Energy Efficient Network Memory for Ubiquitous Devices

Joshua B. Fryman, Chad M. Huneycutt, Hsien-Hsin S. Lee
Kenneth M. Mackenzie, David E. Schimmel
Center for Experimental Research in Computer Systems (CERCS)
Georgia Institute of Technology
Atlanta, GA 30332-0280
{ fryman, chadh, leehs, kenmac, schimmel }@cercs.gatech.edu

Abstract

This paper explores the energy and delay issues that occur when some or all of the local storage is moved out of the embedded device, and into a remote network server. We demonstrate using the network to access remote storage in lieu of local DRAM results in significant power savings. Mobile applications continually demand additional memory, with traditional designs increasing DRAM to address this problem. Modern devices also incorporate low-power network links to support connected ubiquitous environments. Engineers then attempt to minimize utilization of the network due to its perceived large power consumption. This perception is misleading. For 1KB application "pages," network memory is more power efficient than one 2MB DRAM part when the mean time between page transfers exceeds 0.69s. During each transfer the application delay to the user is only 16ms.

1. INTRODUCTION

This paper explores the energy and delay issues that occur when some or all of the local storage is moved out of the embedded device, and into a remote network server. Contrary to designer intuition, we demonstrate that this is more power efficient than local storage.

Embedded consumer systems continue to add more features while shrinking their physical device size. Current 2.5/3G cell phones incorporate 144kbps or better network links [5], offering customers not only phone services but also e-mail, web surfing, digital camera features, and video on demand. With feature expansion demanding additional storage and memory in all computing devices, densities of DRAM and Flash are increasing in an attempt to keep pace. This continuous storage expansion translates to a growing power dissipation and battery drain.

To reduce energy effects and increase battery life, designers use the smallest parts and lowest part count possible. This has the added benefit of keeping manufacturing cost down. This effort at minimizing available resources works against application feature expansion as well as the need for device flexibility for dynamic upgrades.

In an attempt to address some of these problems, companies like NTT Japan are investing time and research effort in solutions that allow for Mobile Computing – dynamically migrating application code between the remote device and other network connected systems [16]. Research efforts in academia are investigating alternate designs [28].

Other examples of ubiquitous network computing environments in the wild can be seen in the ZebraNet project [17] and similar animal or habitat monitoring systems [20, 8], houses with computer awareness integrated [23], urban traffic or location monitoring cameras, or similar widespread devices [26].

One avenue for power savings has not been fully considered, however. Many embedded devices, and all mobile devices, have a network link (GSM, Bluetooth, Ethernet, etc.) into a larger distributed environment. After designers incorporate sufficient power to support a network link, they then attempt to minimize use of the link due to its excessive energy needs during activity. Products therefore incorporate all the needed local storage in the device, buffering as much as possible to avoid retransmission. This ignores the fact that the remote server has an unrestricted power budget, and can be made arbitrarily fast to handle requests quickly.

For ubiquitous always-on devices like 3G cell phones, there is the potential to use the network link as a means for accessing applications remotely. This could reduce local storage space, thereby reducing energy demands on the mobile platform. This remote memory could lie in a remote server, or simply be cached within the network infrastructure.

Utilizing the network link to access remote memory provides a more energy efficient solution than traditional local memory. Traditional designs assume that the additional cost of utilizing the network link for moving code and data will far outweigh any benefit of removing or reducing local storage. As we will demonstrate, the best low-power mobile DRAM available today is 100 times less expensive to access in terms of energy per bit than a very low-power Bluetooth network. However, the sleep-mode current of the same Bluetooth network module is 10 times less expensive than the same DRAM part. Given sufficient time between accesses, the network link will always be more power efficient than local DRAM. As network links continue to increase in speed, this required time between accesses continues to fall.

The rest of this paper is organized as follows: Section 2 explores the three fundamental embedded platform models that we analyze for energy and delay impacts. In section 3 we present the analytical results for each model. Section 4 provides a brief overview of some related work, and section 5 presents our conclusions.

2. DEVICE MODELS

To investigate the possible performance effect of using a network as a mechanism for accessing remote storage, different device models and characteristics must be considered. There are three fundamental models of embedded computing devices that we examine: Legacy, Pull, and Push. Each model is characterized by the type of network link and communications model incorporated. We assume that the applications exhibit sufficient locality such that there are well-defined "working sets" that change infrequently [1, 4].

Each model is considered independently. While general comparisons can be made across models, each has different design-time

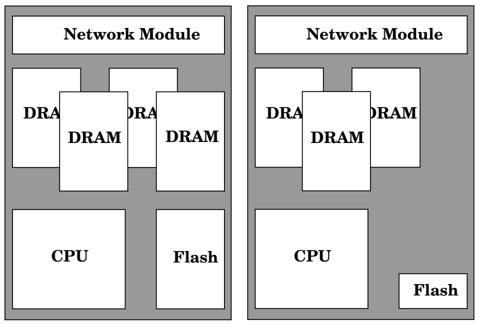

Figure 1: **Basic 3G cell phone or other ubiquitous networked device. On the left is part (A) a typical mobile embedded device. Part (B) on the right shows the small reduction proposed in this work.**

Variable	Meaning
P	Power (Watts)
T	Time (seconds)
E	Energy (Joules)

Subscript	Meaning
TX	Network transmit
S	Remote server processing
RX	Network receive
Nslp	Network sleep state
N	Network total
L	Local storage access
Lslp	Local storage sleep state
Idle	CPU Idle
Busy	CPU Busy
C	Application computing

Table 1: **Variables and subscript terms in model equations.**

characteristics which make direct comparison difficult. The underlying hardware design behind each model is the same, however, and is shown in Figure 1A.

The basic operation of such a device is that the necessary program and data values are copied from Flash to local DRAM for performance issues. The Flash components draw little current when not in use, and as well could be V_{DD} gated to draw zero current when idle. This copying requires a large amount of DRAM for holding all or part of the Flash contents.

We propose that by utilizing the network link to access the equivalent contents of Flash from a remote server, a more energy efficient model is constructed at a lower cost. This is achieved by reducing the Flash component to just a boot-block sized unit, and removing some part of DRAM from the local storage. The DRAM removed would normally contain the contents of Flash copied on boot-up or during application mode change.

Instead, we propose only a space large enough to hold the working set is reserved in the local DRAM, along with sufficient space for all local data. This concept for reduction is shown in Figure 1B.

In the following sections, we introduce each of the three embedded models and the notation we use to analyze the energy and delay issues inherent in each. Full analysis of each equation will be discussed in Section 3. Table 1 is provided for a quick reference on the primary variable types introduced below, as well as the subscripts used.

2.1 Legacy

The "legacy" device was originally conceived and constructed without expectation for ever needing communication. We examine the issues of energy and delay if a network link is added and local storage is reduced. The legacy application remains unchanged, but the code and data now come from network memory.

The original design expected a certain amount of normal energy consumption. To see how adding a network link impacts this, we model the extra energy incurred by not only using the network link to fetch new code and data, but also the energy consumed by the network link when in a sleep state. We assume the network link is only used for fetching new code and data, and that the legacy application itself is not attempting to communicate to other devices.

In order to "request" new code or data, a message must be generated and sent to the remote server. This transmission time, T_{TX}, will consume power P_{TX} as determined by the type of network link. Once the request is received at the remote server, there will be some interval of time processing the request, T_S, during which there will be additional power consumption on the local device monitoring the network, P_S. Once processed, the server will reply with the necessary information, which takes time to receive, T_{RX}, consuming power P_{RX}. The "payload" of the transmission will consist of some number of bits, B_N. In comparison, local storage only incurs a very minor time to access, T_L, with a correspondingly small power use, P_L. Whether transferred by network or from local storage, the CPU will be idle during these transfers, consuming some power P_{Idle}.

Regardless of which method is used – network or local storage – after transferring the payload B_N, some amount of time is spent in computation, T_C, before the next request is generated. During this time, the CPU will consume power while busy, P_{Busy}, and the backing store can be put into a powered-down state or sleep mode. Thus, during the work period T_C, the network link will consume sleep power P_{Nslp}, and the local storage will consume sleep power, P_{Lslp}.

The total energy consumed by the network link in the legacy model is

$$\begin{aligned} E_{TX} &= T_{TX} \cdot (P_{TX} + P_{Idle}) & (1)\\ E_S &= T_S \cdot (P_{RX} + P_{Idle}) & (2)\\ E_{RX} &= T_{RX} \cdot (P_{RX} + P_{Idle}) & (3)\\ E_C &= T_C \cdot (P_{Nslp} + P_{Busy}) & (4)\\ E_N &= E_{TX} + E_S + E_{RX} + E_C & (5) \end{aligned}$$

and the total energy in the local storage model is

$$E_L = T_L \cdot (P_L + P_{Idle}) + T_C \cdot (P_{Lslp} + P_{Busy}) \quad (6)$$

In terms of energy, the network model is equivalent to the local storage model when $E_N = E_L$, but to consider the delay impact on application performance, we construct the energy-delay product:

$$E_N \cdot (T_{TX} + T_S + T_{RX} + T_C) = E_L \cdot (T_L + T_C) \quad (7)$$

Solving this equation for T_C provides the energy-delay equilibrium point where using a network backing store is equivalent to using local storage. When T_C is greater than this equilibrium value, the network link is more efficient from a total system perspective.

2.2 Pull

Unlike the isolated model, the pull model assumes a network link has been incorporated in the embedded device since creation. The characterization "pull" comes from how the network is used: the local device, only at its own direction, pulls information from the network. External network devices can not arbitrarily send information to a device operating in a "pull" mode.

Using the same notation as the legacy model, however, there are only minor differences in the energy analysis. In the pull model, the original design engineers already budgeted power for a network link. The link, however, was expected to be in a power-down sleep state during normal operation except when the program requested remote activity. We only need to calculate the impact of new behavior (additional traffic) over the original expected behavior (sleep state). Therefore, we need to account for the *difference* between the network link being in sleep state as opposed to actively sending and receiving messages.

The result is that the original equations 1 through 4 are replaced with

$$\begin{aligned} E_{TX} &= T_{TX} \cdot (P_{TX} + P_{Idle} - P_{Nslp}) && (8)\\ E_S &= T_S \cdot (P_{RX} + P_{Idle} - P_{Nslp}) && (9)\\ E_{RX} &= T_{RX} \cdot (P_{RX} + P_{Idle} - P_{Nslp}) && (10)\\ E_C &= T_C \cdot P_{Busy} && (11) \end{aligned}$$

The equations for local energy 6 and the energy- delay product 7 are otherwise the same.

2.3 Push

Similar to the pull model, the push model also assumes a network link was built-in originally. The difference between the pull and push model lies in how the network link is used. With the pull model, only the local device could initiate connections to other network services. In the push model, the network link is always "on" so that if not actively transmitting, it is in receive-listen mode. Thus external network services can immediately "push" information to the local device, such as email notices, software upgrades, etc.

As the pull model reduced the energy drain to store information in the network compared to the legacy model, the push model reduces the drain further. Since the device was designed assuming an always active receive mode network, the original design expected to have a large power dissipation for this purpose. Therefore, we subtract the power term for normal receive-mode network links, P_{RX}, rather than the smaller power term for a sleep-mode link as in the pull model. That is, we again replace equations 1 through 4 by

$$\begin{aligned} E_{TX} &= T_{TX} \cdot (P_{TX} + P_{Idle} - P_{RX}) && (12)\\ E_S &= T_S \cdot (P_{RX} + P_{Idle} - P_{RX}) && (13)\\ E_{RX} &= T_{RX} \cdot P_{Idle} && (14)\\ E_C &= T_C \cdot P_{Busy} && (15) \end{aligned}$$

As with the pull model, the local energy and energy-delay product equations remain the same.

3. ANALYSIS

We now analyze in detail both the energy equilibrium point as well as the energy-delay product for each of the three modes discussed in Section 2. In order to have a quantitative analysis, we obtained technical data on various low-power current market products for both DRAM and Flash memory.

Using data sheets available from vendors including Elpida, Fujitsu, Intel, Micron, NEC, and Samsung, we selected low-power or "mobile" parts to represent typical market performance. We calculate the energy consumption in terms of pJ per bit by computing the *best*-case power consumption listed in the electrical characteristics of each product. This gives us a relative measure of how much energy is used in a *best*-case situation to read or write to the local storage device. During sleep mode, these devices consume very low current but still require some power for refresh functions. These calculations are shown for DRAM in Table 2, and Flash in Table 3.

Similarly, we calculate energy information from the data sheets published by several network link vendors. In a similar manner as for the DRAM, we determine the *worst*-case power per bit consumed, and the standby or sleep-mode power. In this situation, the transmit (TX) and receive (RX) currents are considered separately, as some links display different needs by operating state. We restricted our search to monolithic, fully-integrated network modules to ensure valid power measurements. Using multiple chip solutions requires external components and glue logic which make power calculation difficult if not impossible. The components we considered and their power calculations are shown in Table 4.

For our analysis, we demonstrate a conservative extreme: *best*-case local storage vs. *worst*-case network links for remote storage. While neither of these models is generally realistic, they demonstrate the *extreme* bounds where network links are more effective than local storage. Thus in actual application, network links should be more efficient than we demonstrate here.

3.1 Simplifications

To reduce the complexity of analysis, we make several simplifying assumptions:

First, we assume that the local device will not transfer user data (dirty data) into the network for storage, but will keep it all locally. The bulk of transfers we consider to be code, unalterable items such a video streams, or one-time user-centric data such as email. While email could be buffered on a remote server and not stored locally, it still represents this model in that reading email is no different than receiving a new program to run.

Second, we assume that the "request" for code or data to a remote server can be encapsulated in a 64-byte packet. This could be reduced or expanded based on the network topology and error handling needs, but has sufficient storage space for simple requests.

Third, we assume that the "response" packet being a variable-payload version of the "request" packet will consist of 20 bytes for control information followed by the actual payload of size B_N bits.

Fourth, we simplify what "local storage" means. Since Flash is substantially slower than DRAM, we assume that the application is copied from Flash to DRAM for faster execution, and then Flash is placed in deep-sleep mode. Therefore, we *ignore* the contribution of Flash to the total energy for the *best*-case memory analysis.

Finally, we assume that for the total count of DRAM chips, at least one is for mirroring Flash. However, based on the working set principle [6], only a small fraction of this space is actually needed at any given moment. Rather than store an entire mirror image, only sufficient space for the working set should be reserved in local DRAM, with excess DRAM then removed. By using the network link to access applications, we can also shrink the Flash such that it contains only a boot image, and not all applications that could ever be run. Since steady-state mode changes occur relatively infrequently [1, 4], the need to load new code and data from the network will also occur infrequently.

Therefore, our analysis uses the assumption that *one* DRAM chip is removed, although it could include reducing Flash as well. For the local storage comparison, we ignore the effects of initiating and waiting for memory access, and assume all accesses begin instantaneously at the maximum supported rate of the DRAM device. Moreover, we assume transition from idle- or sleep-mode to active- mode is also instantaneous. We assume minimal V_{DD} and current consumption at all times, while ignoring refresh operations. This constitutes a *best*-case memory model.

The *worst*-case network model uses typical V_{DD} with worst-case current consumption in all cases. With slower transfer rates, higher current consumption, and a long duration of remote server processing T_S, the network appears unattractive for energy savings at first glance. We will now demonstrate that this is not the case.

For analysis arguments, we use for the DRAM device a Fujitsu FCRAM model MB82D01171A, a 2MB part with the lowest power

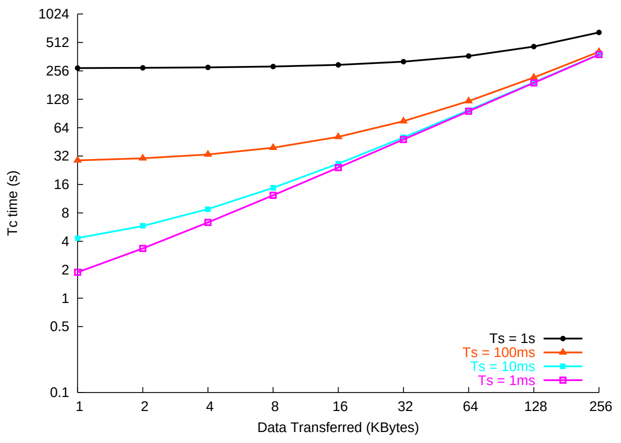

Figure 2: LEGACY: Duration of computation T_C required for the network link to more energy efficient, where T_S and B_N vary.

consumption of all devices measured in pJ/bit. Our network link model is the CSR BC2-Ea, a fully integrated Bluetooth module. Our CPU model is the DEC SA-110, a 0.5W processor during high computation and 0.02W during idle periods [22].

3.2 Legacy

The legacy model presents the worst energy-delay product result. We have added a network link to a design that did not expect it. For the network link to be more efficient than local DRAM, it requires a significant amount of time spent in computation, T_C. Figure 2 illustrates the energy-delay solution of Equation 7 for T_C such that the network link is as effective as the local DRAM, using different values for server processing time T_S. Times beyond this T_C are a "win" for using remote storage instead of local storage.

As the figure illustrates, when T_S is large it dominates the energy-delay product. Given sufficiently large sizes to transfer, however, the network becomes the limiting factor in the result. This presents the strong argument that optimization of the remote server for rapid response times will have a more significant impact on the energy efficiency of the local device than any other network characteristic.

3.3 Pull

The pull model provides better energy-delay results than the isolated model, as can be expected from subtracting the sleep mode power. However, the improvement turns out to be small compared to the energy costs associated with transferring the data as well as the remote server processing time T_S. Figure 3 shows a very similar energy-delay solution to that of Figure 2.

The difference between legacy and pull models is approximately 8%. While this is an improvement, the dominating factor in the energy-delay product is not the sleep mode energy consumed.

This analysis does indicate, however, that adding a network link to a legacy system when using a pull-based communications model will have a small impact when compared to the energy consumed by local storage devices.

3.4 Push

The push model uses the least additional energy and thereby benefits most from using remote storage. Since the network link is already expected to be in a receive-mode state at all times, the only extra energy used to access remote storage is the energy of the transmit operations. Figure 4 demonstrates an energy-delay savings that is substantial compared to a pull model.

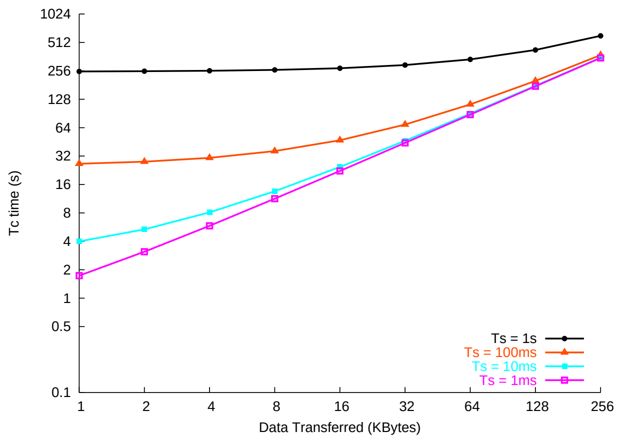

Figure 3: PULL: Duration of computation T_C that must pass for the network link to be more energy efficient, where T_S and B_N vary.

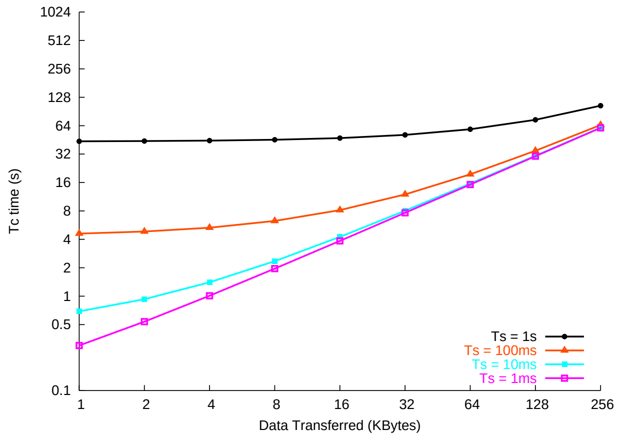

Figure 4: PUSH: Duration of computation T_C that must pass for the network link to more energy efficient, where T_S and B_N vary.

The energy-delay benefit for a 1KB "page" change with T_S of 10ms in the legacy model required 4.3s as a minimum change interval. With the push model, this time is reduced to 0.69s. In relative comparison, this same 1KB "page" of code loaded across the network with $T_S = 10ms$ will present an application delay of 0.016s to the user while accessing the network.

A larger "page" size to transfer may be more realistic to consider, however. For a 32KB change with $T_S = 10ms$, the legacy model requires 26.6s. The push model reduces this time to 4.24s. The delay the user experiences while the network transfer occurs is 0.185s.

4. RELATED WORK

Utilization of the network as a mechanism for accessing backing store as a low-power mechanism is novel. Prior work concentrated on using remote memories for high-performance reasons, avoiding accesses to slow disks or to expand memory for working sets of code or data [7, 25, 10, 21, 15, 14, 11]. Other work examining the network in power-limited devices has concentrated and minimizing the usage [12] and optimizing protocols.

A significant amount of effort is being spent to find ways of improving the overall energy efficiency of networks. WLANs can improve their efficiency by using ad-hoc relaying [19]. Other efforts have looked at tying battery level with ad-hoc routing methods to increase network robustness as well as node run-times [18], or trying to characterize the energy usage in wireless devices [9].

Using the availability of low-power short-range devices such as Bluetooth, others are building larger networks in an energy-efficient manner. These new systems compete with other, more traditional, network options [2]. Such efforts strengthen the viability of using otherwise limited hardware.

Other efforts are focusing on expanding CPU on-die storage with addressable memory [24, 3, 27] to reduce energy consumption and improve performance. One group has proposed to reduce or eliminate local storage by using a client-server network storage model [13].

With each generation of network technology, data rates increase and power consumption decreases (see Table 4 for examples). Next-generation technology such as Ultra-Wideband is anticipated to be higher data rate and lower power than current Bluetooth devices, with improved range. As network links approach the performance characteristics in bandwidth and power of local DRAM, the argument for moving to network-based storage becomes more evident.

5. CONCLUSION

Typical designs for embedded platforms include multiple low-power local storage parts such as DRAM and Flash. Many contemporary devices also include a network link of some type. Design efforts focus on minimizing the energy consumption of the network by adding sufficient local storage to hold all possible programs and data that users need. We have shown that the underlying assumption is misleading, and that using remote servers to store information can be more energy-efficient than using local storage. Accounting for processing times, protocol overheads, and using *best*-case local storage behavior versus *worst*-case network link behavior, we demonstrated that networks are more efficient when transfer times exceed a small threshold.

For a reasonable, average working set of 32KB transferred via the network, the link is more efficient than local DRAM if transfers occur no more frequently than every 4.24s, using a push model. This assumes the network link replaces *one* local 2MB DRAM chip. Removal of larger or multiple components makes network usage even more efficient.

Today, network links use 100 times the energy of DRAM during accesses, but consume 10 times less energy during sleep. Network devices continue to approach low-power DRAM performance characteristics in terms of speed and power, making network memory increasingly attractive. The reduction in part count, and thus price, can aid in the marketing of disposable devices such as cell phones. This model where applications are downloaded on demand also provides a mechanism for pay-per-use services, such as custom games or video players. We are currently working on detailed simulations to study the impact of this network memory model on overall network congestion.

6. ACKNOWLEDGEMENTS

This work was funded in part by the National Science Foundation under grants CCR-98-76180, CCR-01-21638, and EIA-99-72872.

7. REFERENCES

[1] Gheith A. Abandah and Edward S. Davidson. Confi guration Independent Analysis for Characterizing Shared-Memory Applications. Technical report, EECS Department, Univerisity of Michigan, CSE-TR-357-98 1998.

[2] Simon Baatz, Christoph Bieschke, Matthias Frank, Karmen Kühl, Peter Martini, and Christoph Scholz. Building Effi cient Bluetooth Scatternet Topologies from 1-Factors. In *Proceedings of the IASTED International Conference on Wireless and Optical Communications*, July 2002.

[3] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, and Peter Marwedel. Scratchpad Memory: A Design Alternative for Cache On-chip memory in Embedded Systems. In *Proceedings of the 10th International Workshop on Hardware/Software Codesign*, May 2002.

[4] Ravi Batchu, Saul Levy, and Miles Murdocca. A Study of Program Behavior to Establish Temporal Locality at the Function Level. Technical report, Rutgers University, DCS TR-475 2001.

[5] J. Blecher. Cell Phone Carrier Technology Chart. CNet Wireless Watch (`http://www.cnet.com/wireless`), September 2001.

[6] Peter J. Denning. Virtual Memory. In *Computing Surveys*, volume 2, No. 3, pages 153–189, September 1970.

[7] Sandhya Dwarkadas, Nikolaos Hardavellas, Leonidas Kontothanassis, Rishiyur Nikhil, and Robert Stets. Cashmere-VLM: Remote Memory Paging for Software Distributed Shared Memory. In *Proceedings of the International Parallel Processing Symposium and the Symposium on Parallel and Distributed Processing*, pages 153–159, 1999.

[8] Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the Physical World with Pervasive Networks. In *Pervasive Computing*, Jan 2002.

[9] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *IEEE INFOCOM*, 2001.

[10] Michail D. Flouris and Evangelos P. Markatos. The Network RamDisk: Using Remote Memory on Heterogeneous NOWs. In *Cluster Computing*, volume 2, pages 281–293, 1999.

[11] Kieran Hart and David R. Cheriton. Application-Controlled Physical Memory using External Page-Cache Management. In *ASPLOS*, pages 187–197, 1992.

[12] Paul J.M. Havinga and Gerard J.M. Smit. *Energy-effi cient Wireless Networking for Multimedia Applications*. 2000.

[13] Chad M. Huneycutt, Joshua B. Fryman, and Kenneth M. Mackenzie. Software Caching using Dynamic Binary Rewriting for Embedded Devices. In *International Conference on Parallel Processing*, 2002.

[14] Liviu Iftode, Kai Li, and Karin Petersen. Memory Servers for Multicomputers. In *Proceedings of the IEEE International Computer Conference*, pages 543–547, 1993.

[15] Sotiris Ioannidis, Evangelos P. Markatos, and Julia Sevaslidou. On Using Network Memory to Improve the Performance of Transaction-Based Systems. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, 1998.

[16] NTT Japan. BLUEBIRD Project. 2003. `http://www.ntts.co.jp/java/bluegrid/en/`.

[17] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein. Energy-Effi cient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiments with ZebraNet. In *Proceesings of ASPLOS-X*, October 2002.

[18] D. Kim, J.J. Garcia-Luna-Aceves, K. Obraczka, J. Cano, and P. Manzoni. Power-Aware Routing Based on The Energy Drain Rate for Mobile Ad Hoc Networks. In *Proceedings of the IEEE International Conference on Computer Communication and Networks*, October 2002.

[19] M. Kubisch, S. Mengesha, D. Hollos, H. Karl, and A. Wolisz. Applying ad-hoc relaying to improve capacity, energy effi ciency, and immission in infrastructure-based WLANs. Technical report, Technical University Berlin, July 2002.

[20] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[21] Evangelos P. Markatos and George Dramitinos. Implementation of a Reliable Remote Memory Pager. In *USENIX*, pages 177–190, 1996.

[22] James Montanaro and et al. A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor. In *IEEE Journal of Solid-State Circuits*, volume 31, No. 11, pages 1703–1714, November 1996.

[23] Georgia Institute of Technology. The Aware Home Project. 1999-2003. `http://www.cc.gatech.edu/fce/ahri/-publications/index.html`.

[24] Preeti Ranjan Panda and Nikil D. Dutt. Memory Architectures for

Embedded Systems-On-Chip. In *Proceedings of High Performance Computing*, December 2002.

[25] Dionisios Pnevmatikatos and Evangelos P. Markatos. On Using Network RAM as a non-volatile Buffer. In *Cluster Computing*, volume 4, pages 295–303, 1999.

[26] Sameer Tilak, Nael Abu-Ghazaleh, and Wendi Heinzelman. A Taxonomy of Wireless Micro-Sensor Network Models. In *Mobile Computing and Communication Review*, April 2002.

[27] Manish Verma, Stefan Steinke, and Peter Marwedel. Data Partitioning for Maximal Scratchpad Usage. In *Proceedings of Asia South Pacifi c Design Automated Conference*, January 2003.

[28] Dong Zhou, Santosh Pande, and Karsten Schwan. Method Partitioning - Runtime Customization of Pervasive Programs without Design-time Application Knowledge. In *International Conference on Distributed Computing Systems*, May 2003.

Vendor	Model	MB	width	MHz	V_{DD}	access mA	sleep mA	pJ/bit	pJ/bit/MB
Elpida	EDL1216AASA	16	16	133	2.3-2.7	80	1.5	86.5	5.4
Fujitsu	MB82D01171A-80	2	16	125	2.3-2.7	20	0.2	23.0	11.5
Micron	MT48V4M32-10	16	32	100	2.3-2.7	100	0.35	71.9	4.5
Micron	MT48V16M16-10	32	16	100	2.3-2.7	80	0.35	115.0	3.6
NEC	μPD4664312	8	16	150	2.7-3.3	45	0.1	49.6	6.2
Samsung	K4S643233-75	8	32	100	2.3-2.7	85	5	61.1	7.6
Samsung	K4S283233-75	16	32	100	2.7-3.6	220	6	185.6	11.6
Samsung	K4S561633-1H	32	16	100	2.7-3.6	130	6	219.4	6.9

Table 2: **Mobile DRAM characteristics: size, bit-width, speed, voltage, best-case access current, best-case sleep-mode, current use, pJ per bit in accessing, and a normalized pJ per bit per megabyte of memory. Refresh impact not included.**

Vendor	Model	MB	width	MHz	V_{DD}	access mA	sleep mA	pJ/bit	pJ/bit/MB
Fujitsu	MBM29LV320xE	32	16	12.5	2	7	0.005	70.0	2.19
Fujitsu	MBM29DS163xE	16	16	10	1.8	8	0.005	90.0	5.63
Intel	28F256K3	32	16	75	2.7	24	0.030	54.0	1.69
Intel	28F64OW30	8	16	40	1.7	8	0.007	21.3	2.66
Intel	28F32OW18	4	16	66	1.7	7	0.008	11.3	2.82
Micron	MT28S2M32B1LC	8	32	66	3	130	0.300	91.6	11.45
Micron	MT28F642D18	8	16	54	1.7	10	0.025	19.7	2.46
NEC	μPD29F032203AL-X	4	16	12.5	2.7	16	0.005	216.0	54
NEC	μPD29F064115-X	8	16	12.5	1.8	15	0.025	135.0	16.88
Samsung	K9F2816x0C	16	16	20	1.65	8	0.010	41.3	2.58
Samsung	K9F2808x0B	16	8	20	1.7	5	0.010	53.1	3.32
Samsung	K9F6408x0C	8	8	20	1.65	5	0.010	51.6	6.45

Table 3: **Low-power Flash characteristics: size, bit-width, speed, voltage, best-case access current, best-case sleep-mode, current use, pJ per bit in accessing, and a normalized pJ per bit per megabyte of memory.**

Vendor	Type	Range	Model	kbps	V_{DD}	TX mA	RX mA	sleep uA	TX μJ/bit	RX μJ/bit
AMI Semi	SpreadS	300m	ASTRX1	40	3.3	14	25.0	10.0	1.155	2.063
AMI Semi	Modem	n/a	A519HRT	1.2	5.0	0.6	0.6	n/a	2.500	2.500
CSR	Bluetooth	100m	BC2-Ea	1500	1.8	53	53.0	20.0	0.064	0.064
MuRata	Bluetooth	100m	LMBTB027	1000	1.8	60	58.0	30.0	0.108	0.104
NovaTel	Wireless	n/a	Expedite	38.4	3.3	175	130.0	5.0	15.039	11.172
OKI Semi	Bluetooth	100m	MK70	921.6	3.3	115	72.0	n/a	0.412	0.258
Option	GSM	n/a	GlobeTrotter	116	3.3	550	50.0	50.0	15.647	1.422
Radiometrix	UHF	30m	BiM-UHF	40	5.0	21	16.0	1.0	2.625	2.000
Siemens	Bluetooth	20m	SieMo S50037	1500	3.3	120	120.0	120.0	0.264	0.264
UTMC	Bus	n/a	UT63M1xx	1000	5.0	190	40.0	n/a	0.950	0.200
Vishay	IrDA	Varies	TFBS560x	1152	5.0	120	0.9	1.0	0.521	0.004
Wireless Futures	Bluetooth	100m	BlueWAVE 1	115.2	3.3	60.9	60.9	50.0	1.745	1.745

Table 4: **Link characteristics: speed, voltage, current draw in various states, and worst-case μJ per bit power consumption for TX and RX.**