Vector Programming Abstraction for Sensor Networks

Tamim Sookoor and Kamin Whitehouse
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
{sookoor, whitehouse}@cs.virginia.edu

1. INTRODUCTION

Many application domains are adopting sensor networks as a viable computation platform with which to solve problems. A recent study indicates the global market for wireless sensor networks would grow tenfold by 2011 [3]. Currently sensor networks are being used for applications ranging from environmental monitoring [7, 17] and elderly monitoring [23] to seismic analysis of structures [1, 14] and military surveilance [2]. Despite being used in such a wide array of applications, sensor networks are notoriously difficult to program [16, 19, 22].

Sensor network applications can be classified as either data driven or event driven. Data driven applications involve periodically collecting data which is relayed to a base station while event driven applications involve identifying events. Applications such as habitat monitoring can be classified as being data driven while applications such as elderly monitoring are event driven. A number of sensor network applications, such as seismic analysis of structures, are inherently hybrids of the two as most of the time is spent collecting and routing data while being sensitive to events such as earthquakes. Many programming abstractions are most suitable for one or the other class of applications. For instance, TinyDB [15] is ideal for data driven applications as it enables querying the network, but is not as suitable for event driven applications which can greatly benifit from neighborhood level in-network collaboration. Other abstractions, such as Pleiades [13], trade runtime efficiency for expressiveness. We assert that this tradeoff is not fundamental and present vector programming as an abstraction which would provide expressiveness without sacrificing runtime efficiency.

The difficulty of programming sensor networks arises from the the distributed nature of the system, the lack of a sophisticated development environment with debugging support, the resource constraint on sensor nodes, and the unreliability of the communication channel between the nodes. In order to most efficiently utilize the limited resources and mitigate data loss, sensor networks are usually programmed at the device level. Programming at such a low level burdens the programmer with details such as message passing, power management, and routing. Also, such an abstraction forces the programmer to decompose the desired global functionality into node level actions which would give rise to such functionality. This makes coding, as well as debugging, tedious and error prone.

As mentioned previously, the use of sensor networks is expected to grow considerably in the next few years. Yet, such a growth would not occur if sensor networks are difficult to program. Most, if not all, sensor networks have been deployed by the researchers studying them. We conjecture that by making sensor network programming accessible to domain specialists, the expected growth in their use could be realized. For instance, an environmentalist should be able to deploy a sensor network to monitor the light under shrub thickets [18], instead of having to request sensor network researchers to deploy such a network. In order for sensor networks to move from the labs of sensor network researchers to the hands of application area specialists, many aspects of sensor network deployment have to be simplified. One of the most important aspects is programming, which continues to be difficult even for the sensor network experts.

Multiprocessor programming is a problem related to macroprogramming sensor networks. A considerable amount of research has been expended to enable programmers to write sequential code which could be automatically parallelized and executed on a multiprocessor system. Compilers such as the Vienna Fortran compiler [5], the Polaris compiler [6], and the SUIF [10] compiler were built to convert sequential code into multi-threaded code. Yet, no fully automatic compiler has been developed for anything but the most trivial programs. Programmer input, in the form of special keywords and annotations, is necessary to indicate parallelizable sections of code.

We think vector programming is a good abstraction for sensor networks due to a number of reasons. One reason is the availability of operations which apply transparently to vectors as they would to scalars. For instance, two vectors could be added together using the + operator which would result in corresponding elements of the two vectors being added. Such operators indicate parallelism. This would enable the programmer to indicate parallalism without resorting to special keywords or annotations. Also, the vectors, as we implement them, mimic the state of information in sensor networks making it easy for programmers to describe applications by manipulating this information.

The goals of our project are to make it easier to write programs for sensor networks which utilize their resources efficiently. Our method of programming should be expressive enough to enable most sensor network applications to be written easily. Two sets of applications are compared in order to evaluate the ease of programming and efficient resource utilization. We expect a vector abstraction to enable programs to be written with considerably fewer lines of code than using traditional abstractions. We believe shorter programs would be easier to write and debug. The efficient resource utilization is evaluated by simulating the code in Matlab. In terms of resources, we evaluate the energy utilization since energy efficiency has the greatest effect on the lifetime of a sensor network. This evaluation is carried out by counting the number of messages passed between devices during the simulation. Since communication is relatively expensive in terms of energy [11], the number of messages transmitted serves as a valid proxy for energy efficiency. Finally, the expressiveness of the application is demonstrated by writing a wide variety of programs encompassing all application area in which sensor networks have been used as well as areas where sensor networks would be used in the future.

We hypothesize that vector programming is a natural abstraction for sensor networks, enabling over 90% of sensor network applications to be written with at least 30% fewer lines of code than identical programs written without the abstraction. Furthermore, we expect a vector programming abstraction to make it easy for a compiler to incorporate many runtime optimizations in the output code resulting in at least 50% fewer messages passed when compared to identical applications written in Pleiades which we believe is the obvious alternative in terms of expressivness among the currently available sensor network macroprogramming systems. This would lead to an abstraction that makes programming easier and execution more efficient.

Vector programming is a familiar abstraction to many people since programs, such as Matlab and Mathematica, which rely heavily on vectors and arrays, are widely used in many fields. We expect this familiarity to make the abstraction easier to use. Also, the abstraction, which is based on a data structure called a distributed vector, mimics the flow of information in a sensor network. This would enable optimizations such as in-network aggregation and neighborhood-level data processing to take place on the data.

2. BACKGROUND AND RELATED WORK

A vector is a one-dimensional array. Vector programming is a programming model where scalar operations are generalized to apply transparently to vectors. For instance, two vectors can be added without explicitly iterating through their elements. The vector programming abstraction we propose involves a special type of vector called a distributed vector. A distributed vector is a vector who's elements are unsorted. The elements of a distributed vector are called distributed elements and are composed of an index and a value. A distributed vector can either be centralized or distributed. In its centralized state, all the distributed elements of a distributed vector reside on a single device. In its distributed state, each element resides on a separate device. The index of a distributed vector element is the ID of the device on which it resides in its distributed state.

As described previously, the aims of automatic parallelization are similar to the aims of macroprogramming for sensor networks. Automatic parallelization involves building compilers to automatically convert sequential code into multithreaded code to be executed on a multiprocessor machine. A number of compilers that covert sequential code to multithreaded code have been developed. Among them are the Vienna Fortran compiler [5], the Paradigm compiler [4], the Polaris compiler [6], and the SUIF compiler [10]. Titanium [25] is a dialect of Java which supports high-performance scientific computing on large-scale multiprocessors. The associated Titanium compiler optimizes the parallel programs written in the Titanium language. Yet these solutions are not very general and/or require a considerable amount of user input in order to parallelize anything but trivial applications. Also, most, if not all, of these compilers depend of special constructs such as $parfor$ and $cfor$ which indicate that the statements within that scope can be parallelized.

There have been many attempts at simplifying sensor network programming. These solutions have taken many forms, one of which is providing abstractions for low-level functionality of sensor networks. Directed diffusion [12] is a communication paradigm for sensor networks which utilizes reinforcement-based adaptation to select the best path for a message. Hood [20] is an abstraction for neighborhoods within sensor networks. It utilize mechanisms called mirrors to implement reflected variables among neighbors. Abstract regions [19] is an abstraction similar to Hood that enables devices within specific regions of a network to be addressed. Functions can be mapped onto data within these regions which can then be reduced.

A streaming database has been a popular abstraction for sensor networks. Semantic Streams [21], TinyDB [15], Cougar [24], and IrisNet [8] are all such database abstractions. Global system behavior is determined using queries. For instance, Semantic Streams uses declerative queries in Prolog while TinyDB uses SQL-like queries.

Recently, a number of imperative macroprogramming languages have been developed for sensor networks. Kairos [9] and Pleiades [13] are examples of such languages. Kairos is an extension to Python which provides support for iterating over nodes and accessing node-local state. Pleiades builds on Kairos and extends C instead of Python. This enables the code to be compiled down to nesC which can execute natively on Mica motes.

While the above solutions ease sensor network programming, they each have weaknesses. The abstractions for low-level functionality are not powerful enough on their own to describe the global behavior of a sensor network. For instance, Hood cannot be used, by itself, as a macroprogramming language, but can be a component of a macroprogramming language which describes global behavior. The database abstractions, while permitting the control of global behavior, are not sufficiently expressive for all sensor network applications. In fact, a large portion of sensor network applications do not easily conform to a database abstraction. Finally, while the imperative languages are highly expressive, their current implementations do not execute efficiently. Pleiades, for instance, requires a large number of messages to be passed between devices in order for the thread of control to migrate through the network.

There is a large body of work in the fields of distributed systems and high-performance computing which is relevant to the problem of macroprogramming sensor network. Many attempts have been made to solve this problem, yet an expressive language which executes efficiently has yet to be implemented. This project is another attempt at finding a solution to the problem.

3. APPROACH

In a vector programming abstraction, information in the sensor network is stored in variables, which are usually vectors. This information includes sensor readings, node-local information such as location and neighbor table, and constants. The global behavior of the network is specified by manipulating these variables in the context of a sequential program. Since a lot of information is stored as vectors, the features of a vector programming language allows applications to be written in fewer lines of code. Also, the vector programming abstraction allows the programmer to indicate parallel operations on vectors.

We extend Matlab, a widely used numerical computing environment which supports vector operations, by introducing a number of data types. The new data types we introduce are the distributed element, the distributed vector, the shared distributed element, and the shared distributed vector. These new data types permit the introduction of new control and data transfer paradigms which increase runtime efficiency. Also, having different data types enable multiple versions of common functions, such as min and sum, to be implemented which execute optimally depending on the data type. The Matlab interpreter selects the appropriate version of the function, depending on the data type it is applied to, automatically.

3.1 Requirements of a Macroprogramming Abstraction

In order to develop a macroprogramming abstraction that would be applicable to a large class of applications, the characteristics of these applications have to be analyzed. We analyze five applications which we think are representative of sensor network applications. The first is the Pursuer Evader Game (PEG) which involves a network of sensors attempting to localize a vehicle moving within the network. The second is a temperature control application for a smart house where, if motion is detected, the temperature of the house is adjusted to be within a particular range. The third is a battlefield management system where injured soldiers are identified and the closest medic alerted of thir position. The fourth is a parking space finder application where a network of sensors, which identify if parking spaces are free or reserved, are used by drivers to locate a free parkins space close to their destination. The final application we analyze is a contour detection application where nodes attempt to generate a contour enclosing an area of interest, for instance a fire. Table1 show the functions required by these applications, their traffic patterns, and requirements for data sharing.

In Table1, BS stands for base station. The traffic patterns indicate how data has to transfered within the network for a particular implementation of the application. The data sharing requirements fall into two classes: reliability and synchronicity.

3.1.1 Reliability

Reliability indicates the resiliance of the application to the loss of shared data. For instance if an application requires data to be reliably shared, the device which generates the data would resend the data until al of the receivers get the data. In sensor networks, which have lossy communication channels, this could result in a hugh overhead in terms of energy as message transmission requires a relatively large amount of energy. Many applications can tolerate some loss during data sharing and these applications can, therefore, utilize a best-effort sharing semantic, trading off some loss for energy conservation. For instance, in the temperature controller application, if a motion sensor reading is not received by the base station the worst that could happen is the temperature not being adjusted immediately. Most probably, if a person is moving around the house, motion would be detected again which would trigger temperature control. On the othehand, if a soldier is critically wounded in a battlefield, it is important that the information get to a medic. In such a situation best effort may not be sufficient.

3.1.2 Synchronicity

Synchronicity relates to the timing of shared data. If an application requires synchronous data sharing, all shared variables on a device must be assigned within a specific time frame. For instance, in the pursuer evader game, a node should have its neighbors magnetometer readings from approximately the same instant that it detected the evader in order for it to decide if it was a false positive. Yet, the data sharing between nodes and the base station could be asynchronous.

3.2 Programming abstraction

We abstract the sensor network as variables and functions. The variables store the information within the network such as sensor values, locations, and neighbor information. Functions are applied to these variables in order to describe applications. Functions can also be used to indicate actuation. We selected a vector programming language, since vector functions inherently indicate parallelism without the need for special keywords. For instance, $A + B$, where A and B were vectors, indicate that the elements in A and B could be added in parallel. If sequential exexution were required, the authors would have to sequentially iterate through the vectors using a for loop.

As mentioned above sensor network applications utilize require certain functions to be applied to data and utilize certain traffic patterns to move data through the network. Also, certain applications require specific constraints to be met when sharing data. Our vector programming abstraction provides the functions necessary for these applications and, since we belive they are representative of most sensor network applications, most applications which may be implemented using sensor networks. Also, the data structures we introduce: distributed elements and distributed vectors, enable the traffic patterns required as well as support reliability and synchronicity constraints for data sharing.

3.2.1 Functions

MATLAB, a vector programming language, provides a number of functions that can be applied to variables. If the variables the function is being applied to happen to be vectors, they are transparently applied to the elements of the vectors. Thus, two vectors can be added using + just as two integers would be added. The result of this function would be a vector with the same size as the input vectors having the sum of corresponding elements of the two vec-

Table 1: Characteristics of representative sensor network applications

Application	Required functions	Traffic patterns	Data sharing requirements
PEG	find(nodes), max(neighbors)	Node-to-BS, Neighborhood	Unreliable, a/synchronous
Temp ctrl	find(nodes), max(nodes), min(nodes)	Nodes-to-BS, BS-to-Nodes	Unreliable, asynchronous
MedEvac	find(nodes), subtraction, min(nodes)	Nodes-to-BS, BS-to-Nodes	Reliable, synchronous
Parking	find(nodes)	BS-to-Nodes, Nodes-to-BS, Neighborhood	Unreliable, synchronou
Contour	min(neighbors), max(neighbors)	Neighborhood	Unreliable, synchronous

tors. In addition, MATLAB provides a number of vector functions such as min, max, find, and any.

3.2.2 Distributed Element

Distributed elements are the elements which comprise distributed vectors. They have indices and values. Distributed elements can be thought of as variables with the namespace spanning multiple devices. Thus, they are usually used to store information which may be used by functions across device boundaries. For instance, a network of sensors collecting light values may store these values in distributed elements, since the application may require the maximum of all the light values which is an application of a function across device boundaries. The indices of distributed elements are usually the IDs of the device from which the data stored in value was obtained. This acts as the unique name for each distributed element across the multi-device namespace.

The distributed element version of most functions make use of in-network aggregation in order to minimize the number of messages passed. For instance, if we want to find the max of the light values that are stored in distributed elements, each node would send the max of its light value and all of its children's light values up the routing tree resulting in each node having to transmit only one message instead of a message for each of its children in addition to its own message.

3.2.3 Distributed Vector

Distributed vectors are collections of distributed elements. Distributed vectors reside on a single device. Unlike vectors in regular sequential programs, distributed vectors are unordered. This allows distributed vectors to be composed of distributed elements from a subset of the devices in the network. Yet, it prevents distributed vectors from being iterated through sequentially like regular vectors. Instead, elements in a distributed vector are indexed by the ID of the device from which the value in the element was obtained. Distributed vectors are implemented as hash tables, allowing the indexing to take place in $O(1)$ time.

4. EXPERIMENTAL DESIGN

The vector programming abstraction is implemented in Matlab by implementing the new data types as Matlab objects. A sensor network simulator is implemented in Matlab. This application simulates the transfer of control between nodes in the network as well as the sending and receiving of messages between nodes.

We do not modify the simulator and therefore the radio model does not change. Also, the size of the region over which the network is deployed remains unchanged.

We implement two sets of applications that are identical except for the fact that one uses the vector programming abstraction and the other does not.

The independent variables in our experiment is the number of nodes in the network and the application.

The dependent variable is the number of messages passed.

5. EXPERIMENTAL RESULTS

It is easier to program sensor networks using the vector programming abstraction than without the abstraction. Applications written using the abstraction execute more efficiently than applications written without the abstraction. The efficiency gained by the abstraction scales as the size of the network increases.

Table 2: Comparison of lines of code for applications written with and without the abstraction

Application	With abstraction	Without abstraction
PEG	10	25
Temperature control	10	20
Medevac	20	50
RFID	8	15
⋮	⋮	⋮

Table2 shows that applications written with the abstraction are 50% shorter on average than applications written without the abstraction. This indicates that it is easier to write applications with a vector programming abstraction than without such an abstraction.

¡A graph with the x-axis being the number of nodes and the y-axis being the number of messages passed. Ten pairs of lines, one for each application showing the number of messages passed as the number of nodes increases.¿

The graph above shows that applications implemented using the vector programming abstraction require fewer messages to be passed between devices than applications written without the abstraction. The graphs also indicate that the abstraction is scalable with the number of nodes as the applications written using the abstraction require fewer message passes than the applications written without the abstraction for all izes of the network from 100 nodes to 1000 nodes.

6. CONCLUSIONS

We demonstrate that a vector programming abstraction makes it easier to program sensor networks while retaining runtime efficiency. This abstraction would help move sensor networks from the labs of sensor network researchers into the hands of application area specialists such as environmentalists since many people are familiar with Matlab-like vector programming and thus would find it easy to program sensor network applications using the vector programming abstraction.

7. REFERENCES

[1] Center for information technology research in the interest of society. smart buildings admit their faults. `http://www.citris.berkeley.edu/applications/disaster_response/smartbuil%dings.html`, 2002.

[2] Ákos Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, and G. Simon. Countersniper system for urban warfare. *ACM Trans. Sen. Netw.*, 1(2):153–177, 2005.

[3] B. Bacheldor. Wireless sensor adoption expected to grow tenfold. *RFID Journal, Inc.*, September 2007.

[4] P. Banerjee, J. A. Chandy, M. Gupta, E. W. H. IV, J. G. Holm, A. Lain, D. J. Palermo, S. Ramaswamy, and E. Su. The paradigm compiler for distributed-memory multicomputers. *Computer*, 28(10):37–47, 1995.

[5] S. Benkner. Vfc&colon; the vienna fortran compiler. *Sci. Program.*, 7(1):67–81, Jan. 1999.

[6] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu. Parallel programming with polaris. *Computer*, 29(12):78–82, 1996.

[7] A. Cerpa, J. Elson, M. Hamilton, J. Zhao, D. Estrin, and L. Girod. Habitat monitoring: Application driver for wireless communications technology. In *SIGCOMM LA '01: Workshop on Data communication in Latin America and the Caribbean*, pages 20–41, San Jose, Costa Rica, Apr 2001. ACM Press.

[8] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a worldwide sensor web. 02(4):22–33, October-December 2003.

[9] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2005.

[10] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion, and M. S. Lam. Maximizing multiprocessor performance with the suif compiler. *Computer*, 29(12):84–89, 1996.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, Nov. 2000.

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, New York, NY, USA, 2000. ACM Press.

[13] N. Kothari, R. Gummadi, T. D. Millstein, and R. Govindan. Reliable and efficient programming abstractions for wireless sensor networks. In J. Ferrante and K. S. McKinley, editors, *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 200–210. ACM, 2007.

[14] V. A. Kottapalli, A. S. Kiremidjian, J. P. Lynch, E. Carryer, T. W. Kenny, K. H. Law, and Y. Lei. Two-tiered wireless sensor network architecture for structural health monitoring. In *Proc. the SPIE 10th Annual International Symposium on Smart Structures and Materials*, San Diego, CA, March 2000.

[15] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, Mar. 2005.

[16] G. Mainland, L. Kang, S. Lahaie, D. C. Parkes, and M. Welsh. Using virtual markets to program global behavior in sensor networks. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 1, Leuven, Belgium, September 2004. ACM Press.

[17] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, Atlanta, GA, USA, September 2002. ACM Press.

[18] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. Luster: Wireless sensor network for environmental research. In *The 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Sydney, Australia, Nov. 2007.

[19] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 3–3, San Francisco, California, March 2004. USENIX Association.

[20] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110, New York, NY, USA, June 2004. ACM Press.

[21] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette: using rpc for interactive development and debugging of wireless embedded networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 416–423, New York, NY, USA, April 2006. ACM Press.

[22] A. Woo, S. Seth, T. Olson, J. Liu, and F. Zhao. A spreadsheet approach to programming and managing sensor networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 424–431, Nashville, Tennessee, USA, April 2006. ACM Press.

[23] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic. Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. Technical report, University of Virginia, 2006. `http://www.cs.virginia.edu/~adw5p/pubs/tr06-alarmnet.pdf`.

[24] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, Sep. 2002.

[25] K. A. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. N. Hilfinger, S. L. Graham, D. Gay, P. Colella, and A. Aiken. Titanium:

A high-performance java dialect. *Concurrency: Practice and Experience*, 10(11-13), September-November 1998.