

Energy-Performance Trade-offs for Spatial Access Methods on Memory-Resident Data *

Ning An Sudhanva Gurumurthi Anand Sivasubramaniam N. Vijaykrishnan
Mahmut Kandemir Mary Jane Irwin

Dept. of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802.
Email: {an,gurumurt,anand,vijay,kandemir,mji}@cse.psu.edu

Abstract

The proliferation of mobile and pervasive computing devices has brought energy constraints into the limelight, together with performance considerations. Energy-conscious design is important at all levels of the system architecture, and the software has a key role to play in conserving the battery energy on these devices. With the increasing popularity of spatial database applications, and their anticipated deployment on mobile devices (such as road atlases and GPS based applications), it is critical to examine the energy implications of spatial data storage and access methods for memory resident datasets. While there has been extensive prior research on spatial access methods on resource-rich environments, this is, perhaps, the first study to examine their suitability for resource-constrained environments. Using a detailed cycle-accurate energy estimation framework and four different datasets, this paper examines the pros and cons of three previously proposed spatial indexing alternatives from both the energy and performance angles. Specifically, the Quad-tree, Packed R-tree, and Buddy tree structures are evaluated and compared with a brute-force approach that does not use an index.

The results show that there are both performance and energy trade-offs between the indexing schemes for the different queries. The nature of the query and the actual dataset also play an important role in determining the energy-performance trade-offs. Further, technological trends and architectural enhancements are influencing factors on the relative behavior of the index structures. The work in the query has a bearing on how and where (on a mobile client or/and on a server) it should be performed for performance and energy savings. The results from this study will be beneficial for the design and implementation of embedded spatial databases, accelerating their deployment on numerous mobile devices.

Keywords: Spatial Data, Multidimensional Indexing, Energy Optimization, Resource-constrained Computing.

1 Introduction

Computing is becoming a pervasive and ubiquitous part of everyday life. The traditional modus-operandi of sitting at a desk to interact with a computer system is gradually going out of style, with users demanding access to computational resources and information whenever and wherever (even when they are on the move) they choose. These needs have opened the door to several interesting and crucial topics for research in the broad domain of mobile and resource-constrained computing. Focusing specifically on spatial databases (an important and useful class of mobile applications), this paper explores the energy (a scarce and valuable resource in mobile devices) consumption and performance trade-offs of different storage organizations for spatial data on resource-constrained mobile devices.

Programs running on mobile devices (PDAs, laptops, etc.) can be subject to very different operating conditions compared to their desktop/server counterparts. This includes limited computational resources, storage capacity, battery energy, and connectivity, that are a consequence of design considerations such as small form factor, weight, cost and diverse operating conditions. It is widely recognized that battery energy is, perhaps, one of the the most challenging limitations, with

*This paper is a significantly extended version of preliminary work that appeared in the Proceedings of the Very Large Databases (VLDB) 2001 Conference. The extensions include (i) a comparison of indexing alternatives with doing the operations in a brute-force manner, (ii) observations showing that datasets do play a role on power consumption, (iii) architectural solutions to address the cache and memory hotspots for energy, and (iv) benefits when off-loading the work to a server over a wireless medium compared to doing everything on the handheld device.

many other factors (such as computational speed) directly or indirectly related to energy availability. Mobility precludes the use of a wall socket to power the device, and at the same time one does not wish to carry a heavy battery along for its operation. The growing mismatch between energy capacity of batteries and the energy consumption of mobile devices makes it all that much more critical to employ algorithmic, software and architectural techniques for energy savings. It is hypothesized [IKVS00] that high level optimizations in algorithms and data structures can give much more energy savings than micro-managing the energy consuming resources at a very low level. Such optimizations can even amplify the savings obtained from well-known low level energy saving techniques [VKI⁺00], and are thus the motivation for this work.

Database applications are expected to be one of the dominant workloads running on the mobile devices [AK93]. Apart from the prevalent personal organizer database applications (address book, calendar, etc.), there are numerous productivity-enhancing commercial, entertainment and convenience-based database applications envisioned for such devices. This paper specifically focuses on spatial databases, an important class of applications for the mobile devices. In general, Spatial Database Management Systems (SDBMS) [SCR⁺99] have found widespread adoption in numerous areas including Geographical Information Systems (GIS), Image Processing, Military Planning and Logistics, Computer Aided Design (CAD), Multimedia Systems, and Medical Database Systems. SDBMS are important for mobile computing, with several possible applications in this domain. Already, mobile applications for spatial navigation and querying using a street atlas are available for many PDAs [Mic, GEO]. In addition, traditional data input and querying for conventional SDBMS can be supplanted by mobile operations for better productivity and convenience.

Even in a resource-rich environment, SDBMS design and implementation is a difficult problem [SCR⁺99], because the system has to deal with multidimensional data as opposed to conventional databases, where single dimensional indices may suffice. Data objects have varying sizes associated with them, and spatial operations are in general much more complex than standard relational operators. Further, the efficiency of the storage organization is highly dependent on the nature and idiosyncrasies of the spatial dataset. Moving the target to a mobile device makes the design and implementation of a SDBMS even more challenging. Resource constraints such as limited energy, computational power and memory add to the complexity of the problem. Performance is not necessarily the only goal for optimization. Sometimes a user may be willing to sacrifice some amount of performance if that will enable a device to run longer on battery. Further, power dissipation of different system components may also be an important issue for thermal considerations.

There are several important and interesting issues in designing a SDBMS for a resource-constrained mobile device, and a few of them include *connectivity* (communication), *data storage and access methods*, *query processing*, and *dynamic adaptation* which are detailed below.

With limited resources, there is the important question of where should the operations (queries) be performed. Does it make sense to ship an operation to a resource-rich server (which may, perhaps, have access to the data) and simply ask a mobile device to act as an intermediary to display the end results to the inquiring user, or should the device itself perform the operation? There are several practical considerations that may force the latter choice. The first reason is connectivity. It may not always be possible to be able to connect to a resource-rich machine (e.g. the connection could be bad, a user may be in an unreachable location, or a user may not have subscribed to a service for communication). Communication is also energy consuming in the wireless components, and it is essential to trade-off the computation energy saved with the additional communication cost [IVB94]. Second, even if the user is able to connect, there is the issue of privacy (to avoid giving out query parameters or user location). Finally, several (not all) spatial databases are static with information rarely updated. For example, once a road atlas is download to a mobile device, the user may not want any more updates till the device is resynchronized explicitly later on. All these reasons may warrant the storage of the spatial data on the mobile device itself, and with the queries directly performed on it. Work partitioning between a mobile device and a server is also briefly explored in this paper. A more detailed study of this topic is given in [GAS⁺01], and for most of the initial discussions in this paper we consider the dataset to be fully resident on the mobile device.

If the dataset needs to be stored in a mobile device, how should it be organized for good performance? Earlier work has

focussed mainly on optimizing the retrieval and processing of large disk-resident spatial datasets on server environments. It is imperative to revisit this issue for resource-constrained devices with limited memory and without the presence of a disk (while laptops are equipped with small disks, few other mobile devices enjoy this luxury) not only from the performance viewpoint, but from the energy consumption angle as well.

Query processing and optimization is always a key determinant to performance [IP95, PMD98]. Decomposing the high level user request into the fundamental database operations, and deriving a query execution path should be based on both performance and energy consumption. Dynamic adaptation based on changing resource constraints (such as energy, connectivity, etc.) is another important consideration. Modulation of the storage structures, query execution and optimizations is needed when the operating conditions are changing.

These are just a few of the important issues for consideration when designing a SDBMS for a mobile device. Examining all these issues is overly ambitious, and is well beyond the scope of this paper. Instead, we specifically focus on the second problem listed above: *what are the performance and energy implications of storing and processing memory-resident spatial data on a resource-constrained device?* In particular, we address the issue of storing spatial data in main memory and performing certain basic spatial operations on this data including point queries, range queries and nearest-neighbor queries. We assume that all of the dataset is resident in the memory of a mobile device, there is no necessity for communication with a server (no dynamic updates), and complex queries (and their optimizations), such as spatial joins, are not considered. This is a largely unexplored area, with most previous work on spatial databases examining storage organizations on disks of resource-rich environments. Memory resident spatial data organization has not been extensively studied from the performance angle, let alone the energy viewpoint.

The first step to the development of energy and performance efficient storage organizations for memory-resident spatial data is a rigorous examination of the pros and cons of the already existing solutions [GG98] that have been proposed for resource-rich environments. Such a study can not only identify energy-performance trade-offs between the existing solutions, but can suggest enhancements, or can even suggest entirely new storage organizations (though this issue is not extensively explored in this paper). At the same time, performance and energy profiles can suggest architecture/hardware enhancements to improve the performance and energy savings of resource-constrained systems. This is similar to the motivation behind a recent study [ADHW99] that has examined the execution profile of commercial relational DBMSs, except that our focus here is on SDBMS and energy profiling (together with performance-energy trade-offs) that has not been explored before. This paper takes the first step to the development of energy-efficient SDBMS by attempting to answer the following important questions:

- How do the previously proposed alternatives for spatial data organization such as Quad-trees [FB74, NS87, HS91, HS92], R-trees [Gut84, KF93] and Buddy trees [SK90, See91], compare for memory resident datasets in terms of performance? What are the energy consumptions of these different structures when answering queries?
- During the processing of a query, how much energy and time are expended in traversing the index structures to identify candidates that are potential solutions for the query (filtering step)? Subsequently, how much energy and time are expended in performing the geometric operations on the actual candidate data items to find the exact solutions (refinement step)? Such software profiles are very useful to find hotspots for potential optimization (code restructuring), and to study the pros and cons of the structures in detail.
- For each phase of query processing, how much energy is consumed by the different hardware components of a mobile device - processor core, processor clock, cache, memory and buses? Such a hardware profile can also help us structure the code and suggest architectural enhancements to fix hardware hotspots, potentially without extending the execution time.
- How does the nature of the queries affect the performance and energy profiles? Spatial proximity can translate to improved locality in the data access patterns of the processor, thus reducing the cache and memory energy consumption.

At the same time, queries resulting in the selection of several data items can cause capacity and conflict misses in the cache, thereby increasing the energy consumption of the memory hierarchy.

- Traditionally node sizes of the hierarchical index structure are governed by performance related issues such as disk access costs, tree spans, etc. With memory-resident structures, how important a role does node size play in performance for spatial data? Are there any additional insights that an energy perspective can give to the choice of a good node size?
- What is the impact of technological trends on the relative performance of the schemes? In particular, with the observed hardware and software profiles, what energy and performance enhancing architectural features would be beneficial?
- Is there anything to gain in terms of energy and/or performance by offloading the query to a resource-rich server, where energy is not a concern, or should it be done only on the client because of the communication overheads?

To explore these issues, this study uses a detailed energy and performance estimation execution-driven simulator, called SimplePower [VKI⁺00], that is available in the public domain. A wireless network interface model has been added to this simulator. Four different storage organizations have been implemented on this simulator, and they have been used to evaluate three kinds of spatial queries on four different datasets. Detailed hardware and software profiles are used to answer the questions listed above.

To our knowledge, this is the first paper to present the energy and performance profiles/trade-offs for storage organizations for main memory spatial datasets. By presenting the first set of results on this topic, this study not only fills an important void in this area, but also sets the tone for future research on energy optimized storage organizations. The rest of this paper is organized as follows. The next section puts this study in perspective with the current state-of-the-art. Section 3 gives a quick overview of previously proposed index structures that are used in this evaluation. Section 4 explains the experimental setup and workloads. The results are presented in Section 5 and their implications are given in Section 6. Section 7 summarizes the contributions of this work.

2 Related Work

A great deal of prior work has been done in the area of storage organizations for spatial (multidimensional) data [Sam89, GG98]. This has led to the development of numerous index structures such as Grid Files [HN83], Quad-trees [FB74], $k-d$ -trees [Ben75], $k-d$ -trees [Rob81], Cell-trees [Gun86], BANG files [Fre87], h B-trees [LS98], Buddy-Trees [SK90], R-Trees [Gut84, TS96] and its variations $R+-$ Tree [SRF87] and R^* -Tree [BKSS90]. The book [Sam89] and survey articles [GG98] are testament to the indepth research that has been done in this area. In fact, there is a figure in [GG98] which shows the pictorial evolution of spatial data structures in which over 56 multidimensional access methods are depicted. All these structures have been proposed and evaluated from the performance, scalability, space overheads, simplicity, and concurrency management viewpoints. There has been no prior study examining their energy consumption behavior. Further, many of these are meant to be secondary storage index structures, and there has not been a detailed comparison of their suitability for memory resident datasets.

On the normal relational databases front, there have been investigations on memory resident datasets [LC98]. This has included new index structures and alternate algorithms for selection/projection/join to accommodate main memory processing. There has also been a recent study [ADHW99] that has profiled the execution time of a relational DBMS to understand the processor and memory implications on memory resident datasets. Again, none of these studies have explored the issues from the energy angle.

To our knowledge, most prior work on energy efficient indexing is for broadcast data [IVB94]. The problem there is to be able to power down wireless network interfaces intelligently so that they do not need to unnecessarily consume energy

listening in on conversations between a base station and other mobile devices. With intelligent indexing, the device can turn on exactly when there is data of interest to it, thus saving energy. However, this problem is quite different from the one under investigation in this paper where we are specifically looking at storage organization and query processing. Further, we are interested in queries that may be different for each user, rather than look at information that is needed by all users (broadcast data).

With the recent popularity of numerous mobile devices (especially PDAs), there has been a sudden plethora of applications that have been thrust on these devices. A popular one that several vendors seem to offer [GEO, Mic] is a version of a road atlas (a simple spatial database application), allowing the user to get driving directions (shortest path problem), examine detailed map information (range queries), give details on a landmark/restaurant that the user points to (point queries), and so on. However, many of these applications have been developed in an ad hoc manner, and there is no published result so far on the energy consumption of these applications or the energy optimizations that have been performed. Further, several such offerings are very primitive in terms of the querying support that they support (for instance, proximity queries are not well supported). On the other hand, a detailed investigation of the energy consumption of storage organizations and querying can provide a much more systematic way of designing and implementing an energy-efficient SDBMS for resource-constrained environments. This can also suggest hardware enhancements to improve the performance and energy savings of such systems.

3 Spatial Structures Under Consideration

As was mentioned in the previous section, there are numerous spatial data organizations that have been proposed [Sam89, GG98] and exploring the energy behavior of all these structures is well beyond the scope of this paper. Rather, we select three previously proposed structures - PMR Quadtrees [HS91, HS92], Packed R-Trees [KF93], and Buddy Trees [SK90] - that have been argued to perform relatively well for a range of datasets [GG98]. These structures are also representative examples from the design space of storage structures for spatial data. In Quadtrees, the index nodes at the same level have non-overlapping spatial extents, while R-trees and Buddy-Trees allow overlaps. Quadtrees are improvements over spatial partitioning techniques such as Grid Files, while R-trees are extensions of B-trees for spatial data. Buddy trees are representative of hashing based schemes using a tree structured directory. R-trees give more balanced structures than Quadtrees or Buddy trees.

As for the datasets, we consider line segments in a two dimensional space in this study. We believe that this does not significantly impact the main contributions of this work. Line segments represent an important class of datasets, especially in the road atlas applications for the mobile devices. Line segments (or polylines) can be used to represent streets, rivers, etc. Other related studies have also used line segment datasets [HS91, HS92]. In all the structures, the line segments are sorted based on the Hilbert-order [Gri86] of their centroids and kept in an array. The leaf nodes of the structures have pointers (index into the array) to the actual data items. As was mentioned in Section 1, we do not consider dynamic structures in this study, and assume that all the data items are pre-loaded into the memory-resident database (and do not change).

We consider three kinds of queries that have been identified [HS91, HS92] as important operations for line segment databases:

- **Point Queries:** In these queries, the user is interested in finding out all line segments that intersect a given point. For instance, such an operation could be used to find out which streets meet at a given intersection.
- **Range Queries:** These are used to select all line segments intersecting with a specified rectangular window. Very often, the user wants to magnify a portion of the atlas for a closer examination, and this query can serve such a request.
- **Nearest Neighbor Queries:** These are proximity queries where the user is interested in finding the nearest line segment (street) from a given point (e.g. what is the closest street to a given landmark, subway station, etc.). This is

the perpendicular distance to the line segment if the perpendicular intersects the segment, and is the distance to one of the end points (closest one) otherwise.

Range and Point queries are typically implemented using a *filtering step* where the possible candidates are first identified using their minimum bounding rectangles (MBRs). Each index node of the hierarchical spatial structures represents a rectangular region of the spatial extent that it covers, and is represented by the MBR of this region. The filtering step, that traverses the index structure, uses these MBRs to identify possible candidates. Subsequently, a *refinement step* is needed to perform the actual geometric operations on each short-listed data item to find the exact answers to the query. In structures (Quadtrees) that do not allow overlapping ranges between the index nodes at the same level, a line segment that spans more than one range needs to be replicated in all those ranges (we do not consider clipping based approaches that break a segment into multiple parts for each region that it falls in, and recombine/reconcile them in the refinement step). This does not need to be done for structures that allow overlapping ranges such as R-trees. As a result, part of the refinement step for Quadtrees involves duplicate elimination as well.

The Nearest Neighbor query is a little more complicated to implement for index structures, with different previous suggestions [HS91, Pat98, R⁺95, PM97]. For instance, [Pat98] uses a progressively expanding (in size) range query centered around the query point till the first data item is found. Another possibility [HS91] is to actually go to that region of the index structure, and examine around this region in the structure instead of composing the searches as separate range queries. A more interesting, and perhaps more efficient, approach is studied in [R⁺95] that is the strategy used in this paper. The search starts at the root node and examines the MBRs of its children. It orders these MBRs in terms of distances from the query point, and uses these distances to determine the recursive search order. In addition, it also uses these distances to prune the search when noticing that certain MBRs will definitely contain data items that are closer than those for the other children. The process is then recursively carried out for the candidate child nodes. This is a general technique that can be used for any of the considered hierarchical spatial access methods. The nearest neighbor query does not have separate filtering and refinement steps in our implementation.

3.1 PMR Quadtrees

The PMR Quadtree proposed in [HS91, HS92] is a member of a family of data structures that adaptively sort the line segments into buckets of varying size. Line segments are inserted one-by-one into an initially empty block. When the block reaches its capacity, it is split into four blocks of equal size. A line segment needs to be inserted in all the blocks (at the leaf level) that intersect it, and each of those blocks need to be again checked for capacity and possibly split into four again. One of the properties of this structure is that a block is split only once, and never again (it is no longer at the leaf level).

In the end, all internal nodes have pointers (their MBRs are implicit) to four children, while the leaf node has pointers to the actual data items that fall within its region. A consequence of this structure is that a line segment could be contained in more than one block/subtree (all those that intersect it), which makes it necessary to check for duplicates in the final refinement. Also, the structure may not necessarily be balanced. However, there is the potential of searching fewer subtrees when responding to the query because their spatial coverages are disjoint. Depth-first search is used to traverse this hierarchical structure in our implementation.

3.2 Packed R-trees

The R-tree [Gut84] has been proposed as an extension to the B-tree structure for handling multidimensional data. Many variants of the R-tree, such as R^+ -tree [SRF87] and R^* -tree [BKSS90] have been studied. They differ in the algorithm that is used for insertion, specifically in splitting a node of the tree when its subtree is filled. They attempt to give better balanced (and efficient) trees by dynamically adapting to the insertion pattern/sequence. However, these structures can become inefficient when the database of spatial items is static (and known *a priori*). In such cases, one should use bulk-

loading techniques rather than insert item by item to build the data structure. Roussopoulos and Leifker [RL85] use packed R-trees for such static databases to lower response times. Further, Kamel and Faloutsos [KF93] suggest using Hilbert value (a linearization technique for multidimensional space [Gri86]) for sorting the data items before constructing the bulk-loaded R-tree. This is the structure that is evaluated in this paper. Typically, such R-trees are built in a bottom-up fashion, level by level. After the line segments are sorted, for each line segment, starting from the first and going one after another, a pointer and its MBR are entered into an index node. When the number of pointers exceeds the node capacity, a new index node is created. After all the lines are assigned pointers and MBRs, index nodes for the next higher level are created to point to the index nodes at the lower level, and the process continues recursively till we get a single root index node.

Along with each pointer, we keep track of the MBR of the area that is covered by that subtree. The R-tree structure allows MBRs of pointers to overlap, which actually helps keep it more balanced than a Quadtree. However, the downside to this is that a search has to traverse more possible paths in the hierarchical structure. Depth-first traversal is used to implement the queries.

3.3 Buddy-Trees

The Buddy-Tree [SK90] can be regarded as a compromise between the R-tree [Gut84] and Grid File [HN83]. It is different from Grid Files in that it does not partition spatial regions that do not contain any data items. It is different from R-trees in that the spatial partition into which a data item can fall are pre-determined (similar to Quadtrees). The Buddy-Tree was originally proposed for points [SK90](which fall in exactly one region), and later extended [See91] to work for other objects allowing for overlapping buckets/index nodes. In this study, the overlapping regions strategy proposed in [See91] is used to handle line segments.

The construction of the Buddy-Tree can be briefly explained as follows. Data items are inserted into an initially empty bucket (that corresponds to the spatial extent of the dataset), till its capacity is reached. Subsequently, it cuts the space into two (along one of the dimensions), creates an index node with two pointers, each pointing to a different leaf node (containing the data items in its half of the split region). In case the split results in one of the regions being empty, then the algorithm splits the other region once again in the other dimension. These actions are recursively repeated as leaf nodes get filled to their capacity. The dimension of the split is alternated at each level. Essentially, the centroid of the line segment is used to determine where the line segment falls in the tree structure. The MBRs for this bucket and the nodes going all the way up to the root are adjusted to account for the size/shape of this line segment (which can result in MBRs of sibling index nodes to overlap). Depth-first traversal is used to implement the queries.

4 Experimental Setup

All the above index structures and queries have been implemented and evaluated using the energy estimation framework and workloads discussed below.

4.1 Energy Estimation Framework

Energy consumption is the integral of the power consumed over operating time. There are *three sources* for power consumption in CMOS circuits that are widely used in current computing devices [RP00]. The first source is the *logic transitions* (e.g., going from logic value of zero to one) that occur at the internal nodes of a circuit. This causes power to be drawn for charging the capacitance associated with these nodes, and is called *switching power*. The switching power is also influenced by the difference in the actual supply voltages used to represent logic values one and zero. The smaller the difference, the smaller the switching power. *Short-circuit currents* that flow directly from the supply to the ground when the inputs to a circuit transition are the second cause of power consumption. Both switching and short-circuit components of power consumption are dependent on the transition activity of the inputs, and together constitute the *dynamic power consumption*.

The third source of consumption is the *leakage current* that flows even when the inputs do not change, and is called *static power consumption*.

While the static power consumption of a mobile device could be a dominant factor when it is in a standby or quiescent mode, its dynamic power consumption is much more significant when it is active [CB95]. Since this paper is investigating the energy behaviors of different spatial index structures on a mobile device and they are relevant only when the mobile device is active, the dynamic power consumption naturally becomes the main focus of the paper.

The dynamic energy consumed in a mobile device can be expressed as the sum of the energies consumed in different components such as the *processor datapath*, *caches*, *clock distribution network*, *buses*, and *main memory* (we are not considering disks since we are dealing with memory resident datasets). Other peripherals, such as the display, are not under consideration here since our focus is mainly on energy expended in query execution (it has also been shown that around 52% of the energy is expended by the components under consideration on some mobile devices [RJ96]). The activity, and consequently the energy consumed by these components, is determined by the program executing on the system. The program can modify the number of transitions in the components (note that this affects switching power) by altering the input patterns, or reduce effective capacitance by reducing the absolute number of accesses to high-capacitance components (e.g. large off-chip memories). Note that transition frequency and effective switch capacitance are key contributors to the dynamic energy consumption, together with supply voltage.

Our energy estimation framework uses *SimplePower*, an architectural-level, cycle-accurate execution-driven energy simulator that is available in the public domain [VKI⁺00]. The architecture of the simulated system includes a single-issue five-stage pipelined integer datapath (instruction fetch (IF), instruction decode/operand fetch (ID), execution (EXE), memory access (MEM), and write-back (WB) stages), on-chip instruction (I) and data (D) caches, that is connected to an external (off-chip) memory. This architecture is representative of some of the current commercial offerings in the PDA domain [VKI⁺00]. The instruction set architecture is a subset of the instruction set (the integer part) of *SimpleScalar*, which is a suite of publicly available tools to simulate modern microprocessors [BA97].

The major components of *SimplePower* are: SimplePower core, RTL power estimation interface, technology dependent switch capacitance tables, cache/bus simulator, and loader. The *SimplePower* core simulates the activities of all the functional units and calls the corresponding power estimation interfaces to find the switched capacitances. These interfaces can be configured to operate with energy tables based on different micron technologies. Transition sensitive, technology dependent switch capacitance tables are available for the different functional units such as adders, ALU, multipliers, shifter, register file, pipeline registers, and multiplexors. The *SimplePower* core continues the simulation until a predefined program *halt* instruction is fetched. Once the simulator fetches this instruction, it continues executing all the instructions left in the pipeline, and then dumps the energy statistics. SimplePower provides the total number of cycles in execution and the energy consumption in different system components (datapath, clock, cache, memory and buses) as explained below.

Processor Datapath: The energy consumed in the processor datapath is dependent on the number, types, and sequence of instructions executed. The instruction type determines the components in the datapath that are exercised while the number of instructions determines the duration of the activity. Whenever a component is exercised, there is a switching activity in that particular component contributing to dynamic energy consumption. For example, when executing an integer addition instruction, energy is consumed in the instruction cache and instruction fetch logic in the first stage of the pipeline, in the register file when accessing the source operands in the decode stage, in the ALU when executing the operation, and, again, in the register file during write-back. It must be noted that energy is also consumed in the pipeline registers of the datapath, and that the components in the memory stage of the pipeline are not exercised by this instruction.

Caches: The cache simulator of *SimplePower* is interfaced with an analytical memory energy model based on [SC99]. The memory energy is divided into those consumed by the cache decoders, cache cell array, the buses between the cache and

main memory, and the main memory. The components of the cache energy is computed using analytical energy formulations and is dependent on the number of cache accesses, number of misses, the cache configuration (e.g., associativity, capacity, line size), and the extent of utilizing energy-efficient implementation techniques (e.g., bitline segmentation, bitline isolation, pulsed wordline). Energy is expended in the row decoders when a particular cache line is selected for a read or write operation, in the lines that activate each cell (wordline) in a particular row of the cache line, in the bitlines when the values are written or read from the cells, in the sense amplifiers to amplify the values read, and finally, in the column decoders that select a part of the activated cache line. The energy consumed during reads and writes vary since the voltage swings in the bit lines vary during these operations (full swing for writes and partial swing for reads). Consequently, it is important to estimate the number of reads and writes individually. The energy consumed in the caches is largely independent of the actual data accessed from the caches, and prior work has shown that the number of cache accesses is sufficient to model energy accurately [GK99].

Memory: The organization of the main memory is similar to that of the caches, with two main differences. First, the memory arrays have no tag comparison portion. Second, the basic cell for implementing memory storage (DRAM cells) is different from that used in on-chip caches (SRAM cells). Consequently, there is a difference in the energy consumed during read/write accesses. Typically, the energy cost of accessing memory is larger than that of on-chip caches because of the additional costs associated with off-chip packaging capacitances and also due to the energy consumed in refreshing the DRAM cells. The energy consumed in the memory can be again analytically modeled fairly accurately by capturing the number of accesses and the interval between the accesses obtained from a cycle accurate simulator [D⁺01a].

Buses: The buses are used to communicate the addresses from the datapath to the caches and memories, and to transfer the data between these units. The energy consumption is quantified analytically based on the number of bus transactions, the width of the buses, the switching activity on these buses, and bus line capacitance. The switching activity here is assumed to have equal probability 50% since no exact information about it could be obtained.

Clock Network: The simulator also reports the on-chip clock energy using detailed models [D⁺01c]. The components of the clock network that contribute to the energy consumption are the clock generation circuit (called PLL), the clock distribution buffers and wires, and the end-load on the clock network presented by the clocked components. The energy consumed in a single cycle depends on the parts of the clock network that are active. The PLL and the main clock distribution circuitry are normally active every clock cycle during execution. However, the participation of the end-load varies based on the active components of the circuit as determined by the software executing on the system. For example, the clock to the caches are gated (disabled) when a cache miss is being serviced. Thus, actual execution and stall cycles can be used to calculate the clock energy. Thus the energy in the stall cycles do not manifest in the datapath component in our statistics, but appear in the clock network. If interested in further information on the energy models of the clock network, one can find them in [D⁺01b].

The energy results from the simulator have been shown to be within 8% of those measured from a real system [VKI⁺00]. All results reported in this paper are obtained using the parameters given in Table 1.

We also model a wireless network interface (that is capable of transmission bandwidths of up to 11 Mbps) which offers different operating modes - Sleep (0.02 watts), Idle (0.1 watts), Receive (0.165 watts) and Transmit (3.09 watts), based on its current functionality. This model has been drawn from [SCI⁺01]. We use this interface in one set of experiments in section 5.8, and is assumed to be absent otherwise.

4.2 Workloads

In our experiments, we use four line segment datasets as explained below:

Parameter	Value
Supply Voltage	3.3 V
Cache Sizes (each of I and D)	8KB, 16KB, 32KB (32 bytes line size)
Associativity	Direct-Mapped (DM), 2-way, 4-way
Data Cache Hit Latency	1 cycle
Memory Size	8 MB
Memory Access Latency	100 cycles
Per Access Energy for DM-Caches (in nJ)	0.048 (8K), 0.082 (16K), 0.094 (32K)
Per Access Energy for Memory	3.57 nJ
On-Chip Bus Transaction Energy	0.069 nJ
Off-Chip Bus Transaction Energy	6.9 nJ
Per Cycle Clock Energy	0.18 nJ
Technology Parameter	0.35 micron

Table 1: Base configuration parameters used in the experiments.

- **NYCS**: containing 12355 streets of New York City, taking 1.14 MB, from the Tiger Database [otC].
- **PAFS**: containing 16431 streets in Pennsylvania Fulton county, taking 1 MB, from the Tiger Database [otC].
- **SVR**: containing 5848 rivers from the Shenandoah valley, taking 106 KB, from the Tiger Database [otC].
- **IRR**: containing 12338 railway tracks of Italy, taking 468KB, from the Digital Chart of the World [Ins].

The datasets are also representative of some of the SDBMS applications on mobile devices. NYCS and PAFS are typical of road atlas applications for navigation and locational information, the former is for a city and the latter for a rural county. SVR is a dataset that could be useful for hikers/environmentalists on the trails. Finally, IRR is from a different database and would be useful to find the nearest railway track, finding the identity of a station on a track, etc., with the queries that we are considering. A pictorial view of the four datasets is shown in Figure 1.

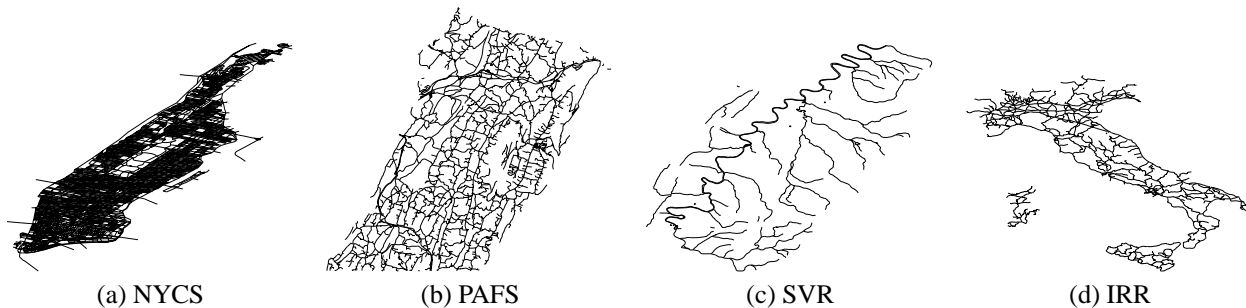


Figure 1: Datasets

On these datasets, we use the results from 100 runs for each of the three kinds of queries (Point, Range and Nearest Neighbor). Each run uses a different set of query parameters. For the Point queries, we randomly pick one of the end points of line segments in the dataset to compose the query. For the Nearest Neighbor queries, we randomly place the point in the spatial extent in each of the runs. For the Range query, the size (between 0.01% and 1% of the spatial extent), aspect ratio (0.25 to 4) and location of the query windows is chosen randomly from the distribution of the dataset itself (i.e. a denser region is likely to have more query windows). The results presented are the sum total over all 100 runs.

The code sizes for the implementation of the index structures and the storage sizes of the index structures (not including the space taken by the dataset) are given in Table 2 for the chosen fan-outs (see Section 5.1). As far as the code size is concerned, the Quadtree code is a little larger because of the duplicate elimination code that is absent in the other two (the code for building the structures is not included in these sizes). Despite these minor differences, the code size is not very

Index	Code Size	NYCS	PAFS	SVR	IRR
Quadtree	39KB	150KB	183KB	59KB	133KB
R-tree	35KB	285KB	378KB	135KB	285KB
BuddyTree	38KB	670KB	989KB	344KB	732KB

Table 2: Code Size and Storage Overheads for the Index Structures

different across these structures. R-tree incurs more storage overheads because of its more balanced nature. Despite the packed R-tree algorithm that is used, some nodes could still be under-utilized. The property of the buddy tree which keeps index nodes that are not entirely packed to capacity (could be much sparser than R-tree nodes), results in a much poorer space utilization compared to the other two structures.

As was mentioned earlier, we do not study the building costs for the structure since we are examining a static situation without dynamic insertions (and the storage structure is downloaded from a server on to the mobile device similar to how it is done in [Mic]). The chosen dataset sizes and their index overheads are also similar to some of the pocket atlas datasets (e.g., the New York City map that is available in the public domain for PocketStreets [Mic] running on Windows CE takes 865KB).

4.3 Metrics

We examine both the energy behavior as well as the performance profile for each execution. This helps us understand the trade-offs between the two if any, and also helps us explain the energy consumption based on the performance results.

For the energy behavior, we profile the consumption (in joules) by each of the hardware components - processor datapath, I-cache, D-cache, Memory, Buses (between cache and memory), and clock network. For the performance profile, we give the breakdown of the cycles spent by the processor performing useful work, and also when stalling on I-cache and D-cache misses.

These profiles are given for each of the query executions on each dataset using the different index structures, and compared with the Brute-Force approach. The profiles are also separately given for the Filtering and Refinement steps, to understand where the overheads are incurred from the software perspective. From the hardware perspective, the impact of different cache organizations on energy and performance behavior is also studied.

Energy consumption, execution cycles and the product of these two (denoted as energy*delay) are the key metrics that are used for comparison. Energy*delay helps us capture the relative trade-offs of how much energy savings can be obtained with one alternative over another without significantly degrading performance (or vice versa). We also present energy/cycles values capturing the average energy per unit time (power), which is important for packaging and thermal considerations.

Many of the results and trends are common across the datasets. As a result, *the graphs show the behavior averaged over all the datasets*. Whenever there is a dataset influence, the effects are explicitly mentioned in the discussion.

5 Experimental Results

5.1 Impact of Fan-Out

One of the important considerations for each index structure is the fan-out issue. For R-tree and Buddy-Tree, this corresponds to the number of (MBR, ptr) pairs at all levels of the hierarchical structure, with each such entry taking 20 bytes. In the Quadtree, the fan-out of the internal nodes is fixed at 4 entries (taking 80 bytes totally) as per the definition of the structure, and the only choice is for the number of pointers to maintain at the leaf level for the lines falling within this bucket (as suggested in [HS91]). Apart from the nature of the dataset itself, several factors govern the choice of a fanout. In a disk-based storage structure, the disk access times have a large influence on the choice of the fan-out, and it will be interesting to see how memory resident datasets affect this issue. We have varied the fan-out of the different structures, and

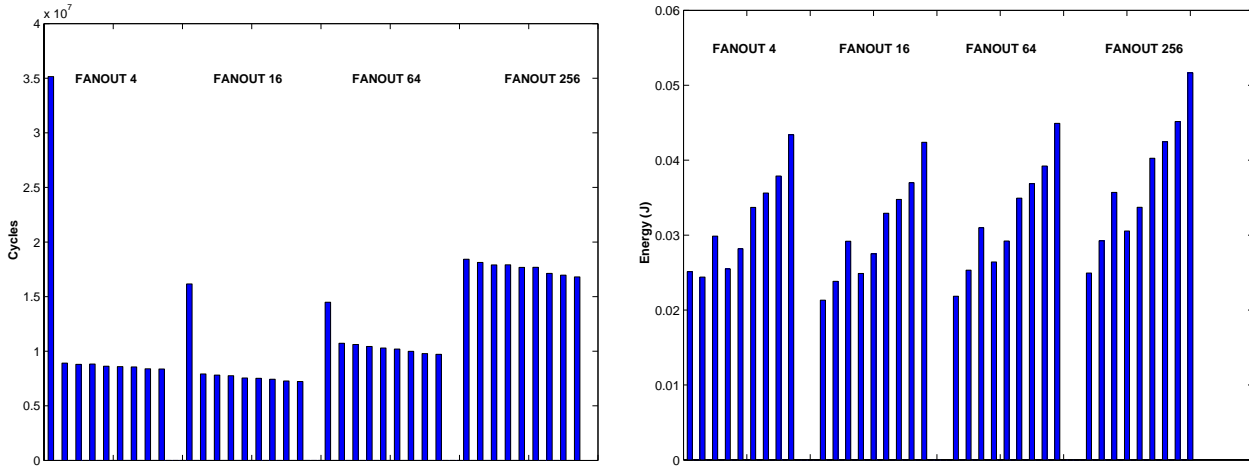


Figure 2: Impact of Fan Out of R-tree on Total Cycles (left) and Energy (right) for Range Queries with PAFS. For each configuration, the nine bars from left to right correspond to cache configurations of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way).

collected both performance and energy profiles for the different datasets. Figure 2 shows a representative result (results are similar for other datasets), illustrating the impact of fan-out with range queries for R-trees on PAFS. As fan-out increases, the depth of the tree goes down, thereby improving performance initially. On the other hand, the number of paths to be searched and the number of comparisons at each index node may increase, which worsens performance (in terms of CPU cycles). With these two contrasting factors, the best fan-out that we observe is at 16 for the R-trees as is shown in the figure (except for 8K direct mapped caches), which yields a node size of 320 bytes. We also observed a similar behavior for the fan-out of the leaf nodes of the Quadtree, where the ideal leaf node size again turned out to be 320 bytes (80 pointers). A fan-out of 16 was observed to give the best performance for the Buddy-Tree as well. These observations hold across cache sizes and associativities as can be seen in the graphs.

Another interesting observation is that the fan-out has a similar effect on energy consumption as performance, suggesting that using one of these metrics to optimize the fan-out may suffice in practice for the overall energy*delay savings. It should be noted that each query can demand a different fan-out, and it is difficult to predetermine this value unless we have a good idea of the workload imposed on these structures. Since range queries are usually much more prevalent, we have chosen a fanout for each structure that is optimized for the range queries, and use this ideal fanout for all our experiments (regardless of the query).

5.2 Results from Brute Force Method

Before we get to the index structures, we present results for the Brute-Force method separately in Figure 3 since its performance is much worse (and including its results in the same graph as the index structures increases the scale significantly, making it difficult to see any noticeable differences between those structures).

For the range and nearest neighbor queries, both the cycles and energy consumption are an order of magnitude higher (if not more) than those with the index structures (compare with Figures 6 and 8). In these queries, the predicate checking for every data item is significantly higher than that incurred for a point query. Since the overall execution energy is a function of both the number of data items checked and the cost of the check, range and nearest neighbor queries are much more expensive in Brute Force compared to the point queries. Since the brute force approach is significantly worse from both performance and energy viewpoints, we do not consider it any further in this paper.

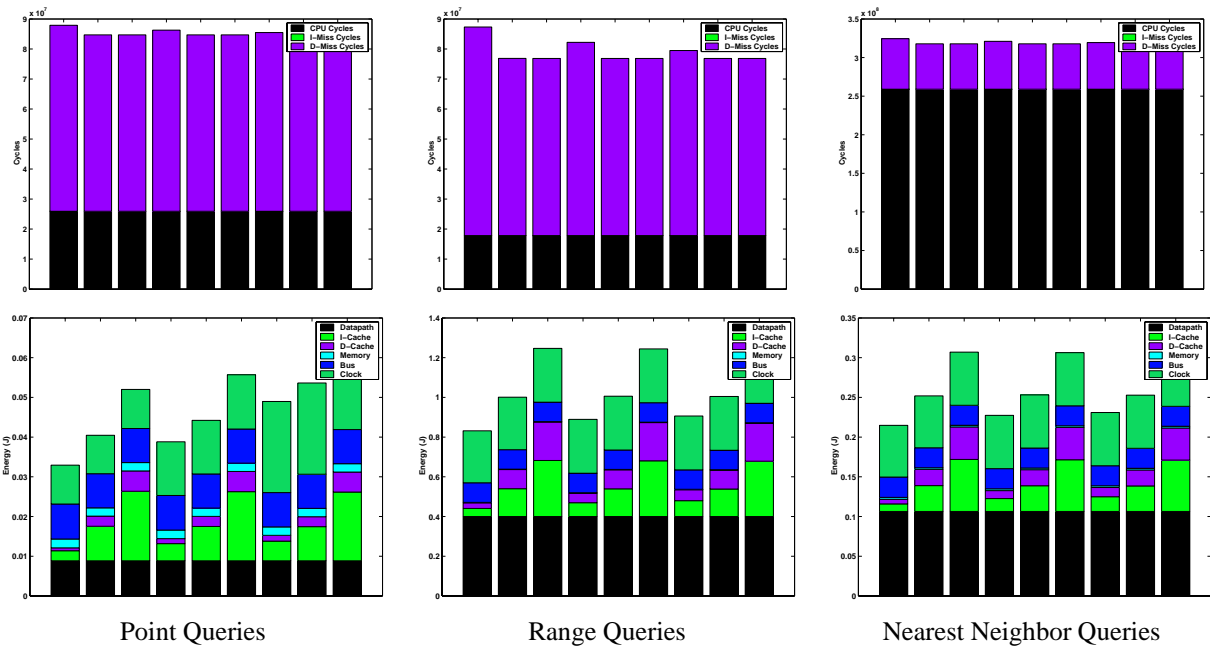


Figure 3: Performance and Energy of Brute Force method for the three queries averaged over the four datasets. In each graph, the nine bars from left to right correspond to cache configurations of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way)

5.3 Results for Point Queries

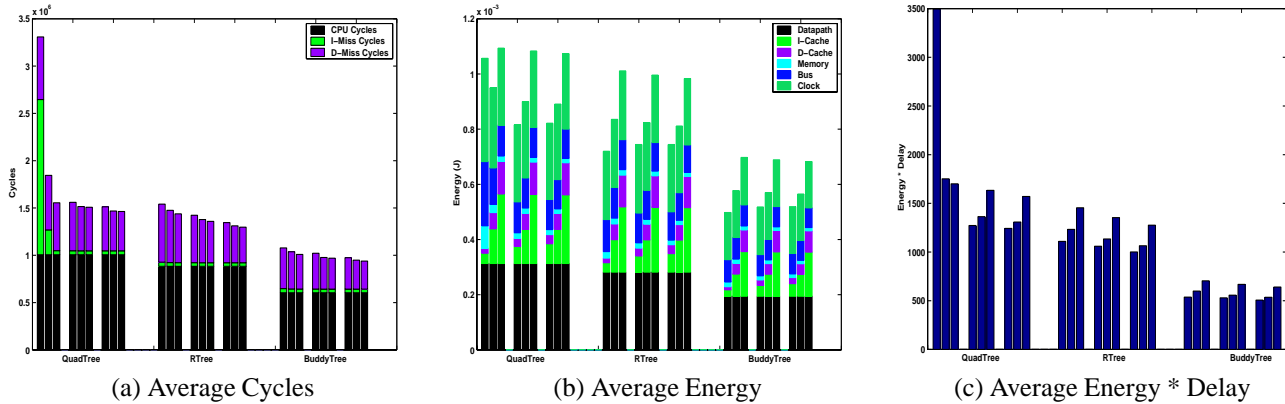


Figure 4: Comparison of Index Structures for Point Queries. The nine bars from left to right for an index correspond to cache configurations (cache size, cache associativity) of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way).

Figure 4 shows the performance, energy and energy*delay profiles for the point queries with the different schemes averaged over the four datasets.

Examining the execution cycles graph, we see that Quadtree has a higher processor cycle count compared to R-tree. Since the Quadtree does not allow overlaps (and a point query does not have a high probability of falling in more than one bucket), the filtering step on the Quadtree to find the candidate leaves is relatively fast. This is evidenced by the execution profile for the filtering step in Figure 5 which shows the cycles for Quadtree are significantly lower than those for the R-tree. However, the refinement step graph in the same figure shows that Quadtree is much more time consuming, placing the sum of these two steps slightly in favor of the R-tree. It should be noted that the overhead for Quadtree in the refinement step is mainly due to the larger number data items (even though the leaf nodes in the two structures have the same size - 320 bytes, this corresponds to 80 data items in the Quadtree which stores only pointers and to 16 data items in the R-tree which stores

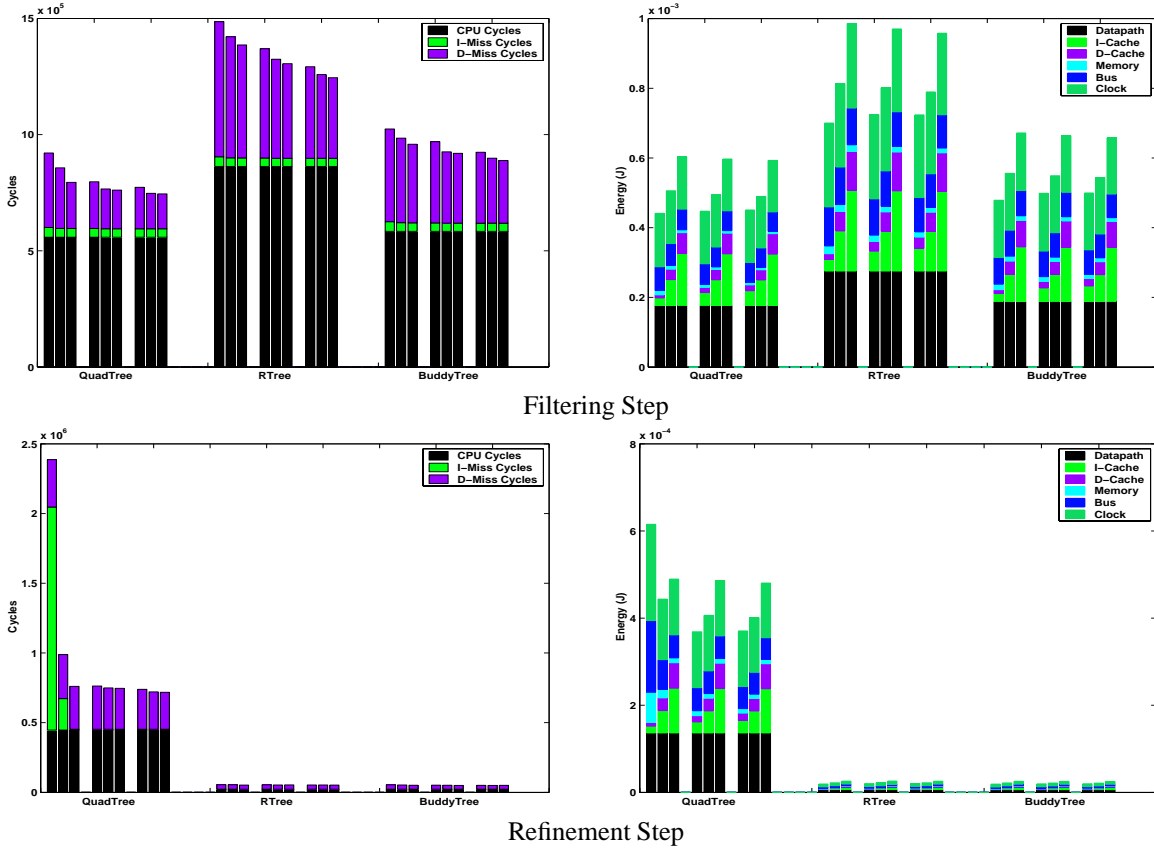


Figure 5: Comparison of Index Structures for Point Queries: Average Cycles and Energy for Filtering and Refinement Steps. The nine bars from left to right for an index correspond to cache configurations of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way).

pointers and MBRs) that the Quadtree has to deal with in the refinement step. The reason we chose the 80 data items fanout in our implementation was because it gives good performance for range queries (as mentioned earlier). Though the graphs are not explicitly shown, we would like to point out that a 16 data item fanout (same as the R-tree leaf node), does indeed give better performance for point queries with Quadtrees, cutting down the overhead of the refinement step, thus making the Quadtree performance similar (or even slightly better in some cases) to the R-tree. Between the R-tree and Buddy-Tree, we find the latter giving better performance. This is mainly due to the splitting criteria for a node, where the Buddy-Tree partitions based on spatial locations while the packed R-tree just uses Hilbert order groupings.

We find that Quadtree has better data locality than R-tree, which can be explained with the higher internal node fanout and overlapping buckets in the latter. With overlapping buckets, R-tree may entail searching more paths even with a point query (while the Quadtree is more focussed). Since the fanout of the internal nodes is higher, there is more scope for eviction of data items from the cache that may be needed again. This is evident by examining the D-misses in the filtering step in Figure 5. The Buddy-Tree locality falls between these two.

It is difficult to comment on the I-cache locality behavior of the different codes without clearly understanding where the called procedures fall within the code segment and how they reference each other (for conflicts). In general, we find that 8K caches direct-mapped caches are not a good idea for the I-cache (which is true in later queries as well). Most of the penalties are reduced with a 16K 2-way I-cache.

From the energy perspective, we find that the components' energies have a strong dependence on the time spent in the cache miss cycles (stalled cycles) vs. CPU cycles (non-stalled cycles). The datapath (and the resulting clock) contribution to the overall energy consumption, for instance, is considerable reflecting the importance of the CPU cycles spent in the

instruction execution shown in the performance graph. Overall, the differences between the three indexes in terms of energy consumption reflect the same observations that were made between them from the performance perspective. As a result, for this set of experiments the energy and performance results go hand-in-hand to a large extent. The only exception to note is the I-cache and D-cache energy consumption changes as we change the cache configuration. With improved (larger size or better associativity), the miss rate is expected to go down, but at the same time energy cost incurred per access goes up. These two factors can help us decide on a good energy-delay conscious cache configuration (captured by the energy*delay values in Figure 4(c)). With 16K and 32K (I and D) caches, most of the locality in instruction and data references is captured well by these configurations, and the energy increase with associativity is more significant. As a result, with these cache sizes, it would be better to have a direct-map structure from the energy*delay perspective (see Figure 4(c)). With 8K I and D caches, we find that the performance penalties due to conflict misses are quite severe, preferring a higher associativity from the energy*delay perspective. For the Quadtree (especially due to its high I-cache misses), an associativity of 4 is needed, while R-tree and Buddy-Tree give the best energy*delay for a direct-mapped cache.

Despite the depiction of lower cycles, energy and energy*delay for R-tree over Quadtree in Figure 4(c), we would like to reiterate that these differences are mainly due to the differences in the chosen fanouts. We believe that these two structures are more or less comparable in terms of these metrics if we fine-tune the fan-out values at the leaf level for the Quadtree to suit this query. In terms of all these metrics, we find that Buddy-Tree delivers the best results for point queries.

5.4 Results for Range Queries

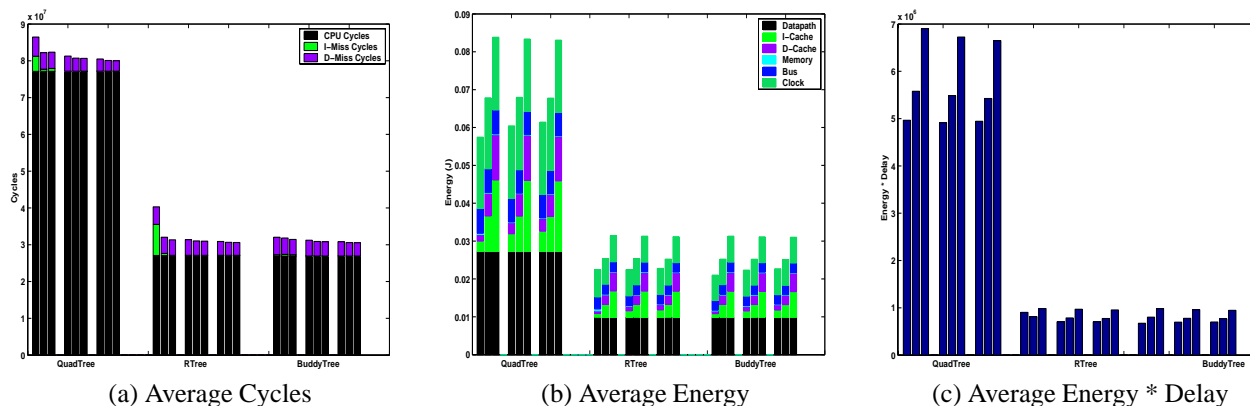


Figure 6: Comparison of Index Structures for Range Queries. The nine bars from left to right for an index correspond to cache configurations (cache size, cache associativity) of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way).

Figure 6 shows the performance, energy and energy*delay profiles for the range queries with different schemes averaged over the four datasets.

Rather than repeat all the observations that are similar to those for the point query, we would like to point out the differences. The first noticeable difference is that the Quadtree performs much worse than the R-tree and Buddy-Tree in terms of both performance and energy (despite having chosen a fan-out that gives the best performance for the Quadtree). Compared to the point query, range queries have higher likelihood of covering spatial extents of more than one leaf node. As a result, the searches are not that focussed any more on a Quadtree, and more than one path may need to be searched. Second, since the region boundaries of a Quadtree’s index nodes are pre-determined and are not adapted to a dataset’s vagaries, there is the scope for traversing more paths in a Quadtree compared to the R-tree (this can be seen by the differences in their performance for the filtering step in Figure 7). Finally, non-overlapping boundaries of index nodes can result in a data item being replicated, and duplication elimination is time-consuming for the Quad-tree (as seen by the performance for the

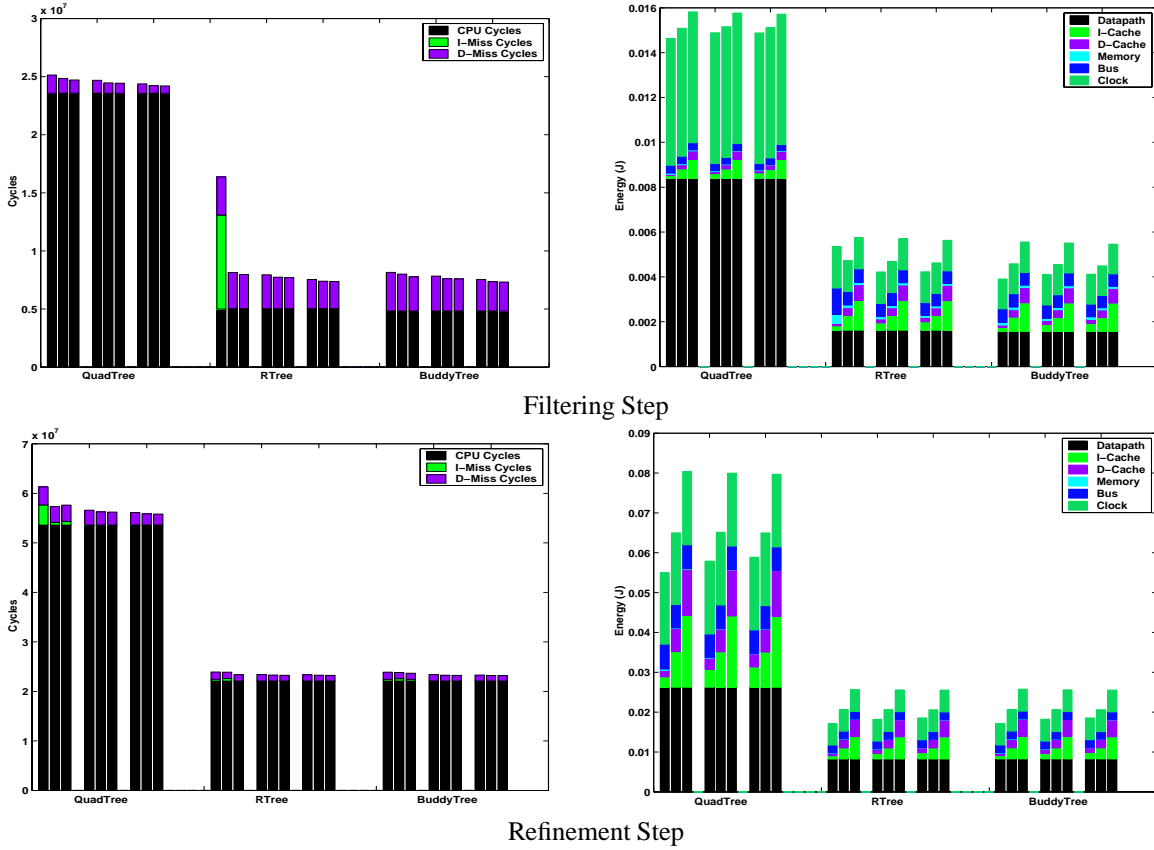


Figure 7: Comparison of Index Structures for Range Queries: Average Cycles and Energy for Filtering and Refinement Steps. The nine bars from left to right for an index correspond to cache configurations of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way).

refinement step in Figure 7). In the filtering step, all candidates are inserted into a list. The refinement step for the Quadtree first sorts this list to remove duplicates, and then performs an item-by-item comparison. These overheads materialize in the refinement step performance, and is also the reason why Quadtree has slightly higher data misses for the refinement step compared to the R-tree and Buddy-Tree.

In general, we find that range queries are more processor intensive than point queries, with a smaller fraction of the time spent stalling on cache misses for both index structures. The significance of the refinement step which has good locality (due to sequentially searching a list for exact matches) in the overall performance picture is the main reason for this behavior (see Figure 7).

We find that performance is the main factor governing these schemes when we examine them from the energy and energy-delay perspectives. Higher number of instruction executions imply a larger datapath energy and clock energy. At the same time, each instruction fetch references the I-cache, incurring an energy cost (even when it is a hit). For instance, we can see a noticeable difference in the I-cache energy costs between the Quadtree and the other two in the refinement step which is a time-consuming fraction.

In the point queries, we could see both performance and energy impact of cache configurations playing significant roles when determining a good operating point in both R-trees and Quadtrees. In the range queries, we find that the energy*delay metric obeys the performance (cycles) trend in nearly all cases (except for the Buddy-Tree with 8K 4-way caches). Both R-trees and Buddy-Trees do a good job for this query along all three perspectives - performance, energy, and energy-delay, with Buddy-Trees having a slight edge.

5.5 Results for Nearest Neighbor Queries

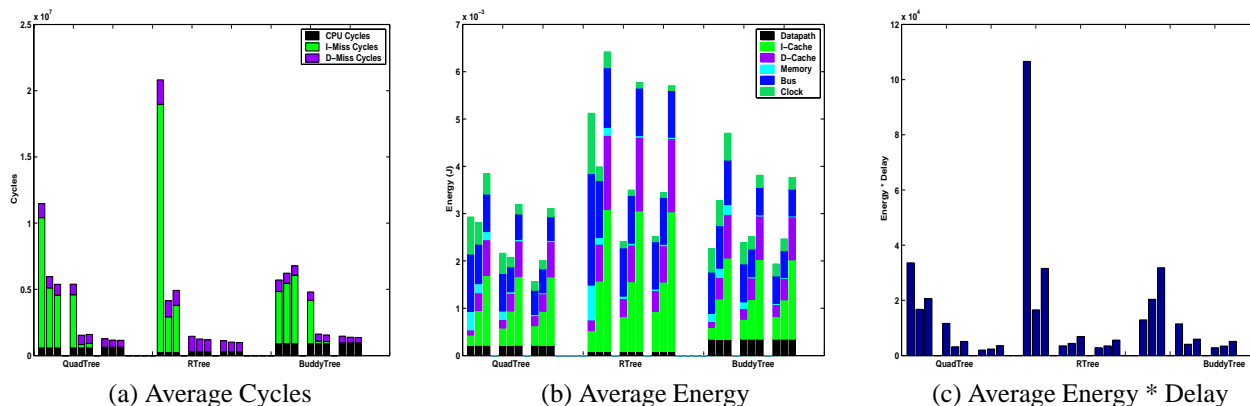


Figure 8: Comparison of Index Structures for Nearest Neighbor Queries. The nine bars from left to right for an index correspond to cache configurations (cache size, cache associativity) of (8K,DM),(8K,2way),(8K,4way),(16K,DM),(16K,2way),(16K,4way),(32K,DM),(32K,2way),(32K,4way).

Figure 8 shows the performance, energy, and energy*delay profiles for the nearest neighbor queries with the different schemes averaged over the four datasets.

The nearest neighbor query presents an entirely different picture from what we have observed in the previous two queries. Compared to the previous two, the results show that cache misses dominate the execution time, and processor cycles are a much smaller fraction in many of the datapoints. In the first place, there is no separate refinement step for this query as was mentioned earlier, with data items examined closely when they are first encountered. Even with the earlier queries we found that misses are more significant in the filtering step (tree traversals) than in the refinement step. Further, the working set sizes for implementing the nearest neighbor algorithm (explained in Section 3) are higher than for point/range queries. Specifically, when traversing a subtree, closest distances need to be calculated for all children and they need to be sorted and pruned, before recursively traversing them. Point/Range queries can examine children one at a time, moving to the next after traversing the subtree under the previous child. These operations make the nearest neighbor query much more dependent on miss penalties. The number of D-misses is closely related to the number of children (fanout of internal nodes), which also explains why R-tree has higher D-cache misses compared to Quadtree and Buddy-Tree. The I-misses show a reverse behavior with R-trees having better code locality (except for the 8K DM case) than Quad-trees. Since R-tree has a larger fanout and lower depth, the sorting/pruning operations and overheads are amortized over a larger number of children at a time, while the Quadtree and Buddy-Tree may keep switching between traversal and pruning more often. Overall, from the performance viewpoint, we find that R-tree does the best except for the 8K DM cache. Of the other two, the Quadtree outperforms the Buddy-Tree in many cases.

While there was not a noticeable difference in the relative performance of the schemes across the datasets for the previous two queries, we would like to mention that there is a difference between the datasets for this query with the Buddy-Tree structure (there was not a significant effect on the other two indexes). In datasets that are much more clustered (NYCS and IRR), the Buddy-Tree incurred more processor cycles than the others since it does not do as good a job as the R-tree (or even the Quadtree) in balancing the hierarchical structure to reduce the number of levels. For the other two datasets, its performance becomes comparable to the R-tree.

The most interesting observation with this query (compare Figures 8(a) and (b)) is that **better performance does not necessarily imply better energy** (except in 8K DM). R-tree takes fewer cycles to service the query, while Quadtree takes lower energy (with Buddy-Tree energy falling in between). The reason for this behavior can be explained as follows. R-tree incurs much lower CPU cycles than the Quadtree, but incurs higher cache misses. Miss penalties (which require crossing

Index	Query	Phase	Energy/Cycles
QuadTree	Range Query	Filtering Step	31.88
		Refinement Step	34.99
QuadTree	Point Query	Filtering Step	31.31
		Refinement Step	30.81
QuadTree	Nearest Neighbor Query	-	37.82
RTree	Range Query	Filtering Step	32.22
		Refinement Step	35.86
RTree	Point Query	Filtering Step	31.31
		Refinement Step	31.33
RTree	Nearest Neighbor	-	38.36
BuddyTree	Range Query	Filtering Step	32.29
		Refinement Step	35.71
BuddyTree	Point Query	Filtering Step	32.19
		Refinement Step	31.97
BuddyTree	Nearest Neighbor Query	-	38.16

Table 3: Influence of Indexing Scheme and Query Type on Datapath Power Consumption (nJ/cycles) for PAFS

pin boundaries and bus to get to main memory) translate to much more overheads in terms of energy (*off-chip energy*) compared to performance. While the additional miss cycles are not significant enough to put R-tree overall cycles higher than Quadtree, the miss energy (in D-cache, Bus and memory) overhead compensates for any savings in the lower datapath energy (e.g. compare the datapath, D-cache, bus and memory energy components for the R-tree with that for the Quadtree for the 32K 2-way caches). The Buddy-Tree energy falls between that for R-tree and Quadtree in most cases.

A consequence of the differences between the energy and performance behavior for this query is the interesting observation in the resulting energy*delay metric shown in Figure 8(c). This captures the facets of whether the improvement in performance warrants the additional energy that is expended. The results show that *even though R-tree is better in terms of performance, Quadtree (which is better in terms of energy consumption) may be a better alternative from the energy*delay perspective* (i.e. the performance benefits for the R-tree come at a much higher energy cost that it may not be as attractive in energy-constrained environments) in most of the better cache configurations. The energy*delay of Buddy-Tree falls in between these two in many cases.

5.6 Examining Datapath Power

It is important to also take note of the *power* consumption in specific system components so as to not exceed a specified limit. This is important for uniform heat dissipation across a chip, since increasing the power consumption in one particular component can create a thermal hot-spot in the chip [V⁺00]. Here we focus on the datapath power. Datapath power is presented as energy consumed per cycle (power in watts can be obtained by accounting for cycle time).

It should be noted that the energy consumption and number of execution cycles can exhibit totally different behavior. For instance, two runs can complete within the same number of cycles, but, depending on the hardware components exercised during these cycles, can result in very different energy consumptions. Power consumption (energy per cycle) shows the average complexity of computations performed within a cycle and Table 3 gives the datapath power for PAFS. We observe that the nearest neighbor query is by far the most power consuming query independent of index structure. This is mainly due to the complexity of the geometric operations (stressing energy expensive units such as the multiplier) that are performed when examining each data item.

The relative power consumption costs of filtering and refinement depend on the type of the query as well. For example, in the Buddytree, with range queries, the refinement step is 10% more power consuming than the filtering step. In contrast, with point queries, the refinement step is 1% less power consuming than the filtering step. Again, the geometric operations for testing whether a point is one of the end-points of a line segment (during refinement for the point query), requires only

Index	PAFS	IRR	NYCS	SVR
BuddyTree	38.16	38.83	37.98	38.71
Quadtree	37.82	38.60	37.28	37.79
RTree	38.36	38.77	37.95	38.34

Table 4: Impact of Datasets on Datapath Energy/Cycles (nJ/cycles) for Nearest Neighbor Query.

simple comparisons and is less power consuming than testing the same with a MBR in the internal nodes (during filtering). For the range query, testing whether a line intersects the rectangular window (refinement) is more complex than testing whether a MBR interests the window (filtering).

Further, we can observe that *optimizing for energy and optimizing for power may result in conflicting choices*. For instance, the refinement step using point queries for the R-trees is more power efficient than that of the Buddy-Tree. But the situation is exactly the opposite in terms of datapath energy consumption (63 μ J compared to 61 μ J for Buddy-Tree).

All through this paper, we have not examined the impact of the dataset itself, and power dissipation is one issue where the differences are brought out. Table 4 shows the impact of differences in dataset on power consumption. We observe that the datapath power consumption varies as much as 3.5% due to the variation in datasets. In comparison, the impact of indexing schemes themselves for the nearest neighbor query is as much as 2.6%.

5.7 Impact of Architectural Innovations and Technological Trends

The previous experimental results have examined current architectures and technologies to evaluate the pros and cons of the indexing schemes. It is also important to consider the impact of technological trends and innovations on the spatial access methods, especially since the hardware capabilities of the mobile devices are constantly changing/improving. We specifically consider two approaches here that both focus on optimizations of the memory system.

5.7.1 Energy-efficient Cache Architectures

Uniformly, we find in the previous results that the cache energy is a significant consumer of the overall system energy (even over 50% in the nearest neighbor queries). It is not only on misses that energy is expended, but on all accesses including hits (which are very frequent). Further, these SDBMS workloads are data intensive, and are thus motivating factors behind energy-efficient cache design. While this issue is well-beyond the scope of this paper, we would like to briefly touch upon the possibilities and impact of energy-efficient cache design.

A common trend in energy-efficient hardware design is the partitioning of components into smaller units, and selectively activating the unit that is needed. This approach reduces the energy consumption per access, since the smaller unit takes less power during the activation. Such an optimization can be applied to the I and D caches as well, wherein a single monolithic cache can be partitioned into several smaller ones (subcaches) and selectively activating one of them. One simple way of storing data in a cache partitioned into two is based on whether the referenced data exhibits spatial or temporal locality [G⁺95]. The downside to this approach is to be able to selectively activate the subcache where the current data resides. For the I-cache, simple prediction strategies like Most Recently Used (MRU) subcache can give good performance, since instruction references usually have good locality. The data references on the other hand can be difficult to predict, especially for the database workloads considered here that have very dynamic reference patterns when traversing hierarchical structures. When the prediction fails, a performance (since other subcaches need to be searched subsequently) and energy penalty is incurred. There are also set associative cache organizations where non-accessed ways within a set can be disabled by predicting the way that is being referenced (called way prediction [ITM99]). To explore the feasibility and benefits of such energy-efficient subcache structures we conducted a simple experiment where we fed the D-cache references to: (a) a single monolithic 32 KB DM cache (DM); (b) a single 32 KB 2-way set-associative cache using way-prediction

to determine the next way within the set for selective activation (*Way-Prediction*); and (c) two subcaches (each of 16 KB) holding spatial and temporal data respectively, using access history to predict the subcache to be accessed next (*Subcaching*). Parameters used in this experiment such as the way-predictor accuracy and energy per access to sub-caches are taken from a simulation study in [KV⁺01]. Table 5 shows the resulting energy, access time penalty (delay) and energy*delay with these optimizations for the range query using Quadtrees on the PAFS dataset (values are normalized with respect to DM).

	Energy	Latency	Energy*Delay
Way-Prediction	0.7	1.15	0.8
Subcaching	0.4	0.8	0.32

Table 5: Impact of Cache Optimizations on Memory system energy and latency for Range Query using Quadtrees with PAFS normalized with respect to DM for D-Cache

The reason for this experiment is not to evaluate the subcache designs in detail or propose new strategies. Rather, we are only examining whether such a strategy would work for these SDBMS workloads with dynamic reference patterns, and if so how it would affect the relative performance of the indexing mechanisms. We find that there is definitely an energy benefit from these optimizations, and in some cases the latency can also be improved. We find that the overall energy*delay from such structures can give us as much as 68% energy*delay savings in the memory system energy stressing the importance of energy efficient caches. If partitioned caches are employed, then the resulting executions will favor those schemes with a large number of I-cache and D-cache hits. Specifically, this will help Quadtrees which has more memory references in general.

5.7.2 On-chip Main Memory (eDRAM)

While the previous technology provides energy optimizations for hits, it is important to study the technological trends affecting energy and performance on cache misses. For instance, embedded DRAM (eDRAM) is one such technology [Son97] where a portion of the main memory can be moved on-chip. This can significantly reduce miss penalty apart from reducing energy consumed by off-chip buses (transactions going out of the chip are expensive). To study how the indexing mechanisms would compare with eDRAMs of the future, we conduct an additional experiment. Using the method discussed in [D⁺01b], we derive the values of eDRAMs energy consumption for this experiment by scaling down the DRAM values by a factor of 10, and this factor is obtained from the comparison between an actual eDRAM implementation - M32RD and a conventional DRAM [NST97]. The miss latency is also cut down to 10 cycles from 100. The results of this experiment are shown in Table 6.

	Normal			With eDRAM		
	Cycles	Energy (J)	Energy*Delay	Cycles	Energy (J)	Energy*Delay (improvement)
Quadtree (8K,2-way)	5959406	0.0028102	16747.1	1111398	0.0020920	2325.0 (86.1%)
R-tree (8K,2-way)	4145641	0.0039902	16541.9	614581	0.0034671	2130.8 (87.1%)
Buddy-Tree (8K,2-way)	6218897	0.0032773	20381.2	1429412	0.0024918	3561.8 (82.5%)
Quadtree (16K,2-way)	1532206	0.0020735	3177.0	668678	0.0019456	1301.0 (59.0%)
R-tree (16K,2-way)	1252116	0.0034984	4380.4	325228	0.0033611	1093.1 (75.0%)
Buddy-Tree (16K,2-way)	1625547	0.0025138	4086.3	970077	0.0024064	2334.4 (42.9%)
Quadtree (32K,2-way)	1169681	0.0020068	2347.3	632425	0.0019272	1218.8 (48.1%)
R-tree (32K,2-way)	1012516	0.0034450	3488.1	301268	0.0033398	1006.2 (71.2%)
Buddy-Tree (32K,2-way)	1383872	0.0024659	3412.5	945909	0.0023939	2264.4 (33.6%)

Table 6: Impact of eDRAM on Nearest Neighbor Queries averaged over all datasets

We observe that the eDRAM significantly improves the cycles, energy and consequently the energy*delay of all indexing structures. More importantly, the effect is more pronounced for executions that incur more misses, and for this query it is the R-tree. As a result, the benefits of the R-tree are amplified in terms of energy*delay for the (8K,2-way) execution, where

the R-tree was already doing better without a eDRAM architecture. In the (16K,2-way) and (32K,2-way) executions, the eDRAM architecture makes the R-tree a better alternative from the energy*delay viewpoint (though it still consumes higher energy than Quadtree), even though it was worse than the Quadtree for the same metric without the eDRAM. Since we find that R-tree in general incurs more misses across the queries, eDRAM technologies are expected to benefit R-trees more than the other two structures.

5.8 Offloading Work to the Server

Throughout this paper, we have assumed that both the dataset and index are completely resident in memory and the entire task of query processing is performed locally at the mobile client. Sometimes, as pointed out earlier, a wireless network and a resource-rich server may be available, and in such cases it would be interesting to see whether it makes sense to offload the query to the server to save on energy and cycles. The mobile device (the client) communicates with the server via a wireless Network Interface Card (NIC). Previous studies have shown that the NIC is a significant power consumer [GHS96] which can offset benefits provided by a server. The trade-offs depend on factors such as the query-type, computing power of the client and the server, network bandwidth etc. In this section, we briefly motivate how work partitioning can provide both energy and performance benefits. A more detailed evaluation is conducted in [GAS⁺01].

Since it would require much more work to offset communication costs, we consider a larger dataset here, specifically we have extended PAFS to include the streets of Franklin, Bedford, and Huntingdon counties of Pennsylvania, making the dataset approximately 10.06 MB in size (client memory availability is increased to 32 MB and with 16 KB I-cache and 8 KB D-cache, both 4-way set associative). We show results with the packed R-tree that takes about 3.56 MB for this dataset. The server is assumed to be a 4-issue superscalar processor clocked at 1 GHz, with adequate memory to hold all of the dataset and index considered in its memory, and the client is clocked at one-eighth the rate of the server (i.e., 125 MHz) that is comparable to what is found today in commercial offerings such as the StrongARM SA-1110 [Int]. We consider network bandwidths of 2, 4, 6, 8, 11 Mbps (which is in the range of what is available or expected to be soon available in commercial offerings [ANS99]).

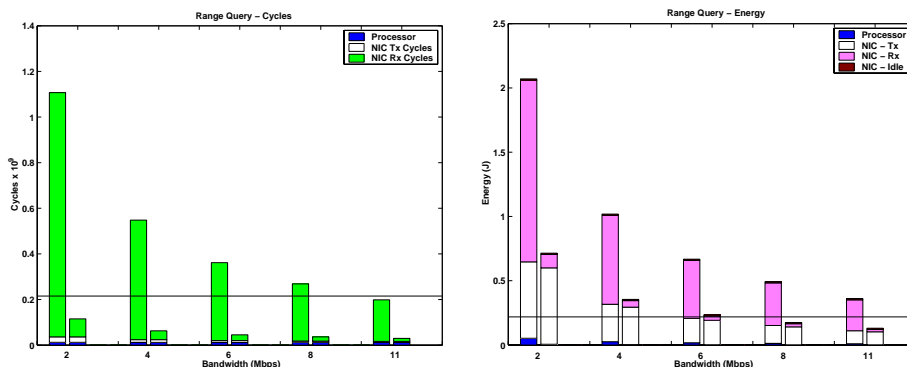


Figure 9: Range Queries. Comparing the option of performing the query at the client (shown as the horizontal line, vs. the option of performing the query completely at the server (shown as bars). The left bar for each bandwidth, are for the case where data objects are not available at the mobile client and need to be shipped from server, while the right bars are for the case where data objects are already available on the client. The profile for energy and cycles is given in terms of what the mobile client incurs in the NIC (given separately for transmission, reception and idle) and all other hardware components that are bunched together as processor.

There are several ways of partitioning the task of query-processing between a client and server. The query can be executed fully on the client, without any involvement from the server. This strategy has been the main focus of this paper. On the other hand, the query can be sent to the server, which in turn can execute it completely and send the results back to the client. Figure 9 presents the results comparing these two schemes. For each bandwidth, the bar on the left shows

the result when the data is not available on the client whereas the one on the right is for the case when the data is available locally as well. In the latter case, the server does not have to send back the data items that satisfy the query; instead, only the ids of the items need to be returned, thereby reducing the transfer-size of the message. Reducing the transfer-size reduces the amount of time for which the NIC needs to be active, thereby reducing the network transfer-time and also the energy consumption. It is interesting to note that the *gains in the performance and energy show different operating-points over the range of chosen bandwidths*. When the data is kept locally, work partitioning outperforms the fully-local case even at 2 Mbps, though it takes over 6 Mbps before it gets more energy-efficient. This happens because the energy cost of using the NIC is much more than the performance cost, and higher bandwidths are required to offset this difference.

There are other options for partitioning the work between the client and server, such as positioning the filtering at one end and the refinement at the other, and the trade-offs between those options are studied in [GAS⁺01].

6 Discussion

Despite the relatively small size of the datasets (to fit in the main memory or resource-constrained devices) we find that it is imperative to provide an index-based spatial access method to answer the three considered queries. Performance penalties of brute-force approaches are so significant (despite not incurring storage overheads needed to maintain index structures), and have a direct consequence on energy costs as well. If storage space overhead is a major concern for the resource-constrained environments, Quadtree is a better alternative than the other two structures (see Table 2). Of the other two, the packed R-tree makes better utilization of the space taken by its index nodes (see Table 2).

	Cycles	Energy	Energy*Delay
Point Query	1. BuddyTree 2. R-Tree,QuadTree	1. BuddyTree 2. R-Tree,QuadTree	1. BuddyTree 2. R-Tree,QuadTree
Range Query	1. BuddyTree,R-Tree 2. QuadTree	1. BuddyTree,R-Tree 2. QuadTree	1. BuddyTree,R-Tree 2. QuadTree
Nearest Neighbor	1. R-Tree 2. QuadTree 3. BuddyTree	1. QuadTree 2. BuddyTree 3. R-Tree	1. QuadTree 2. BuddyTree 3. R-Tree

Table 7: Comparison of Index Structures for different queries and criteria using the results of 2-way 16K cache configuration. (1) denotes the best and (3) denotes the worst for each entry in this table

Between the three index structures, we find no clear winner across all queries and criteria that have been studied. Table 7 summarizes some of the observations that have been made in the earlier sections. It ranks the schemes (from 1 to 3) based on their relative merits for the performance, energy, and energy-delay criteria, and a list of observations follow:

- For the point queries, we find the buddy tree giving better performance while incurring a lower energy cost. Consequently, it has the lowest energy*delay values of the three. Between the other two, the differences are not very prominent, especially if we can tune their fan-outs for this query.
- With range queries, both R-trees and Buddy trees are giving good performance, energy savings and energy*delay values. Quadtree is worse than these with queries needing to process more data for refinement.
- While performance largely dictates energy costs for the point and range queries, this study has shown that these criteria do not always go hand-in-hand. There could be circumstances when a scheme giving the performance can incur the highest energy cost. This was observed with the nearest neighbor query where R-tree was giving the best performance but incurs the highest energy. Quadtrees turn out to be better from the energy or energy*delay perspective for this query.
- Power dissipation is also another important consideration to keep the packaging and cooling costs low. We observed how filtering and refinement steps dissipate different amounts of power, with the nature of the query playing an

important role on which phase is more significant. While the energy and performance trends for the four datasets did not appear very different, on a closer examination we find that the dataset has as much an impact on power dissipation as the index structure itself.

- The results show that index-based query executions on spatial databases exercise the memory system considerably. A similar result has been noted recently in [ADHW99] where misses have been found to constitute around 40% of the execution time for memory-resident relational databases. The energy perspective shows that it is not only important to optimize miss behavior (by lowering number of misses, or by reducing energy consumption during misses), it is crucial to optimize energy consumption of hits as well (or even reduce the number of memory references). Energy consumption of caches plays an even more dominant role than its performance impact. While improving the caches (in terms of size and associativity) can reduce the miss behavior, the access costs increase. Consequently, we find that 16K 2-way associative caches are a good compromise between performance and energy for these workloads.
- To investigate the influence of evolving architectural enhancements and technological trends that can help reduce cache hit and cache miss energy consumption, we examined the impact of partitioning caches and embedding main memory on-chip. Partitioned caches for energy efficiency are likely to benefit Quadrees more than the others, since they tend to have better locality during filtering and refinement compared to R-trees or Buddy trees. On the other hand, embedding the main memory on-chip favors R-trees which in general has poorer locality than the other two.
- Off-loading the work to a resource-rich server using a wireless medium is a good option in certain demanding queries (range queries) when we can get reasonable communication bandwidths. Energy and performance, again have different implications on when you need to perform such off-loading. Energy savings requires a higher communication bandwidth than that for performance savings.

These observations can help a designer customize a SDBMS for a given target resource-constrained environment, fine-tune the implementation to dynamically adapt for changing energy and performance criteria, and to even provide guidelines on incorporating architectural enhancements that can help meet energy-performance criteria in a more effective manner.

7 Concluding Remarks

The growth in mobile computing has made mobile databases one of the most prominent segments of embedded database market [Ols00, Ort00]. The market for embedded databases is expected to grow about 12% annually to 705 million dollars in 2003. Many of these applications are targeted for automotive and handheld devices which are likely to hold, access and process spatial data. With the resource-constraints imposed on these embedded environments, energy and limited memory designs, take center-stage together with performance. This paper has presented the first indepth examination of memory-resident spatial access methods for three index structure (Quadrees, R-trees and Buddy Trees) from the energy, performance and energy-delay perspectives. By doing so, this paper has identified the key issues affecting both energy and performance, at the algorithmic and architectural levels. It has taught us several important lessons including the fact that optimizing performance does not necessarily optimize energy and could in fact aggravate power dissipation. Since the target environments may have different (storage) capacities, processing power, and resource-constraints, the results from this work will be helpful to select and tailor the spatial access methods for designing mobile applications operating in diverse conditions. This exploration can in turn provide insight on new problems for research on embedded and spatial databases, accelerating their deployment on numerous mobile devices.

8 Acknowledgements

This research has been supported in part by several NSF grants: 0073419, 0093082, 0093085, 0103583, 0097998, 9701475, 9988164, Sun Microsystems, Gigascale Silicon Research Center, and equipment grant EIA-9818327.

References

- [ADHW99] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs On a Modern Processor: Where Does Time Go? In *Proceedings of Very Large Databases Conference*, 1999.
- [AK93] R. Alonso and H. F. Korth. Database System Issues in Nomadic Computing. In *Proceedings of the ACM-SIGMOD Conference*, pages 388–392, 1993.
- [ANS99] ANSI/IEEE 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.
- [BA97] D. Burger and T. Austin. The simplescalar tool set, version 2.0. Technical report, Computer Sciences Department, University of Wisconsin, June 1997.
- [Ben75] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *CACM*, 18(9):509–517, September 1975.
- [BKSS90] N. Beckmann, H-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for points and Rectangles. In *Proceedings of the ACM-SIGMOD Conference*, pages 322–331, 1990.
- [CB95] A. Chandrakasan and R. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.
- [D⁺01a] V. Delaluz et al. DRAM energy management using software and hardware directed power mode control. In *Proceedings of the 7th International Conference on High Performance Computer Architecture*, 2001.
- [D⁺01b] V. Delaluz et al. Hardware and software techniques for controlling dram power modes. *IEEE Transactions on Computers*, 50(11):1154–1173, 2001.
- [D⁺01c] D. Duarte et al. Formulation and validation of an energy dissipation model for clock generation circuitry and distribution network. In *Proceedings of the International Conference on VLSI Design*, 2001.
- [FB74] R. A. Finkel and J. L. Bentley. Quad Trees - A Data Structure for Retrieval on Composite Keys. *Acta Informatica* 4, pages 1–9, 1974.
- [Fre87] M. Freeston. The bang file: a new kind of grid file. In *Proceedings of the 1987 ACM-SIGMOD Conference*, pages 260–269, May 1987.
- [G⁺95] A. Gonzales et al. A data cache with multiple caching strategies tuned for different types of locality. In *Proceedings of the International Conference on Supercomputing*, pages 338–347, July 1995.
- [GAS⁺01] S. Gurumurthi, N. An, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin. Energy and Performance Considerations in Work Partitioning for Mobile Spatial Queries. Technical Report CSE-01-028, The Pennsylvania State University, November 2001.
- [GEO] GEOPlace.Com. Mobile Technology Takes GIS to the Field. <http://www.geoplace.com/gw/2000/0600/0600IND.ASP>.
- [GG98] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–230, June 1998.
- [GHS96] P. Gauthier, D. Harada, and M. Stemm. Reducing Power Consumption for the Next Generation of PDAs: It's in the Network Interface. In *Proceedings of the International Workshop on Mobile Multimedia Communications (MoMuC)*, September 1996.
- [GK99] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers, and bit-line segmentation. In *Proceedings of the International Symposium Low Power Electronics and Design*, pages 70–75, 1999.

- [Gri86] J. G. Griffiths. An Algorithm for Displaying a Class of Space-filling Curves . *Software - Practice and Experience (SPE)*, 16(5):403–411, 1986.
- [Gun86] O. Gunther. The cell tree: an index for geometric data. Technical Report UCB/ERL M86/89, Univ. of California, Berkeley, December 1986.
- [Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM-SIGMOD Conference*, pages 47–57, 1984.
- [HN83] K. Hinrichs and J. Nievergelt. The grid file: a data structure to support proximity queries on spatial objects. In *Proceedings of the WG'83 Intern. Workshop on Graph Theoretic Concepts in Computer Science*, pages 100–113, 1983.
- [HS91] E. G. Hoel and H. Samet. Efficient Processing of Spatial Queries in Line Segment Databases. In *Proceedings of the 2nd Symposium on Advances in Spatial Databases(SSD)*, pages 237–256, 1991.
- [HS92] E. G. Hoel and H. Samet. A Qualitative Comparison Study of Data Structures for Large Line Segment Databases. In *Proceedings of the ACM SIGMOD*, pages 205–214, 1992.
- [IKVS00] M.J. Irwin, M. Kandemir, N. Vijaykrishnan, and A. Sivasubramaniam. A holistic approach to system level energy optimization. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization, and Simulation*, September 2000.
- [Ins] Environmental Systems Research Institute. Digital Chart of the World. <http://www.maproom.psu.edu/dcw/>.
- [Int] Intel StrongARM SA-1110 Microprocessor Brief Datasheet. <http://developer.intel.com/design/strong/datashts/278241.htm>.
- [IP95] Y. Ioannidis and V. Poosala. Histogram-Based Solutions to Diverse Database Estimation Problems. *IEEE Data Engineering*, 18:10–18, 1995.
- [ITM99] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.
- [IVB94] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy Efficient Indexing on Air. In *Proceedings of the ACM Conference on Management of Data (SIGMOD)*, pages 25–36, 1994.
- [KF93] I. Kamel and C. Faloutsos. On Packing R-trees. In *Proceedings of the ACM CIKM*, pages 490–499, Washington, DC, 1993.
- [KV⁺01] S. Kim, N. Vijaykrishnan, et al. Power-aware Partitioned Cache Architectures. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design(ISLPED'01)*, pages 64–67, August 2001.
- [LC98] T. Lehman and M. J. Carey. Query Processing in Main Memory Database Management Systems. In *Proceedings of the 1998 ACM-SIGMOD Conference*, pages 239–250, 1998.
- [LS98] D. B. Lomet and B. Salzberg. The hB-tree: A Multiattribute Indexing Method with good Guaranteed Performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1998.
- [Mic] Microsoft. Microsoft Pocket Streets. <http://www.microsoft.com/mobile/downloads/streets.asp>.
- [NS87] R. C. Nelson and H. Samet. A Population Analysis for Hierarchical Data Structures . In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 270–277, San Francisco, California, May 1987. ACM Press.
- [NST97] Y. Nunomura, T. Shimizu, and O. Tomisawa. M32RD-Integrating DRAM and Microprocessor. *IEEE Micro*, 17(6):40–48, 1997.
- [Ols00] M. A. Olson. Selecting and Implementing an Embedded Database System. *Computer*, 33(9):27–34, September 2000.

- [Ort00] S. Ortiz. Embedded Databases Come out of Hiding. *Computer*, 33(3):16–19, March 2000.
- [otC] U. S. Bureau of the Census. TIGER/Line(R) 1995 Data. <http://www.esri.com/data/online/tiger/index.html>.
- [Pat98] J. M. Patel. *Efficient Database Support for Spatial Applications*. PhD thesis, University of Wisconsin-Madison, 1998.
- [PM97] A. Papadopoulos and Y. Manolopoulos. Performance of Nearest Neighbor Queries in R-Trees. In *Proceedings of Intl. Conference on Database Theory*, pages 394–408, 1997.
- [PMD98] D. Papadias, N. Mamoulis, and V. Delis. Algorithms for Querying by Spatial Structure. In *Proceedings of Very Large Databases Conference*, pages 546–557, 1998.
- [R⁺95] N. Roussopoulos et al. Nearest Neighbor Queries. In *Proceedings of the ACM SIGMOD*, pages 71–79, 1995.
- [RJ96] K. Roy and M. C. Johnson. Software design for low power. *NATO Advanced Study Institute on Low Power Design in Deep Sub-Micron Electronics*, August 1996.
- [RL85] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. In *Proceedings of the ACM SIGMOD*, pages 17–31, 1985.
- [Rob81] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM-SIGMOD Conference*, pages 10–18, 1981.
- [RP00] K. Roy and S. C. Prasad. *Low-Power CMOS VLSI Circuit Design*. John Wiley & Sons, Inc, New York, 2000.
- [Sam89] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.
- [SC99] W-T. Shiue and C. Chakrabarti. Memory exploration for low power, embedded systems. Technical Report Technical Report CLPE-TR-9-1999-20, Arizona State University, 1999.
- [SCI⁺01] E. Shih, S-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks. In *Proceedings of the ACM SIGMOBILE Conference of Mobile Computing and Networking (MOBICOM 2001)*, July 2001.
- [SCR⁺99] S. Shekhar, S. Chawla, S. Ravada, et al. Spatial Databases - Accomplishments and Research Needs. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):45–55, 1999.
- [See91] B. Seeger. Performance Comparison of Segment Access Methods Implemented on Top of the Buddy-Tree. In *Proceedings of the Second International Symposium on Advances in Spatial Databases*, pages 227–296, 1991.
- [SK90] B. Seeger and H-P. Kriegel. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In *Proceedings of the VLDB*, pages 590–601, 1990.
- [Son97] P. Song. Embedded dram finds growing niche. In *Microprocessor Report*, pages 19–23, August, 1997.
- [SRF87] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the VLDB*, pages 507–518, 1987.
- [TS96] Y. Theodoridis and T.K. Sellis. A Model for the Prediction of R-tree Performance. In *Proceedings of the ACM SIGMOD*, pages 161–171, 1996.
- [V⁺00] R. Viswanath et al. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, Q3, 2000.
- [VKI⁺00] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. Kim, and W. Ye. An energy estimation framework with integrated hardware-software optimizations. In *Proceedings of the International Symposium on Computer Architecture*, 2000.