Holistic Design and Caching in Mobile Computing

Mwaffaq Otoom and JoAnn M. Paul
Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24061
{motoom, jmpaul}@vt.edu

Abstract
We utilize application trends analysis, focused on webpage content, in order to examine the design of mobile computers more holistically. We find that both Internet bandwidth and processing local to the computing device is being wasted by re-transmission of formatting data. By taking this broader view, and separating Macromedia Flash content into raw data and its packaging, we show that performance can be increased by 84%, power consumption can be decreased by 71%, and communications bandwidth can be saved by an order of magnitude.

Categories and Subject Descriptors
C.1.4 [**Processor Architectures**]: Parallel Architectures – *Mobile processors*

General Terms
Performance and Design.

Keywords
Webpages, Mobile Computing, Application-level Caching, Single Chip Heterogeneous Multiprocessing.

1. Introduction
Two trends are dominating the computing landscape. First, computing devices, overall, are becoming more mobile from both the top down as well as the bottom up; laptops are shrinking in size at the same time cell phones and other personal computing devices are taking on the capability of more general kinds of computing. Second, webpages are becoming the standard of information exchange; many users of computers only or primarily need to know how to access webpages.

At the same time, major design decisions continue to be done in isolation, where architects focus on improving against benchmarks, EDA professions seek to improve design time given the requirements, and communications experts do not think in terms of power and performance impacts on the local device [1].

While the computing industry has utilized trends analysis for many years, the focus has been on technology. Application trends analysis is not a common part of the computer design process. We utilized trends analysis of the application space under the assumption that it might lead to a more holistic way of looking at how the applications, Internet communications and device design intersect in the rapidly changing landscape of mobile computing.

CODES+ISSS'08, October 19–24, 2008, Atlanta, Georgia, USA.

Webpage content is becoming computationally more intensive even as that same content is changing more rapidly. Even a few years ago, real-time information exchange on the Internet was done via very simple formatting. Currently, consumers demand that the formatting is user-friendly, and this is happening in Macromedia Flash files (which we will refer to simply as "Flash" in the rest of the paper). Through our analysis, we observe that Flash formats tend to encapsulate relatively simple data with complex formatting. At the same time, the raw data changes much more rapidly than the formatting of the data.

Our contribution is three-fold. First is the observation that a holistic examination of mobile computing from the perspective of computing application trends can lead to novel application-architecture solutions. Second is our proposed altered Flash structure, itself – introduced here, but which we also intend to further develop. Third is the observation that the overall design of mobile computers will likely be heavily impacted by webpage content and as well as the way webpages are accessed.

2. Webpage Trends Analysis
In addition to overall web content, our investigation focuses on www.mlb.com, a popular webpage that represents 0.023% of the total daily Internet traffic in the world [2], chosen for three reasons. First, sports is a leader in creating demand for real-time information exchange in consumer electronics. Second, sports has an interesting combination of news and entertainment, where information changes rapidly, but packaging is important to the consumer. Finally, we chose to focus on a single website because past observations led us to believe that a few websites are harbingers of trends, and we wished to avoid the inclusion of data that would mitigate this kind of effect. Our particular focus on www.mlb.com was the Flash file used for following a game in progress, which is available during baseball season when there are games to follow. Internet Archive [3] was used to obtain historical webpages. The data has course granularity, but computing trends must be established over short time periods.

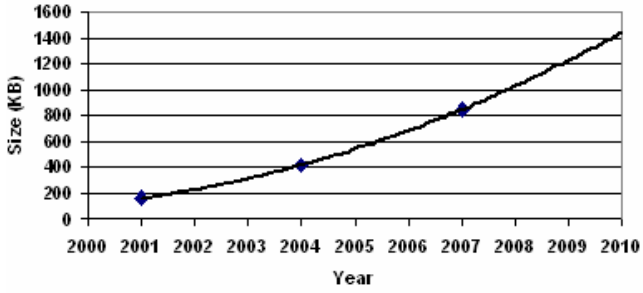

Figure 1. Webpage Total Complexity

Figure 1 shows that total complexity of 2007 webpages is more than 5 times that of 2001 webpages. Our projection is likely conservative – all of our estimates will tend to be conservative.

Multimedia content (still images and movies) are becoming more dominant than text. In 2007, 76% of webpage content is multimedia which represents 1.5 times the 2001 percentage. The trend of future webpage content tends to be media oriented as in image or movie/animation files, a trend which will continue as network bandwidths increase allowing the transfer of large media files to clients in a timely fashion. In 2007, 57% of the multimedia content was Flash, but more importantly, that Flash can be expected to dominate the non-text formatting of webpages ever further, as demand for other kinds of non-textual information on webpages is projected to flatten out by comparison. Image data extracted directly from the outside world (pictures and movies) is slower to generate than forms of raw data which can be computer generated or manipulated.

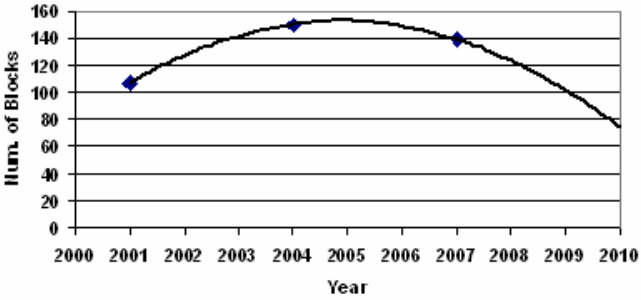

Figure 2. Number of Blocks per Page

Figure 2 shows that the number of blocks (typically files of data) on webpages is decreasing while Figure 3 shows that the number of elements *within* a single Flash file is increasing. In other words, webpage elements like text, links, images, etc. are becoming increasingly encapsulated in Flash formats.

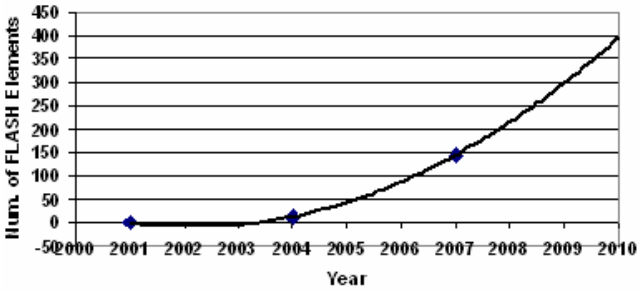

Figure 3. Number of Elements on Flash

Since we are interested in the way Flash will impact mobile computing devices, we are also interested in the dynamic aspects of Flash content, that is, how frequently Flash content is updated. We will refer to the relative frequency of update of information content as *time bins*, so that we may use the concept later.

Figure 4 shows three time bins, where elements are updated every minute, day or year. It also shows that the time bins of these elements are getting smaller. In other words, webpage contents are becoming more dynamic. Webpages are delivering approximately the same *raw data* complexity as in the past, but this raw data is becoming encapsulated with *packaging* to include *presentation data* which is far more computationally intensive than raw data and far less frequently updated. (We will define these terms further in the following sections where we examine them in the context of Flash structure.) Raw data is projected to maintain the same level of complexity while the presentation data grows exponentially over time. This is reasonable; the raw data is used to convey information content from a world that is changing at a much slower pace than a computer's ability to present it.

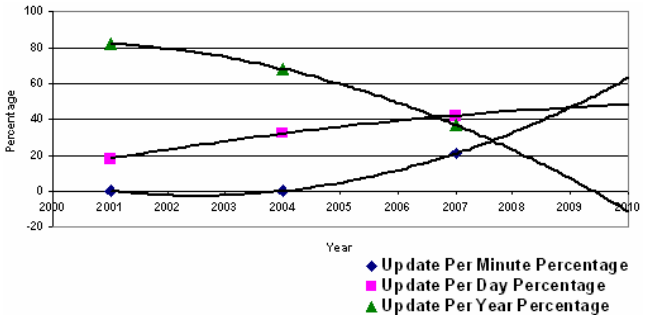

Figure 4. Time Bins

At the same time, the raw data being conveyed by a given Flash file is becoming more frequently updated, which, given the current Flash structure, forces more static presentation data to be transmitted and processed far more than it needs to be. Figure 5 shows that 42% of the 2007 webpage is Flash, where only 17% of the content of these Flash files is dynamic.

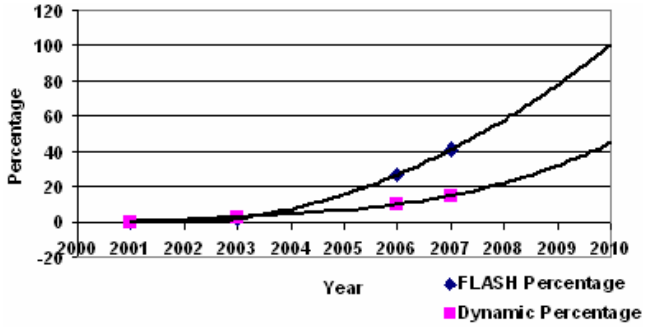

Figure 5. Flash Content Statistics

Overall our trends analysis shows that the increase in information being transmitted across the Internet and processed by mobile computing devices is largely due to formatting – and this trend will likely increase. Since the processing of formatting information is highly bandwidth and compute-intensive, the separation of the raw data being conveyed from its formatting suggests that a caching scheme might save processing time, power and bandwidth. In the next section we examine this further.

3. Flash Structure

Macromedia Flash Player is distributed among over 99% of Web browsers, exceeding that for other media players [4]. The advantages of Flash over other existing animation formats (e.g., GIF) are interactivity, relative small size due to content compression, and efficient rendering through vector graphics. A typical Flash file contains heterogeneous media ingredients (graphics, images, sounds, text, etc).

Figure 6 shows an example of a Flash object (on the left), a stock ticker that shows stock prices both numerically and graphically. These prices are changing every second causing the whole Flash object to change. Our breakdown of the structure of the Flash file is shown on the right, informed by the analysis of the previous section. Our Flash content consists of: raw data (stock prices), presentation data (graphics) and code to process the two (program). We explain these in more detail later. Significantly, conventional Flash processing does not have this structure.

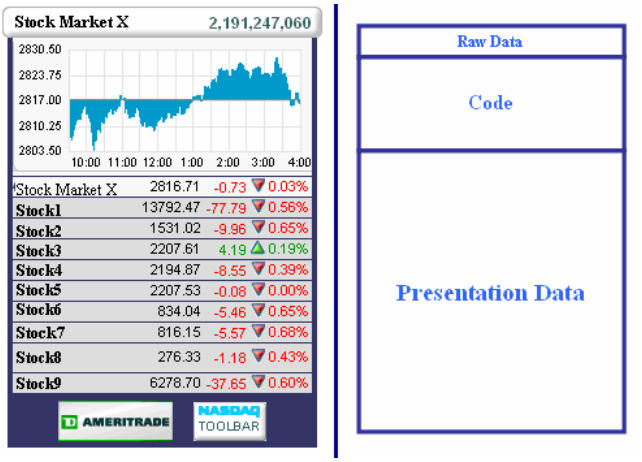

Figure 6. Our Flash Animation File Structure

Conventionally, the client browser pulls the Flash file from the server using the refresh command. The timestamp between pulls is embedded into the HTML document in the header's meta-information that the server sends along with the HTTP response. When time has elapsed the browser requests the webpage content again. When the client requests the Flash file, the server pushes it to the client after generating it. The server utilizes streaming for transferring the Flash file to the client. This means that for the stock market example, the browser will be busy pulling the *entire* Flash file every second in order to keep the user updated, even though only a small amount of actual content will have changed.

By breaking the Flash file down into constituent parts of: raw data, presentation data, and code, we are able to address the increasingly large disparity in time bins associated with how frequently those parts of Flash actually change. Since presentation data and code have approximately the same time bins, we will define the two of them, together, as *packaging*. Thus, Flash content can be broken down into *raw data* and its *packaging*.

An example of presentation data content is the format (font, color, sound, animation, etc) and images used to portray raw data. This data can be encapsulated in Cascading Style Sheets (CSS) or JavaScript (JS) files. Both raw data and presentation data are processed by the Flash program to generate the Flash object by the code, which we consider a part of the packaging. The packaging represents the majority of the Flash files and is far more static in nature than the raw data – even as the raw data is becoming more dynamic. This means that the coupling of the raw data and packaging could be causing tremendous waste in communications bandwidth, processing and power consumption. The Flash file of Figure 6 has raw data size of 0.2KB, while the packaging is 10KB (the presentation data size is 4KB and the program codes size is 6KB).

Figure 7(a) shows the current memory hierarchy and the corresponding theoretical bandwidth [5]. Based on the observation that the dynamic part of the Flash file is on average 10% of the total file size, if a scheme can be introduced to reduce the amount of information communicated on the Web by 90%, the *effective* bandwidth of the Web is increased an order of magnitude. This is shown in Figure 7(b). Previously, any caching of Flash content was done on a computer's disk because it posed no bottleneck to performance. With a more sophisticated caching scheme, such as the one we propose, the hard disk will become the bottleneck in the memory hierarchy as shown in Figure 7(b). Since hard disks are not found in all mobile devices yet, a caching scheme that eliminates the need for storage to disk while effectively improving Internet bandwidth has two-fold value. Also, since Web bandwidth is improving faster than memory bandwidth [6], we expect that the main memory will be the bottleneck in the next a few years as shown in Figure 7(c). Our experiments, discussed later, will take this into account.

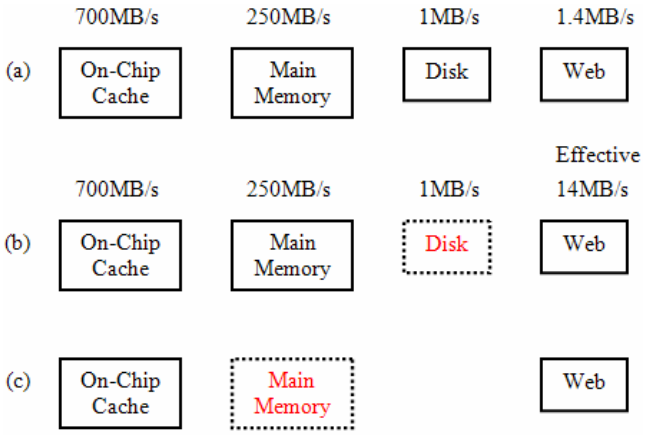

Figure 7. Memory Hierarchy

4. Proposed Caching Scheme

The ultimate goal for mobile computing is to increase device performance while saving communication bandwidth and power consumption. Thus, it seems reasonable to investigate how a caching scheme within Flash structure can address the increasing use of mobile computing devices to process information conveyed in Flash. The cost of caching schemes is in the increased complexity of protocol and the cost of storage. We will investigate these in the context of architecture design later in this paper. In this section, we introduce our scheme.

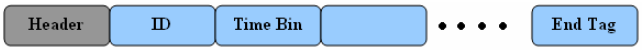

Figure 8. Webpage Object Structure

Figure 8 shows sample webpage object structure format with the suggested tag. In addition to the tag, a simple modification is also required in the Flash protocol between client and server. In our proposed model, the client generates the Flash file locally by executing the code on both the raw data as well as the cached presentation data. A request is made to the web server, the web server then serves up or pushes Flash application code (HTML) up to the client with the presentation data. The web server will also embed pointers back to the web server that will allow the Flash application program to acquire the raw data. Thus, two requests are made to the web server. The first request is for the initial web page that contains a Flash program and presentation data (*INIT*). The program on the client then makes a second request to the server for the raw data (*UPDATE*). Once the Flash program on the client side receives the data it will then process the data and generate the Flash file locally.

If we define P for packaging and R for raw data, more attributes are motivated, which can be leveraged for more intelligent design. The size of the Flash file, F_S, can be defined by two parameters – the packaging and raw data sizes at a given time. The update of the Flash file, F_U, can be defined by the change of the presentation and raw data at a given time. These will be impacted by the nature of the data, as well as the way the mobile computer is utilized, i.e. user preferences [7]. The access frequency of the Flash file, F_F, models

how many times the presentation and raw data are requested due to the change of their content and/or user demand.

$$F_S(P(t), R(t))$$
$$F_U(\Delta P(t), \Delta R(t))$$
$$F_F(\sum_{i=1}^{N} P_i(t), \sum_{i=1}^{M} R_i(t))$$

Using our framework for analysis of application and usage trends, much potential exists to exploit application knowledge in order to impact design decisions of mobile devices, beyond the scope of this paper. Here, we introduce a simple concept for Flash in order to illustrate the need to consider mobile device applications in a more holistic way, even to include the way they are used by individuals and not simply a program or an application (or even sets of programs or applications) at a time. The modification to Flash structure is relatively simple. The real test is whether the cost of storage of packaging in the mobile device will, in general, be a design win. If so, additional usage factors can be considered, such as those outlined in this section.

5. Background

Many surveys about web caching schemes exist [8,9,10]. Current web caching schemes assume that most web access patterns are HTML text and images [11,12,13], none of them have effectively explored web multimedia caching techniques, especially Flash. Conventional page-level caching cannot effectively address our observations about Flash content, because they do not break up the content of blocks – a small change in raw data will still lead to renewed Web content generation and transmission [14].

A number of new caching strategies have been proposed. The fragment-level caching strategy has been proven to be the most practically effective. Conceptually, a fragment unit is a portion of a page that is distinguishable from other parts of the page. This method also does not work for our purposes because in Flash, the fragments are not otherwise distinguished. Prefix Caching [14,15,16] permits the cache proxy to store only the initial frames of popular clips, depending on the received requests for those streams. In addition to prefix caching, grouping the number of block media streams received by the proxy server into variable sized segment, distance-sensitive with initial segment cache, has been utilized. In the Segment-Based scheme, the less popular media is partially cached while the most popular media is fully cached. These schemes are based on popularity [17], which does not correlate well with the form of the rapidly changing data within Flash files.

Conventional caches are ineffective for stream-type data structure, where prefetching may pollute the cache if the prediction is not accurate. Thus, hardware prefetching techniques have been proposed [18]. A simple prefetching mechanism to improve the memory performance of multimedia applications has also been proposed [19]. Each stops short of our proposal to remove streaming from real-time Flash. Our proposal is more consistent with the view that many MPEG implementations of video data are "un-cacheable" [20]. Most data-dominated multimedia applications do not use their cache efficiently [21], largely due to the streaming nature of video, which our approach avoids.

While other efforts focus on the structure of the cache memory, we propose a broader view that encompasses application patterns – specifically multimedia structure and usage patterns.

6. Experiments

Our experiments are designed to investigate whether our proposed technique for Flash reorganization has positive impact on communications bandwidth, processing time, and power consumption. Our experiments cull data from the Web for both architectures and Flash applications. More detailed data for mobile devices as well as future plans for webpage designs are proprietary. However, this paper also has a broader objective – to illustrate how system-level design concerns require designers to think holistically and about application trends when they consider mobile device design.

We designed our experimental system to execute a heterogeneous application set, consisting of three application types with widely varying computational requirements and characteristics. These applications are JPEG, text and Flash which represent the current webpage content types. We used the data we collected from www.mlb.com as an input to the experiment system. The relative performance for each task on each processor is taken from EEMBC [22].

Our target implementation is a single-chip heterogeneous multiprocessor with a fixed area budget. We divided the chip into four regions, to be populated by four categories: Media Processors (M), Digital Signal Processors (D), General Purpose Processors (G) and Chip-Level Cache (C), intended to support the Flash content. Sets A and B consider processor arrangements that fit on a chip when there is no cache on chip. Set C considers processor arrangements that fit on a chip with a 1024K cache set aside for Flash content. These sets produce a total of 41 different architectures, as described in Table 1. Architectures in the table are differentiated by the numbers and types of processors they contain. Future computing devices are expected to have hundreds of heterogeneous processors [23].

Three different processors were chosen for our experiment because of the diversity of the computational capabilities, power consumption, and area requirements. Philips PNX1700 is the Media (M), Blackfin533 is the DSP (D) and the AMD K6-2E+ is the GPP (G). These processors are inadequate for use in a mobile device such as a cell phone due to area and power requirements. However, they are adequate to represent the key relative system-level design trade-offs for fixed-area devices, since they are consistent with each other. Lack of access to proprietary information makes a detailed examination of processors used in mobile computing devices impossible. The area and power consumption for these processors were derived from information available from [24,25,26].

Our experiments are broken down into three parts. These three parts correspond to the three parts of Figure 7, except that we do not include a hard disk since we presume that many future mobile computing devices will not have them. In part A, we consider traditional Web caching only. An object with a time bin tag equal to or is greater than one day is saved in the main memory. Based on the data which was gathered from www.mlb.com, 345KB of the entire webpage content (all of the content on the Website, not just Flash) can be cached for one day, main memory size is sufficient to host hundreds of times this amount. Currently, Apple iPhone has 128MB of main memory [27]. This produced 25 different architectures as shown in Set A and B of Table 1.

In part B, we combine the Web caching with our suggested Flash file structure where a Flash file is broken into raw data and presentation data. The caching is again done in the main memory. Based on the collected data from www.mlb.com, 640KB is the data which can be cached in main memory for one day. In this part we are able to cache the packaging content of the Flash file. The same architecture set is produced as in part A since the caching is done in main memory.

Table 1. Processing Elements for Sets A, B and C

Arch.	Set A, B	Arch.	Set C
1	4G	26	3G
2	8D	27	6D
3	16M	28	12M
4	4M, 3G	29	8M, 1G
5	8M, 2G	30	4M, 2G
6	12M, 1G	31	4D, 1G
7	6D, 1G	32	2D, 2G
8	4D, 2G	33	6M, 1D, 1G
9	2D, 3G	34	4M, 2D, 1G
10	2M, 5D, 1G	35	2M, 3D, 1G
11	4M, 4D, 1G	36	2M, 1D, 2G
12	6M, 3D, 1G	37	10M, 1D
13	8M, 2D, 1G	38	8M, 2D
14	10M, 1D, 1G	39	6M, 3D
15	2M, 3D, 2G	40	4M, 4D
16	4M, 2D, 2G	41	2M, 5D
17	6M, 1D, 2G		
18	2M, 1D, 3G		
19	2M, 7D		
20	4M, 6D		
21	6M, 5D		
22	8M, 4D		
23	10M, 3D		
24	12M, 2D		
25	14M, 1D		

Finally, in part C, we again combine the Web caching with our suggested Flash file structure. In this part, the caching is done on chip. We anticipate that if caching can effectively be done on the chip, we may be able to find additional power savings. This results in 16 additional architectures to consider as shown in Set C of Table 1. The 1024K cache was chosen to fit the contents collected from www.mlb.com home page, where each Flash file can represent a different game in progress. The data which can be cached for one day is 640KB – the same data as in Part B.

eCacti [28] was used to determine the cache area and cache power consumption based on the selected cache size, block size and technology. We assumed a 0.13µ manufacturing technology. Also, we assumed that the power consumption when a processor is in an idle state is 20% of its active power consumption [29]. Furthermore, we assumed a perfect memory system, i.e., caches always have the requested data. This is reasonable, since we are interested in the many applications which persist in the system for hours or even days, constantly updating information in real time. While we do not consider the initial cost of loading the packaging data for the Flash, it would only be significant if the entire working set of Flash content did not fit into the device's cache.

For simulation, we used a C-based simulation tool called MESH (Modeling Environment for Software and Hardware), which permits performance and power evaluation when threads execute on sets of heterogeneous resources under a variety of custom schedulers [30]. Energy consumption is calculated as the summation of the power consumed during ON and IDLE times [30]. We added energy consumed by cache or main memory, thus,

$$E = \sum_{i=0}^{N} (PE_i\_ON * PE_i\_APW) + (PE_i\_IDLE * PE_i\_IPW) + \sum_{i=0}^{M} CA_i$$

Where, N is the number of processors in a specific architecture. M is the number of cache or main memory accesses. PE_i_ON is the total time where PE_i is in ON state, PE_i_IDLE is the total time where PE_i is in idle state, PE_i_APW is the active power for PEi processor, PE_i_IPW is the idle power for PE_i processor and CA_i is the power consumed by the cache or main memory when cache read happened.

Our scheduling strategy is the best available resource scheduler – each task is scheduled on the best available resource by order of appearance in the queue. Scheduling overhead is not included in the model. Our prior work has shown that scheduling overhead for single chip heterogeneous multiprocessors is insignificant without inter-job data dependences [30]. In Figures 9 - 11, the best performer is normalized at 1; thus lower numbers are always poorer performers. Table 1 architectures are on the independent axes with Sets and Experiments grouped as A, B, and C.

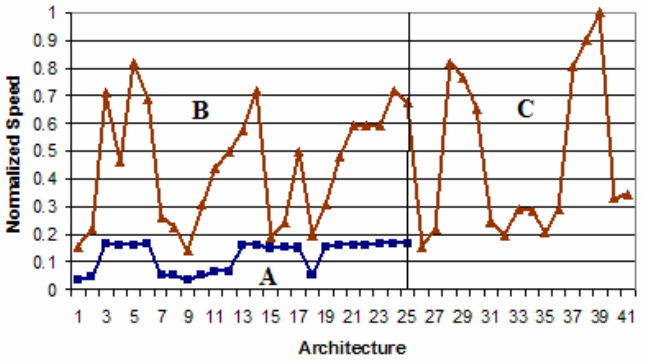

Figure 9. Loading Time

Figure 9 shows the normalized webpage loading time. Normalized speed is the reciprocal of dividing the architecture speed by the lowest speed among all architectures. Parts B and C, which are our proposed approach for caching Flash content, shows a significant improvement over that of part A. It also shows the effects of picking an optimal architecture even within the optimized Flash caching structure.

Figure 10 shows the normalized energy consumption over all architectures. Normalized energy consumption is the reciprocal of dividing the architecture consumed energy by the lowest consumed energy among all architectures. The decrease in energy consumption is significant when using our approach and, again, there is an interesting variation – and the potential for future work – for architectural optimization within our new framework. Part C – caching the data including the Flash file breakdown in on chip cache – results in significant improvement in energy consumption. This is in part due to the use of fewer processors. But it also points in the direction of the possibility that more aggressive caching schemes could result in further performance improvements.

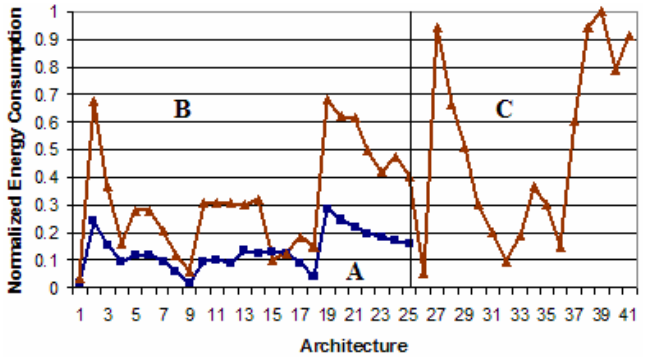

Figure 10. Energy Consumption

Our scheme used a relatively simple separation of raw data and packing – more sophisticated vector schemes could ultimately reduce network bandwidth even further, posing the possibility that main memory may actually become the performance bottleneck in the future.

Figure 11 shows the normalized value of speed and energy consumption or $NV = N(NS + NE)$, where, NV is the Normalized Value resulting from the two values; *NS* which is the Normalized Speed and *NE* which is the Normalized Energy. This graph shows an even more significant advantage for our approach – especially Part C – as well as interesting variation between architectures that utilize our system-level caching scheme.

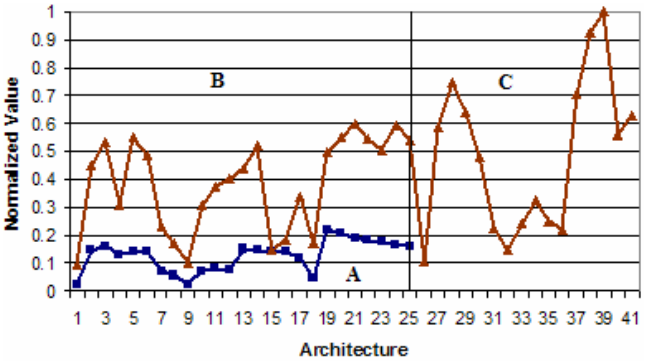

Figure 11. Loading Time and Energy Consumption

While we focused on www.mlb.com for the reasons previously described, these results are more broadly applicable. We found similar results for the stock ticker example of Figure 6. There, the raw data is 0.2KB and is changing every minute or even every second. The presentation data is 4KB and is changes every week or more. The code is 6KB and is rarely changes.

7. Conclusions

We illustrated a simple caching scheme, which resulted in a performance increase of 84%, a decrease in power consumption by 71%, and an order of magnitude savings in communications bandwidth for mobile computing devices that process webpages. We did this by observing and then leveraging the trends in the time granularity and content of "raw data" from "packaging" in Macromedia Flash files.

More significantly, we illustrated that mobile computing devices must be designed using more holistic techniques than in the past. An analysis of how applications interact with Internet bandwidth and the processing and storage capabilities of mobile devices is a start. We utilized an examination of application trends so that we would be able to find an application hot-spot, thereby bringing together several major facets of mobile computer design. In so doing, our hope is to open up a new dialogue in computer design methodology, less narrowly focused on isolated, and often incremental, performance improvements, instead taking a broader view in pursuit of new ideas for the next frontier in computing.

8. Acknowledgements

This work was supported in part by the National Science Foundation (grant 0607934). Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

9. References

[1] J.M. Paul, D.E. Thomas, A. Bobrek. "Scenario-Oriented Design for Single-Chip Heterogeneous Multiprocessors." *IEEE Trans. VLSI*, pp. 868-880. Aug. 2006.

[2] Alexa – the web information company, http://www.alexa.com

[3] Internet Archive, http://www.archive.org

[4] http://www.adobe.com/products/player_census/flashplayer/

[5] www.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/

[6] D. Patterson. "Latency lags bandwidth," ICCD, pp. 3, 2005.

[7] M. Somers, J.M. Paul. "Webpage-Based Benchmarks for Mobile Device Design," *ASPDAC*, pp. 795-800, 2008.

[8] J. Wang. "A survey of web caching schemes for the Internet," *ACM Computer Communications Review*, 29(5), pp. 36 - 46, 1999.

[9] B. D. Davison. "A Survey of Proxy Cache Evaluation Techniques," *4th International Web Caching Workshop*, pp. 67-77, 1999.

[10] P. Stefan and B. Laszlo. "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, 35(4), pp. 374 - 398, 2003.

[11] A. Mahanti, C. Williamson, D. Eager. "Traffic analysis of a Web proxy caching hierarchy," *IEEE Network*, 14(3), pp. 16-23, 2000.

[12] C. Cunha, A. Bestavros, M. Crovella. "Characteristics of WWW Client-based Traces,"*TR-95-010, Boston University, CS,* 1995.

[13] M. Arlitt, C. Willliamson. "Web Server Workload Characterization: The Search for Invariants," *ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, pp. 126-137, 1996.

[14] S. Sen, J. Rexford, D. Towsley. "Proxy prefix caching for multimedia streams," *IEEE INFOCOM*, pp. 1310-1319, Mar 1999.

[15] S. Gruber, J. Rexford, A. Basso. "Protocol Considerations for a Prefix-Caching Proxy for Multimedia Streams," *Computer Networks*, 33(1-6), pp. 657-668, 2000.

[16] K. Wu, P. S. Yu, J. Wolf. "Segment-based proxy caching of multimedia streams," *Intl. conf. on WWW*, pp. 36-44, 2001.

[17] S. Jin, A. Bestavros, "Popularity-aware Greedy Dual-Size Web proxy caching algorithms," *IEEE ICDCS*, pp. 254-261, 2000.

[18] J. Lee, C. Park, S. Ha. "Memory access pattern analysis and stream cache design for multimedia applications," *ASP-DAC*, 2003.

[19] H. Sbeyti, S. Niar, L. Eeckhout. "Adaptive prefetching for multimedia applications in embedded systems," *DATE*, 2004.

[20] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, H. De Man. "Cache conscious data layout organization for embedded multimedia applications," *DATE*, pp. 686 – 691, 2001.

[21] H. Van Antwerpen, N. Dutt, R. Gupta, S. Mohapatra, C. Pereira, N. Venkatasubramanian, R. von Vignau. "Energy-aware system design for wireless multimedia," *DATE,* pp. 1124-1129, 2004.

[22] www.eembc.org

[23] http://www.eetimes.com/showArticle.jhtml;?articleID=206105179

[24] www.analog.com/en/epProd/0,,ADSP-BF533,00.html

[25] PNX17xx Series, www.nxp.com/pip/PNX17XX_SER_N_1.html

[26] AMD-K6 Series, www.amd.com/epd/processors/6.32bitproc

[27] http://www.semiconductor.com/resources/reports_database/view_device.asp?sinumber=18016

[28] M. Mamidipaka, N. Dutt, "eCacti: An Enhanced Power Estimation Model for On-chip Caches," *Technical Report #04-28, UCI*, 2004.

[29] P. Babighian, L. Benini, E. Macii. "Sizing and characterization of leakage-control cells for layout-aware distributed power-gating," *DATE*, vol. 1, pp. 720-721, 2004.

[30] B.H. Meyer, J.J Pieper, J.M. Paul, J.E. Nelson, S.M Pieper, A.G. Rowe. "Power-performance simulation and design strategies for single-chip heterogeneous multiprocessors," *IEEE Trans. Computers*, Vol. 54, No. 6, pp. 684- 697, 2005.