be a memory intensive task as often exhaustive searches of the database must be performed to retrieve the nearest neighbor matches, particularly for high-dimensional data [1] [3]. Low dimensional data can be organized into tree-like structures for efficient retrieval on the order $O(\log N)$ where N is the number of vectors in the database. However, the advantage of these structures is dependent on the dimensionality of the data, such that the search time grows as 2^d where d is the dimensionality of the data. Figure 1 compares the performance of a KD tree algorithm and the brute force search on the retrieval of nearest neighbors from a database of 100,000 examples. As the dimensionality of the database is increased from 2 to 16 dimensions, the KD tree loses its performance advantage until its performance is on par with, or worse than the brute force search. This is due to the nature of the tree structure. Multi-dimensional trees seperate the data into discrete regions. When a query point is presented, only those few regions neighboring the query are searched for close matches. The other regions can be eliminated given their distance from the query. As the dimension increases, the regions neighboring a query point grow as

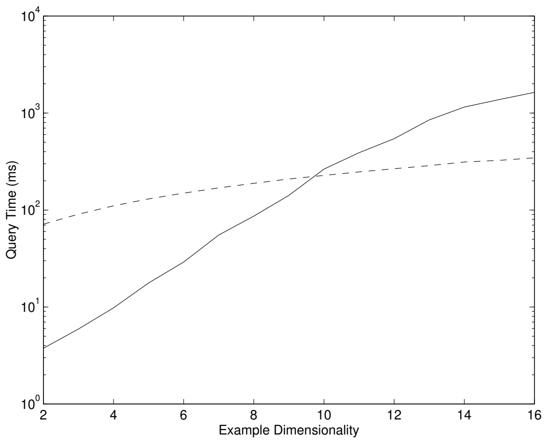

Figure 1: *Comparison of KD Tree (solid line) and Exhaustive Search (dashed line) query times for the retrieval of the 100-nearest neighbors to a query point in a database of 100,000 examples as the dimensionality varies from 2 to 16.*

Brute force nearest neighbor searching is a very I/O intensive task The data vectors stored in memory, must be loaded one after the other into a processing unit for distance calculation, and the resulting distances must be sorted in a priority queue of size k that orders the k nearest neighbors among all the distance measurements. One advantage of this type of search is that it is easily performed in parallel. The data vectors can be divided up among multiple memory banks and searched in parallel.

Thus, the nearest neighbor database search task is well suited to an intelligent RAM (IRAM) approach. A high density, commercial 16Mb DRAM was modified for this task by incorporating bit-serial processing elements for computing distances and a sorting queue to keep track of close matches. The Smart Access Memory (SAM) chip is loaded with the entire example database and can operate as a normal DRAM. However, the SAM chip also runs in a mode where a query is presented to the DRAM, and a list of pointers to the k nearest neighbors and their respective distances are returned. Thus, a simple low bandwidth query is sent to the SAM, and a low bandwidth list is returned, while inside the chip, a high-bandwidth parallel computation and exhaustive search are performed.

2 SAM Chip Overview

2.1 Basic Architecture

There are three basic components of the SAM chip - (i) memory, (ii) distance processors, and (iii) sorting units. This architecture is illustrated in Figure 2

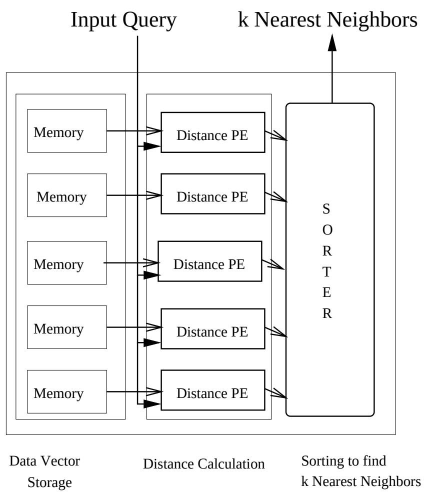

Figure 2: *Block diagram of Smart Access Memory (SAM). Data vectors are stored in memory. An input query is presented, and distance processing elements (PEs) compare the query against the data vectors stored in memory. The distances are fed into a sorting network that returns the k nearest neighbors.*

Data vectors stored in the memory are accessed in parallel by a number of processors which measure the distance between the stored examples and a common query using a programmable metric. The metric chosen for the processors is a weighted L1-norm as shown in equation 1, with fully programmable weights α_i. 1.

$$Distance = \sum_{i=1}^{n} \alpha_i |x_i - y_i| \tag{1}$$

The processors feed their results into a sorting queue which maintains the best matches (i.e. the smallest distances) to the query, as well as pointers indicating the data vector corresponding to the distance measurement. The SAM chip as designed contains 64 blocks of 256Kb DRAM for a total of 16Mb. Each block of DRAM is connected to a single processor, and sorting queue slot which is capable of retaining a single match. Thus, the SAM chip records the 64 nearest neighbors to an input query.

The design of the processors and sorters is bit-serial. This design grew out of a general goal to implement a high-speed I/O system between memory and processing. Though the area required for layout of 16 bit-serial distance calculation processors is the same as a single 16b processor, the speed of bit-serial processors can be 4 to 5 times that of bit-parallel processors due to their simple logic implementation, avoiding long carry chains and other complex design issues. Thus, a bit serial system would have a higher I/O interface due to improved clock rates, though the inherent level of computational parallelism is the same in both cases. Furthermore, the pitch of the small bit-serial processors, and sorters matches the pitch of the DRAM blocks, providing a simple integration of the parts on a single die. Both the processors and sorters are fully pipelined, operating on a new bit of data each clock cycle, and can be driven at 100-200 Mhz.

2.2 The L1-Norm Processor

A block diagram of the L1-norm processor is shown in figure 3. The processor calculates a weighted L1-norm between a query vector and a data vector stored in DRAM. The data vectors are coded as 16b integers and the weights are coded as 8b integers. The L1-norm processor reads in one bit from a query vector and one bit from a data vector stored in a DRAM block LSB-first on each clock cycle. The vectors are fed through a comparator which determines for each component of the vector the large of the two inputs. After 16 cycles, the absolute difference between the two vector components is performed bit-serially by subtracting the smaller number from the larger using the result of the comparison. The absolute difference is fed into a 2b digit-serial multiplying accumulator which multiplies the absolute difference by a programable 8b weight, and adds it to the previously accumulated result. The processor accumulates the distance measurement to 32b using saturating adders, which provides for calculations on data vectors up to 256 dimensions without overflow. If data vectors have more than

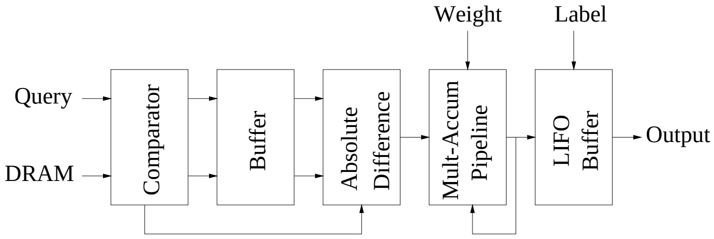

Figure 3: *Block diagram of the bit serial L1-norm processor.*

256 dimensions, overflow can be avoided by reducing the number of bits used for the weights or data representation. However, in many cases, distance measurements which produce overflow can probably be safely ignored on the assumption that they are too far away from the query to be an interesting close match. A 32b tag that identifies the data vector used in the distance caluclation is appended to the 32b distance result. This 64b result is buffered and output from the L1-Norm processor in bit-serial fashion. The 32b distance result is output MSB first followed by the tag. This orientation is required for proper interfacing with the sorting queue.

2.3 The Sorting Queue

The sorting queue is based on a single bit router, which after being reset, routes the larger of two inputs on one output wire, and the smaller of the two on the other (see figure 4). 64 of these routers can store a single 64b result.

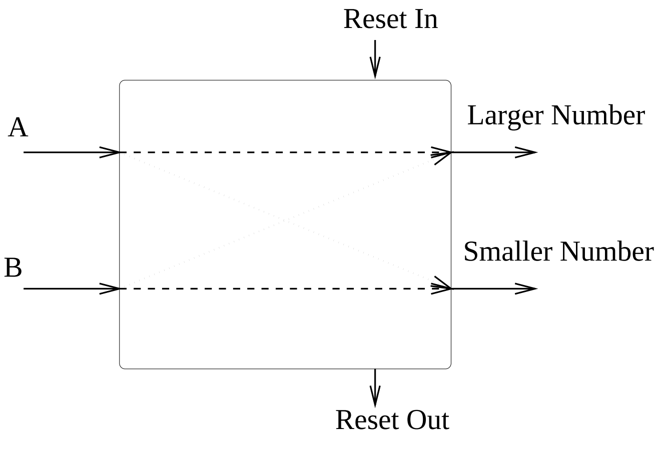

Figure 4: *Bit serial router architecture: Two distances are fed MSB first into the 'A' and 'B' input lines. After a reset signal is asserted, the inputs are compared and routed to the proper outputs until a new reset signal triggers another comparison. The reset signal can be passed among neighboring routers to synchronize a grid of routers.*

The SAM chip contains 4096 of these routers organized in a 64 $\times$ 64 grid to maintain the top-64 results [2]. The sorters operate in a systolic fashion, so a delay skew must be introduced into the architecture. A sample sorting queue is shown in figure 5. This 3 $\times$ 3 grid holds the

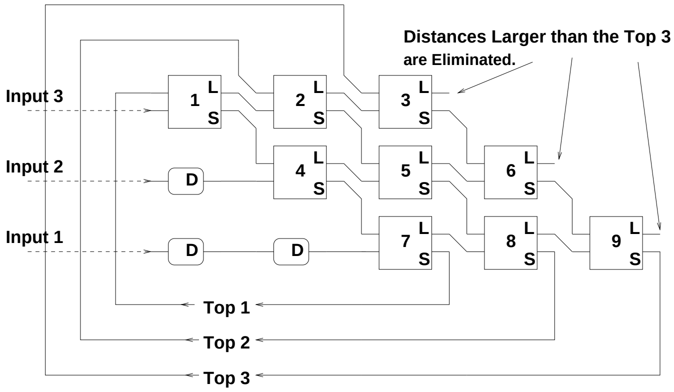

Figure 5: *Block diagram of the bit serial systolic array sorter.*

three smalles 3b numbers it sees. Three new inputs are sent to the grid with varying delay stages. The three values currently stored in the grid are found in increasing magnitudein positions 1,4,7, 2,5,8, and 3,6,9 repectively. As these old results rotate through their positions, they are compared MSB first to the new inputs. If the inputs are smaller than the numbers in the queue, they will replace the old results which drop off the end of the queue. If they are larger, they will pass through the queue and drop off the end. The 64 $\times$ 64 grid of routers used in the sorting queue on the SAM chip can sort 64 new inputs in 64 cycles, giving the same performance as sixty-four 64b comparators, but operates at high speed (100-200Mhz). The only cost is the addition of delay units to introduce a data skew into the sorting queue. Since both the L1-norm processors and sorters are fully pipelined, the data skew is introduced by placing delay units between the DRAM blocks and L1-norm processors for ease of control. This is explained more fully in a later section.

2.4 DRAM Integration

The DRAM blocks are layed out using a 0.35μm DRAM process with a 1.5μm^2 stacked DRAM cell. The DRAM cells are organized into 64 separate 256Kb blocks with an overall 32b external I/O access. Each 256Kb DRAM block is interfaced to a bit-serial L1-Norm distance processing element which connects to a bit-serial sorting column array made of a 64 $\times$ 1 routing elements. These 64 memory-processor combinations operate in parallel to produce the best 64 matches to an input query and are integrated on a single 16mm $\times$ 16mm die. The pitch of

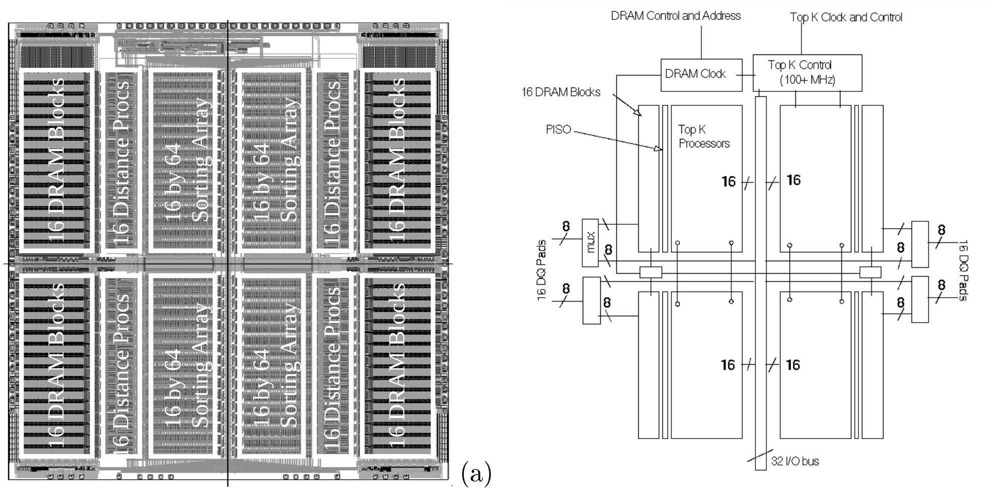

Figure 6: *Figure 6a shows a plot of the SAM chip, identifying the DRAM blocks, distance processors, and sorters. Figure 6b shows a block diagram of the chip, identifying the control and I/O paths.*

the processors and the sorting columns matches the pitch of the DRAM blocks, though the design rules used for the processing elements were quite different. While the high density DRAM regions were designed using 0.35μm technology, the process elements and routers were designed using 1.0μm equivalent technology due to to the standard practice of using highly relaxed design rules in the peripheral (non-DRAM cell) regions. The resulting processing elements and routers were approximately twice the area of the DRAM blocks. However, simple improvements in the design rules used in the peripheral regions, mostly concerning the metal contact and pitch could easily reduce the area required by the processing elements and routers by a factor of 4. Additional metal layers (more than two metal layers) would also significantly improve the suitability of DRAM process technology for implementation of high density logic circuitry and could also be easily incorporated in a DRAM process especially in the peripheral regions where surface topology problems are much less severe than in the stacked DRAM cell regions.

A plot and a block diagram of the SAM chip are shown in Figure 6. The memory is divided into 4 banks of 16 blocks. The processors and sorters are located adjacent to the DRAM blocks. A 32b I/O bus runs the length of the chip and is used to read out the results from the sorting network, as well as to provide input and control signals to the control unit shown at the top of the chip. The processor array I/O bus is connected to the standard DRAM 32b I/O bus and multiplexed to the 32 DQ pads as shown on the sides of the chip. Data from the DRAM blocks is interfaced to the processors via a parallel-in-serial-out (PISO) buffer explained in detail in a later section. There are two separate controllers shown at the top, one to clock the DRAM, and the other to control the processors and sorting for the top k nearest neighbors.

3 DRAM Block Interface

Each 256Kb DRAM block is organized as 256 rows and 1024 columns. Thus, a 1024b output is latched by the sense amps of each DRAM block on every row access (RAS). Given the extremely small 1.3μm pitch of DRAM cells, it is clearly not practical to directly interface all 1024b to external logic. Furthermore, by using bit-serial processing elements, the maximum clock speed of the PEs was several times faster than the minimum RAS cycle time. Thus, the DRAM and PEs were interfaced by using a column selector to multiplex the 1024b static column into an 8b output which fed a parallel-in-serial-out (PISO) unit that subsequently shifted the data bit serially into the processor at high speed. This arrangement is shown in Figure 7. The CAS clocking for the static column reads can be performed at cycles between 40-80ns, while the bit-serial processors are clocked eight times more quickly. While the $\times$8b output from the column selector of the DRAM blocks permitted the simplest interface to the processors, it also results in a less efficient column redundancy scheme for the 256Kb DRAM block since entire blocks of 8 columns must be replaced even if only there is only one bad column.

The nearest-neighbor operation requires in-order streaming through all the data in the DRAM blocks. Reading from all the DRAM blocks in parallel requires a RAS to be performed on all the blocks concurrently. However, due to the very large peak currents required to precharge and read all of the DRAM banks in parallel, the RAS timing must be extended to 110ns, and the

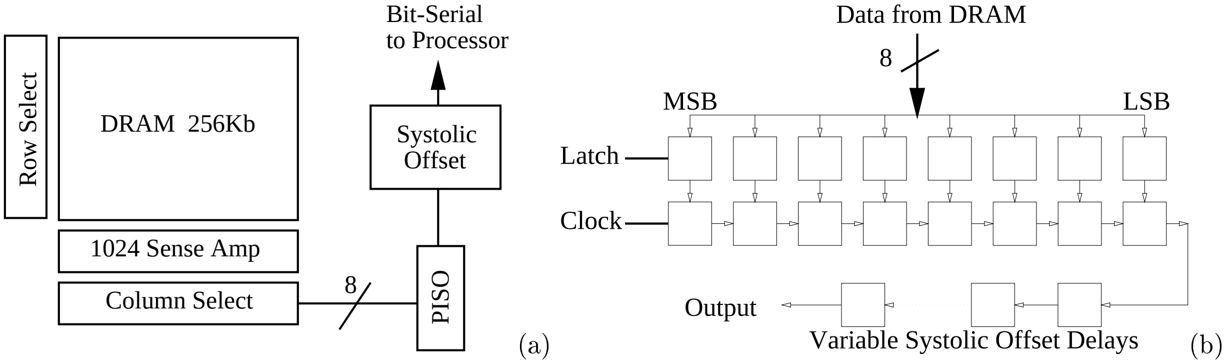

Figure 7: *Figure 7a shows the organization of the DRAM block. The column select unit brings 8b into the PISO shown in Figure 7b. The PISO is a double buffered latch that shifts the data out serially into delay units used to create a systolic skew for the processors and sorters. The first latch is triggered by a load signal that occurs after each CAS, and the second is synchronized to the system clock which shifts the data into the processors.*

logic must be also be able to accommodate stalling during the slow parallel RAS at the end of reading each row. Other alternatives, such as skewing the memory accesses so that only a few blocks must perform a RAS at a single time were not implemented due to greater timing complexity and requirement for additional registers within each DRAM block to buffer row and column addresses.

The memory control is accomplished by a small FSM which produces the row and column addresses for the DRAM blocks. The addresses are multiplexed with the pre-existing address lines. In normal DRAM mode, the addresses are applied externally, but in SAM mode, the addresses are supplied by the FSM. DRAM refreshing can be accomplished externally. During SAM mode, refresh is performed automatically by reading through all the data in series. In fact, if the processors are clocked at 100MHz, the maximum time required to read all the data in the DRAM would be 2.56ms, which is a more than adequate refresh rate.

4 Logic Integration and I/O

The memory, processors, and sorters are connected as shown in figure 8. The first processor in the array passes values to the subsequent processors, whereas the sorting columns are connected in a ring to recycle the currently stored values for subsequent comparisons. The data skew introduced into the processing and sorting array as mentioned in a previous section serves two purposes. The first is to align the data for proper operation with the systolic sorting array. The second is to prevent the necessity of global control signals by introducing a skew into the operation of the processors. Instead of presenting the query bits to each processor at the same time, they are passed from processor to processor in order, as are the majority of the control signals. This local control scheme presents an advantageous implementation on a large die as it is difficult to generate and route global control signals. The reason for this systollic arrangement is analogous to the bit-serial vs. bit-parallel decision that was made with regard to the logic. Control signals can be sent from processor to processor at high speeds on each cycle, obviating the need for the generation of global control and data signals. The only added requirement is the introduction of buffers at each processor to receive and send control and data signals from processor to processor. Only two signals require global distribution - a reset signal, and a load signal sent to all the PISO units which stores the 8 bits produced by the column select circuitry in the PISO buffer. Thus, the DRAM blocks operate with parallel timing signals, but the delay stages introduced between the PISO and the processors allow for systollic control of the logic.

Data in the form of query vector bits and weights, as well as control signals that synchronize the operation of the chip are received via the 32b I/O bus shown in figure 6b. 8b PISO units take a byte each of query and weight data, and feed it bit serially to the first processor in the system. Thus, query data from a host system can take place at a clock rate one eighth of the internal logic. The operation of the chip proceeds as follows. After stalling the logic during a RAS cycle, the 1024 bits of data on the sense amps of each DRAM block are fed 8 bits at a time in the manner previously described to the interface PISO. The first processor in the array recieves data and control bits at the same time it recieves its first bit of data from the PISO. These data and control signals are passed to the next processor on the next cycle, just as it recieves its first bit from the PISO, and a single stage of delay. Additional stages of delay align the control and data signals for each processor as they receive their data from their respective delay chains. The sorting elements are normally stalled, but when new results are produced by the processors, the sorting queue is activated for 64 cycles to process the new results.

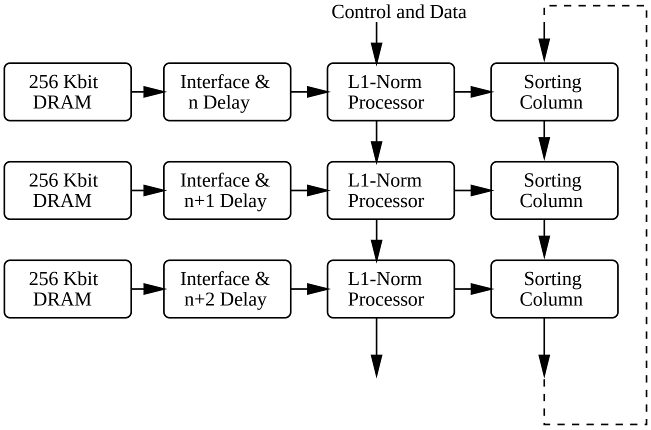

Figure 8: *The data and control flow is pictured above. Each processor passes the control signals to the next processor. The sorters are connected in a ring such that the results in the sorting grid will be continually compared against new results from the processors.*

The processors and sorters are laid out in two columns between two columns of 32 DRAM blocks, with the control and data signals wrapped around on the bottom and top as appropriate.

After all the stored data vectors have been read, the 64 closest matches are read out of the sorting que over the 32b I/O bus. As the results are clocked out of the sorting array, the 32b distance of the best match is read first, and then the 32b address tag. The next best match is read in a similar fashion, and the remaining matches can be clocked out in order.

5 Clocking

There are only three signals that must be applied globally across the chip. Two are the reset and load signals mentioned above, and the third is a global clock signal. Neither the reset or load signals are on a critical path, but the clocking still presents a challenge, as it is necessary to synchronize all of the processors across a large 16mm × 16mm die. A full 4-phase clock generator is required for the logic, which presents a difficulty in generating and synchronizing the four clock signals $\phi_1, \_\phi_1, \phi_2$, and $\_\phi_2$. Instead, we exploit the fact that each processor and sorting column is only connected to two neighboring processors, one of which is read from, the other of which is written to. Thus only a single global clock signal ϕ is generated, and each processor and each sorting column has its own phase generator that produces the four clock phases required for the logic. The only requirement for synchronizing the clock phases is the use of a logical AND on the clock phases of neighboring processors and sorters. Thus, data is transfered only when ϕ_2 is high for both processors. Thus, only simple glue logic is required to enforce local synchronization between neighboring processors and sorters. This configuration is shown in Figure 9

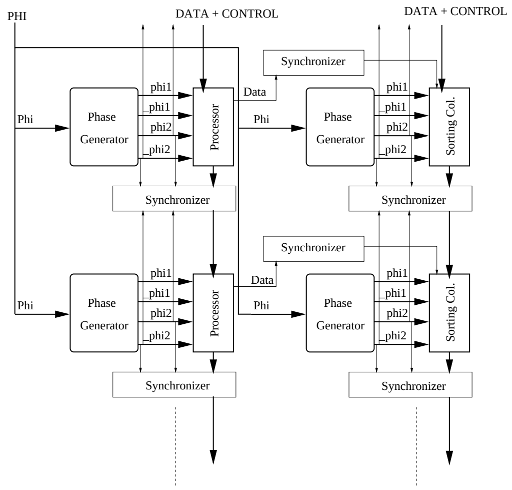

Figure 9: *A single clock signal PHI is split into four phases by independent phase generators. Simple synchronization logic is used to pass values between neighboring processors and sorters, by aligning the phi2 phases.*

6 Combining Multiple Chips

Multiple SAM chips can be used in parallel, dividing up a large database among the chips. This can be done to accomodate large data bases that do not fit on a single chip, as well as to increase the number of processors used to search the data. The sorting queue on each chip will hold its own "local" 64 nearest neighbors, but the true 64 nearest neighbors overall must be determined by merging the results from each chip. This is a simple task, as each chip contains a sorted list, and algorithms for merging sorted lists are simple $O(N)$ operations. Since each chip only stores 64 nearest neighbors locally, the use of multiple chips can only guarantee the 64 nearest neighbors globally without massive communications between the chips during the sorting process. However, a probabilistic method can be used to retrieve more than the 64 nearest neighbors with no inter-chip communication until all the examples have been sorted.

Consider dividing a database of N examples among B chips. Each chip will store roughly N/B examples. Within each chip, the Q closest matches will be stored in a local priority queue. From the Q closest matches in each chip, it is possible to guarantee the correct recovery of the global K closest matches under the following two trivial conditions.

Proposition 1 *If $K \leq Q$, the global K nearest neighbors can always be correctly recovered. The local priority queues are large enough to ensure no overflow from any single priority queue.*

Proposition 2 *If $N/B \leq Q$, the global K nearest neighbors can always be correctly recovered, because the entire example database is small enough to reside within the B priority queues.*

In addition, the recovery of the global K nearest neighbors is impossible under the following trivial condition.

Proposition 3 *If $K > Q*B$, the global K nearest neighbors can never be recovered, because there is insufficient storage space in all the priority queues.*

However, the most interesting conditions arise for large example databases ($N/B \gg Q$) and for the recovery of a moderate number of close examples ($Q * B > K > Q$) from a database that has been randomly distributed among the B sorting chips.

Proposition 4 *If $N/B \gg Q$ and $Q * B > K > Q$ and the database of N examples is randomly distributed among the B sorting chips, then the probability $Prob(fail)$ that all of the K nearest neighbors cannot be recovered is given by*

$$Prob(fail) = \frac{B \sum_{i=1}^{K-Q} (B-1)^{(K-(Q+i))} \frac{K!}{(K-(Q+i))!(Q+i)!}}{B^K} \tag{2}$$

Clearly, the total distinct number of ways of distributing the global K nearest neighbors among B chips is simply B^K. Furthermore, each of these configurations is equally likely since the database is randomly distributed and $N/B \gg K$. However, among these B^K possible configurations, there will be some number of configurations that will have more than Q of the K nearest neighbors in a single chip which will prevent the recovery of all of the K closest matches. These configurations are characterized by having a single chip with $Q+1$ or more of the K nearest neighbors, with the remaining nearest neighbors distributed among the other sorting chips. The number of possible ways of having one chip with $Q+1$ or more of the K nearest neighbors is given by $B \sum_{i=1}^{K-Q} (B-1)^{(K-(Q+i))} \frac{K!}{(K-(Q+i))!(Q+i)!}$. The $(B-1)^{(K-(Q+i))}$ term represents the number of ways of configuring the remaining $K-(Q+i)$ nearest neighbors in the remaining $B-1$ sorting chips. The $\frac{K!}{(K-(Q+i))!(Q+i)!}$ term represents the number of ways of selecting $Q+i$ matches (to be located in a single sorting chip) from among the K nearest neighbors. The coefficient B represents the permutations over each of the B sorting chips. For a system composed of 16 (B=16) chips, each of which has a local priority queue of depth 64 (Q=64), the probability $Prob(fail)$ of incorrectly retrieving the 256 (K=256) nearest neighbors among all the chips is roughly 2.5×10^{-21}. Furthermore, the distances retrieved from the chips are guaranteed to be global close matches as long as less than Q matches have been retrieved from any single sorting chip. Thus, the number of close matches that can be retrieved with high confidence can be much larger than Q. The number of close matches that can be retrieved with high probability increases as more chips are added to the system, so that the system memory, processing power, and priority queue size are all scalable with the number of sorting chips and example database size.

7 Projected Performance

As an example, assume best matches are to be found in a database of 16K 64-dimensional examples. One 64-dimensional example requires 1Kb of storage (16b per dimension $\times$ 64 dimensions). 16K examples require 16Mb of storage, which fits on a single SAM chip. A 100MHz PA-RISC workstation takes about 209ms to retrieve the 64 closest matches to an input query. A single chip with 64 processors requires 2.56ms to retrieve the 64 nearest neighbors - the L1-norm processors load one bit per clock cycle, and will take 256K clock cycles, (the size of the DRAM block in bits) to process all the examples. At 100MHz, this takes 2.56ms, if we ignore the small pipeline latency. A system with 16 chips working together would require only 160μs. Thus, speedup factors of greater than 10^3 are feasible using the special-purpose hardware. The large I/O bandwidth (6.4Gb/s) between memory and processing on the SAM chip provides a considerable advantage over a standard serial computer approach.

References

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching," *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pp. 573-582, 1994. Also available in revised form at "http://www.cs.umd.edu/

[2] F. T. Leighton, "Introduction to Parallel Algorithms and Architectures" Morgan Kaufmann Publishers, Inc., San Mateo, 1992.

[3] R. F. Sproull, “Refinements to Nearest-Neighbor Searching in k-Dimensional Trees” *Algorithmica* vol. 6, pp. 579-589, 1991.