Task Allocation for Distributed Multimedia Processing on Wirelessly Networked Handheld Devices *

Zhiyuan Li † Cheng Wang Rong Xu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{li,wangc,xur}@cs.purdue.edu

Abstract

In this paper, we consider distributed multimedia processing on wirelessly networked handheld devices, where multimedia data are heavily processed when transferred between the server and the handheld. We show that, to minimize battery consumption on the handheld, a tradeoff often needs to be made between the energy consumed by computation versus that by data transfer. We develop an energy cost model and present a task partition and allocation scheme to divide the data-processing responsibility between the server and the handheld. Experiments are performed on a Compaq iPAQ device which is connected to a server via a WaveLan interface. Out of 17 programs from the Mediabench suite, our scheme is shown to save energy significantly in 13 of them over the default mode of processing data locally on the handheld.

*This work is sponsored in part by National Science Foundation through grants ACI/ITR-0082834 and CCR-9975309, by Indiana 21st Century Fund, and by a donation from Sun Microsystems, Inc.

†The author names are listed in alphabetical order.

1 Introduction

The use of handheld devices coupled with wireless network interfaces is growing rapidly. Such wireless connectivity to a server makes the immense information and abundant services on the network accessible to the handheld devices. One of the major challenges for effective use of such devices is to reduce the energy consumption thereby extending the battery life. In addition to that, handheld devices have relatively limited processing power and memory, which places a constraint on the types and the sizes of applications feasible on such devices.

In our recent work, we studied *computation offloading* techniques for reducing execution time and energy consumption on handheld devices, assuming both the input and output are local [5]. We proposed a program execution model and a task partition/allocation scheme to allow the computation to be offloaded, either entirely or partially, from the handheld to the server through a wireless LAN. Our experiments showed substantial reduction in both execution time and energy consumption for a suite of test programs. In this paper, we extend our work by considering distributed multimedia file processing, where multimedia files are heavily processed when transmitted between the server and the handheld via a wireless network.

For example, replaying downloaded images, sounds, or videos from server to a handheld requires decompression if those data were previously compressed. On the other hand, transmitting recorded images, sounds and videos from a handheld to the Internet, via the server, may require compression. Since it is the handheld device which initiates data transfer, by default the responsibility for the associated data processing is on the handheld device. However, such a default choice may suffer time penalty and battery energy loss. Where to perform processing can be a quite complex problem. Processing on the server obviously reduces processing time and the battery consumed by the CPU. However, processing on the server may require additional data communication (as in the examples above), which may increase the time and the battery consumed by communication. Furthermore, it is possible to partition the entire processing into tasks and divide them between the server and the handheld to minimize the elapsed time or the battery, or both. Figure 1 illustrates two data transmission scenarios, both of which involve heavy processing.

Distributed multimedia data processing can be viewed as an extension to traditional distributed file operations such as open, close, read, write and directory listing, which are relatively simple. In this paper, we develop an energy cost model and propose a task partition/allocation scheme to minimize energy consumption on the handheld device for the scenarios illustrated above. A similar framework can be developed to minimize the execution time and these two minimization goals often do not conflict. In this paper, however, we focus on minimizing the energy consumption and do not intend to make a tradeoff between execution time and energy consumption. We perform experiments by running a well-known suite of multimedia benchmarks, namely Mediabench, on a Compaq iPAQ handheld device connected to a desktop machine via WaveLan.

In the remaining sections of the paper, we first briefly surveys related work (Section 2). We then present our overall scheme and the task partition/allocation algorithm (Section 3). We report experimental results in Section 4 and conclude in Section 5.

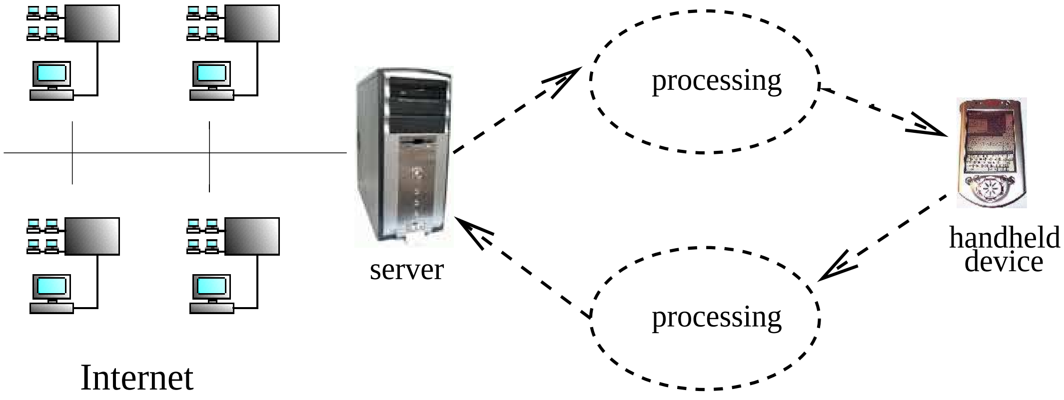


Figure 1: Multimedia Data Transmission & Processing Scenarios

2 Related Work

Although the idea of computation offloading on mobile computers has been explored previously [7, 6, 4, 5], most of previous works target laptop computers instead of handhelds. According to Farkas et al. [2], the energy behavior is quite different on these two kinds of systems.

Rudenko et al. [7] describe experiments using laptops with an early version of WaveLan radio devices. They run three desktop programs, namely the LaTex text formatting software, the GCC compiler, and a Gaussian Elimination program. They conclude that, on a noisy wireless environment, most often offloading does not save energy. On a noiseless environment, however, they show significant energy saving by offloading in some cases. Our work is based on handheld devices, running multimedia codes, and we develop cost models and task allocation schemes to obtain positive results. Othman and Hailes [6] use simulation to show that battery life can be extended through migration at the process level. They use synthetic workload in their study.

There exist several research projects which provide proxy services to mobile computers. For example, the proxy in GloMop [1] provides two functionality to the mobile client. It performs distillation to convert the data to another form which may reduce the size but preserves certain semantic information useful to the user of the handheld device. It also performs incremental refinement to allow the handheld device to request further detailed information. Their system is application-specific while our scheme is general and can be oblivious to the application semantics. Moreover, our scheme adapts to system parameters and it can choose the handheld to perform the processing if it is more beneficial that way.

To our best knowledge, previous papers on offloading assume both input and output to be local. In contrast, our current paper applies to distributed multimedia processing where files are transmitted between the handheld and the server. Our previous work uses branch-and-bound, which has exponential worst-case complexity, to optimize the partition/allocation problem. This paper uses maximum-flow/minimum-cut algorithm whose complexity is $O(N^3)$ where N is the number of tasks in our task graph.

3 The Scheme

To facilitate task partitioning, we have previously developed a program execution framework in which a given program is divided into server tasks and client tasks [5]. We continue to use that framework in this paper. Statically, under our execution model, a task corresponds to a procedure (or a function) call site. Dynamically, a task is a single invocation of the corresponding procedure.

It is important to note that *we do not seek to execute a client task and a server task simultaneously.* This is due to the significant computation speed gap between the two. One sees no realistic gain by parallel execution in this manner. On the other hand, potential advantage may be gained by overlapping client computation with communication. However, we presently do not take such an advantage.

In the absence of simultaneous tasks, program execution simply follows the original sequential control flow. Function invocation serves as a natural task scheduling mechanism at the run time. If a server task invokes a client task, or vice versa, a remote procedure call is performed through message passing.

After the partitioning, all the tasks mapped to the same host can share the program state as they do in the original program. The issue of coherence arises, however, when one deals with data shared between two tasks which end up in different hosts. At the present, we maintain the program state coherence in a conservative way. A piece of data can be shared by both hosts only if the program transformations and message passing can guarantee the correct dependence relations incurred on that data.

3.1 The cost graph

To characterize the energy costs of computation and communication, we make use of the following parameters.

- b_s: the sending bandwidth of the wireless network.
- b_r: the receiving bandwidth of the wireless network.
- p_i: the mean power consumption rate when the handheld device is idle
- p_c: the mean power consumption rate when the handheld device performs computation.
- p_s: the mean power consumption rate when the handheld device sends data.
- p_r: the mean power consumption rate when the handheld device receives data

We integrate both the computation cost and the data communication cost into the directed cost graph G = (V, E) defined below

- Each node $v \in V$ represents a task in the program. We use e_c to denote the energy consumed by executing v as a client task (i.e. on the hand-held device), Similarly, we use e_s to denote the energy consumed by the idling handheld device if v is executed as a server task. We attach the value pair (e_c, e_s) to v. To get their values, we use profiling to first measure $t_c(v)$

and $t_s(v)$, the execution time for v as a client task and as a server task respectively. We then have

$$e_c(v) = t_c(v)p_c$$
$$e_s(v) = t_s(v)p_i$$

- There are two special nodes $s \in V$ and $d \in V$ that represent all the tasks that must run on the local device and remote server respectively.
- Each edge $(v_i, v_j) \in E$ represents the data transfer from node v_i to node v_j. We use e_l and e_r to represent the energy consumption for data transferring from client to server, and vice versa, respectively. We attach a value pair (e_l, e_r) to each edge (v_i, v_j). Let $s(v_i, v_j)$ be the size of data transfer for from v_i to v_j and we have

$$e_l(v_i, v_j) = \frac{s(v_i, v_j)}{b_s} p_s$$
$$e_r(v_i, v_j) = \frac{s(v_i, v_j)}{b_r} p_r$$

The task mapping can be represented as a cut of the cost graph (S, D) where S is the set of tasks that will be mapped to the handheld device, and $D = V - S$ is the set of tasks mapped to the remote server. By definition, we have $s \in S$ and $d \in D$.

The total computation cost for the specific task mapping will be

$$\sum_{v \in S} e_c(v) + \sum_{v \in D} e_s(v) \tag{1}$$

and the total communication cost for that task mapping will be

$$\sum_{v_i \in S, v_j \in D} e_l(v_i, v_j) + \sum_{v_i \in D, v_j \in S} e_r(v_i, v_j) \tag{2}$$

Based on the above formula, the mapping problem is:

find the cut of the cost graph $(S, D), s \in S, d \in D$, such that the sum of communication cost (1) and computation cost (2) is minimized.

3.2 The task mapping algorithm

We can transform the mapping problem to a maximum-flow/minimum-cut problem. The cost graph $G = (V, E)$ is transformed to a flow network $G' = (V', E', C')$ where V' is the node set, E' is edge set and C' is the flow capacity for all edges. The transform is described as follows.

- **Rule 1:** Each node $v \in V$ in the cost graph is transformed to a node $v' \in V'$. The special node s and d are transformed to the source node s' and sink node d' of the flow network respectively.

- **Rule 2:** Each edge $(v_i, v_j) \in E$ in the cost graph is transformed to two edges (v'_i, v'_j) and (v'_j, v'_i) in E' with capacity $e_l(v_i, v_j)$ and $e_r(v_i, v_j)$ respectively.
- **Rule 3:** For each task node $v \in V$ other than the two special nodes in the cost graph, add two edges (s', v') and (v', d') to E' with capacity $e_s(v)$ and $e_c(v)$ respectively.

It is easy to see that for each cut (S', D') of the flow network, there exists a corresponding cut (S, D) of the cost graph, and vice versa. Furthermore we have the follow Lemma.

Lemma: If (S', D') is a cut of flow network G', then the total cut value of the flow network

$$\sum_{v'_i \in S', v'_j \in D'} C'(v'_i, v'_j)$$

equals to the total energy cost for the mapping of the cost graph represented by the corresponding cut (S, D).

proof: We will show in the following that there exists a one-to-one mapping between the capacity of each cut edge in the flow network and the energy cost in the cost graph.

- If the cut edge (v'_i, v'_j) corresponds to the edge (v_i, v_j) in the cost graph (c.f. Rule2), then v_i will be mapped to the handhelds device and v_j will be mapped to the server. That means there will be data transfer from the handhelds device to the server. The capacity of edge (v'_i, v'_j) equals to the data transfer cost of the edge (v_i, v_j) by definition.
- If the cut edge (v'_i, v'_j) corresponds to the edge (v_j, v_i) in the cost graph (c.f. Rule2), then v_i will be mapped to the handhelds device and v_j will be mapped to the server. That means there will be data transfer from the server to the handhelds device. The capacity of edge (v'_i, v'_j) equals to the data transfer cost of the edge (v_j, v_i) by definition.
- For the cut edge (s', v') which is added according to Rule 3 for node v in the cost graph, node v will be mapped to the server. The capacity of edge (s', v') equals to the computation cost of node v.
- For the cut edge (v', d) which is added according to Rule 3 for node v in the cost graph, node v will be mapped to the handhelds device. The capacity of edge (v', d) equals to the computation cost of node v

$\diamond$

By the above lemma, if (S', D') is a minimum cut in the flow network, then its corresponding cut (S, D) in the cost graph is an optimal solution to the mapping problem. There exist many minimum cut algorithms which have polynomial complexity. We use the the preflow-push algorithm [3] which has $O(N^3)$ complexity where N is the number of the task nodes.

Table 1: Test programs

Program Name	Description	Input Parameters
toast	GSM in Mediabench, GSM voice transcoding	toast -fpl clinton.pcm
untoast	GSM in Mediabench, GSM voice transcoding	untoast -fpl clinton.pcm
encode	G.721 in Mediabench, CCITT voice compression	encode -4 -l <clinton.pcm >out.g721
decode	G.721 in Mediabench, CCITT voice decompression	decode -4 -l <clinton.pcm.g721 >out.pcm
epic	EPIC in Mediabench, image compression	epic test_image -b 25
unepic	EPIC in Mediabench, image decompression	unepic test_image.E test_image.out
pgp/encrypt	PGP in Mediabench, encryption	pgp -fes Bill -zbillms -u Bill <pgptest.plain >pgptext.pgp
pgp/decrypt	PGP in Mediabench, decryption	pgp -fdb -zbillms <pgptext.pgp >pgptest.dec
rawcaudio/compress	ADPCM in Mediabench, speech compression	rawcaduio <clintonpcm >out.adpcm
rawcaudio/decompress	ADPCM in Mediabench, speech decompression	rawcaduio <clinton.adpcm >out.pcm
ghostscript	Postscript interpreter	gs -sDEVICE=ppm -S OutputFile=tast.ppm -dNOPAUSE -q –tiger.ps
cjpeg	IJG JPEG in Mediabench,image compression	cjpeg -dct int -progressive -opt -outfile out.jpeg nasa.ppm
djpeg	IJG JPEG in Mediabench,image decompression	djpeg -ppm -outfile out.ppm nasa.jpeg
pegwit/encrypt	PEGWIT in Mediabench, encryption	pegwit -e my.pub pgptest.plain pegwit.enc <encryption_junk
pegwit/decrypt	PEGWIT in Mediabench, decryption	pegwit -d pegwit.enc pegwit.dec < my.sec
mpeg2encode	MPEG in Mediabench, mpeg encoding	mpeg2encode test.par out.m2v
mpeg2decode	MPEG in Mediabench, mpeg decoding	mpeg2decode -b mei16v2.m2v -r -f -o0 rec%d

4 Experiments Result

4.1 Experiments setup

The handheld device used in our experiment a Compaq iPAQ 3650 which has a 206MHZ Intel StrongArm SA1110 processor and 32MB RAM. The back-end server is a PIII 866MHZ Dell Dimension XPS. We run Linux on both machines. The wireless connection is through a Lucent Orinoco(WaveLan) Golden 11Mbps PCMCIA card inserted into a PCMCIA expansion pack for the iPAQ. During the experiments, we maintain the ordinary wireless LAN environment in which many wireless devices use the same access points for communication at the same time. Throughout the experiment, we turn off the screen on the handheld device. We measure the total elapsed time of each program and the current drawn on the handheld.

4.1.1 Test programs

Table 1 lists the programs used in our experiments, their descriptions and their input parameters, including the input files we use, which can be found in the *Mediabench* web-site. A program may have rather different execution paths to provide different functionalities, we perform tests on those different paths, using different profile information to generate the cost graph. As a result, it may have different versions of transformed code.

We were unable to locate the full source code of *NIST SPHERE + RASTA*, which therefore is omitted in the experiments. We have not yet performed tests on *Mesa*.

4.1.2 Physical measurement

We connect an HP 3458a high precision digital multi-meter to measure the actual current drawn of the hand-held computer during the program execution. In order to get a reliable and accurate

Table 2: Power parameters

iPAQ	Send/Recv	Current(mA)	iPAQ	Send/Recv	Current(mA)
idle	no	300	busy	no	430
-	send	410	-	recv	390
busy	send	510	busy	recv	480

reading, we disconnect the batteries from both the iPAQ and the extension pack and we use an external 5V DC power supply instead. Further, we make use of the build-in trigger mechanism in the multi-meter. After the test program starts running, the iPAQ triggers the meter to do the reading in a high frequency. The trigger stops as the test program finishes. According to our real measurement, the overhead associated with the triggering interrupts is less than 0.5% and the readings are consistent over repeated runs.

We measure the electrical current drawn in different running modes. The readings are shown in Table 2. In this table, the first column *iPAQ* lists the functioning mode of iPAQ: *idle* means iPAQ doing nothing; *busy* means iPAQ performing intensive computation. The *Send/Recv* column indicates whether there is intensive sending or receiving. During sending or receiving, the computer is neither idle nor computation-intensive. We just put '-' in the iPAQ column. The last column lists the electrical current drawn from the external DC power. Note that all the numbers we get here are with the screen off (When using the medium back-light, the current will increase about 140mA).

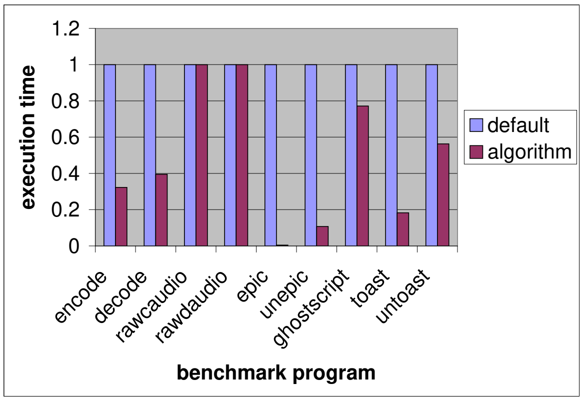
In the experiments, we assume that the WaveLan is to be always connected, mimicking the realistic situations in which the users wish to maintain the continuous readiness for services such as email and instant messaging. Therefore, we use the electrical current reading in the second and the third row of the table. For the 5V voltage, we get $p_i = 1.5w$, $p_c = 2.15w$, $p_s = 2.05w$, $p_r = 1.95w$, approximately. These parameters are used when we apply our partition algorithm.

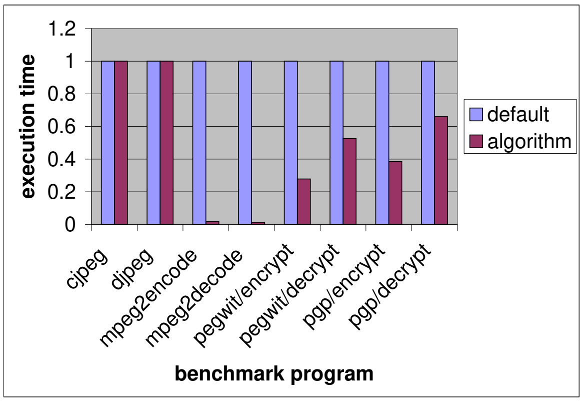
4.2 Energy data

We apply our task allocation algorithm to the programs listed in Table 1. Most of the Mediabench programs performance encoding/decoding, compression/decompression, or other similar operations. If the data get consistently reduced by processing and the data is transfer from the server to the handheld, it is obvious that the processing should be done by the server, because handhelds save both the energy on computation and energy on receiving the data. In contrast, If the data get consistently increased by processing and the data is transfer from the handheld to the server, it is obvious that the processing should be done by the server. In the experiments, we only measure the non-trivial cases. Based on profiling, we know whether the data file consistently gets inflated or reduced in size.

Our algorithm predicts that out of these 17 test programs, 13 of them save energy by choosing not to process the data completely on the handheld device. For the remaining 4 programs, namely, *cjpeg*, *djpeg*, *rawcaudio/compress* and *rawcaudio/decompress*, our algorithm finds that the most energy-efficient running mode is to process locally.

Figure 2 compares the running time of the benchmark programs, where the "*default*" label means processing on the handheld device. the "*algorithm*" label indicates the code generated by our algorithm. Since the running of different programs varies significantly, we normalized running time on the handheld to 1. The table associated with the figure lists the running mode and the corresponding running time. The symbol $c \rightarrow s$ indicates that the input is from the handheld; output goes to the server, while the symbol $s \rightarrow c$ means the opposite.





(Running Time)

program	mode	default (sec)	algorithm (sec)
encode	c→s	2.18	0.70
decode	s→c	4.20	1.61
rawcaudio	c→s	0.23	0.23
rawdaudio	s→c	0.26	0.26
epic	c→s	14.50	0.07
unepic	s→c	1.80	0.19
ghostscript	s→c	13.14	10.14
toast	c→s	2.80	0.51
untoast	s→c	1.14	0.64

(Running Time)

program	mode	default (sec)	algorithm (sec)
cjpeg	c→s	0.19	0.19
djpeg	s→c	0.12	0.12
mpeg2encode	c→s	118.80	2.04
mpeg2decode	s→c	83.20	1.10
pegwit/encrypt	c→s	0.59	0.17
pegwit/decrypt	s→c	0.45	0.23
pgp/encrypt	c→s	0.75	0.28
pgp/decrypt	s→c	0.60	0.40

Figure 2: running time on handheld device

Figure 3 shows the average electrical current drawn during the execution of the program. In some cases, the electrical current is slightly higher when the processing is done on the server. The reason for that is the time spent on sending and receiving data dominates the overall running time.

Figure 4 plots the energy consumed by each program. The energy of processing on handheld is again normalized to 1. We compute the energy consumption of system during the execution by the simple equation:

$$energy = voltage * current\_drawn * elapsed\_time$$

The data shows that to save the energy, for 13 of these 17 test programs, our algorithm beats the default.

For the remaining 4 programs, we also did experiments to verify the correctness of our algorithm. We let the server completely process the task and compare the energy consumption with the data in Figure 4. We found that the energy consumption almost doubles for two programs, *rawcaudio*

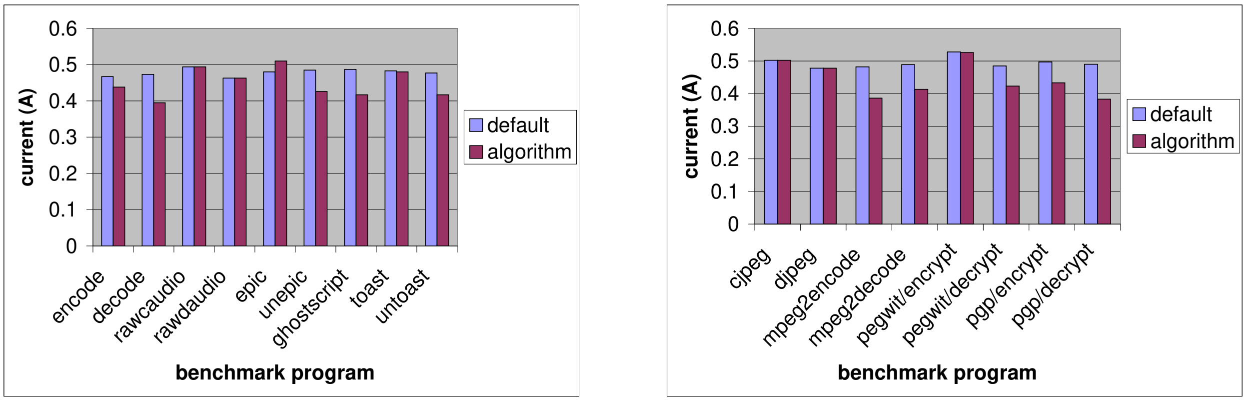


Figure 3: current drawn in the handheld device

and *rawdaudio*. For *djpeg*, the result is almost the same as the default. For *cjpeg*, the energy consumption increases by about 40%.

We should point out that the reason for *EPIC* and *mpeg2* to save energy so significantly is because StrongArm does not have a floating point unit. It uses software floating point library which is extremely slow.

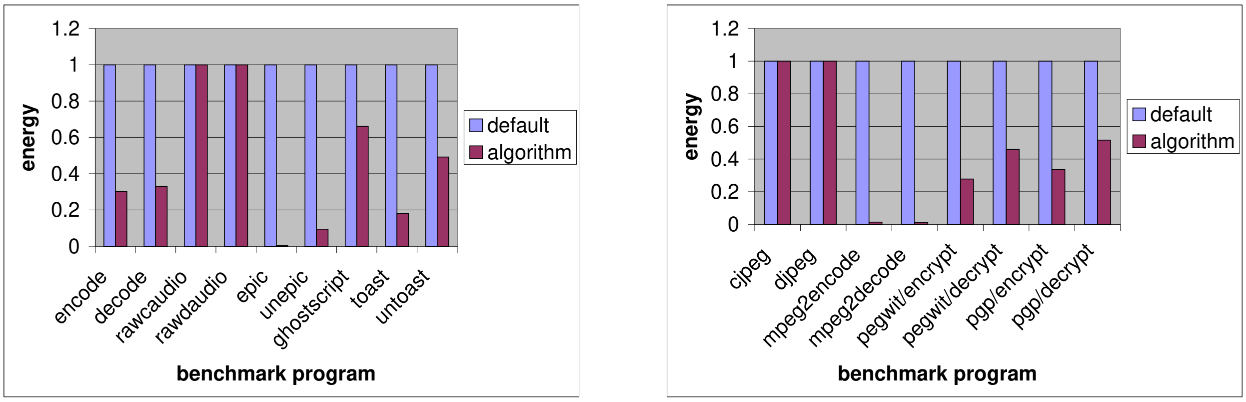


Figure 4: normalized energy consumption in the handheld device

The above results are obtained with the wireless bandwidth set to the default, 11Mbps. We choose G.721/encode as an example to study the energy consumption under different bandwidths. Figure 5 shows the energy result in Joule. One can see that with bandwidth decreasing, in this case, energy spent on data transfer starts to dominate the total cost.

We also experimented with different input sizes to measure their effect on the result. Again we use G.721/encode as an example, choosing the input sizes from 16K to 4M bytes. Figure 6 shows

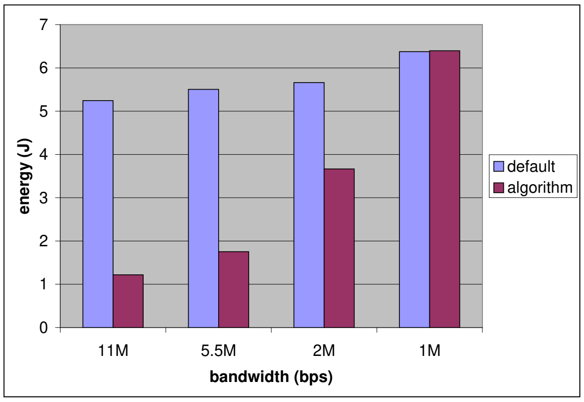


Figure 5: Energy consumed by G.721/encode with different bandwidths

(a) the execution time, (b) average electrical current drawn and (c) the energy consumption. It is easy to see that the execution time and energy consumption consistently increases linearly .

5 Conclusion

In this paper, we have presented a scheme to partition and allocate computational tasks for distributed multimedia processing on wirelessly networked handheld devices. We have targeted common scenarios in which multimedia data are heavily processed when transferred between the server and the handheld device. Our experiments show that the scheme can result in significant energy-saving over the default mode of processing locally on the handheld device. We plan to conduct more extensive experiments after the complete testbed is implemented.

6 Acknowledgment

The authors thank Peifeng Ni for collecting part of the experimental data and Dr. David Yau for his helpful discussions.

References

[1] A.Fox and E.Brewer. Glomop: Global mobile computing by proxy. *CS Dept. Univ of California Berkeley. http://http.cs.berkeley.edu/~fox/glomop*, 2001.

[2] Keith I. Farkas, Jason Flinn, Godmar Back, Dirk Grunwald, and Jennifer M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. *ACM International Conference on Measurement and Modeling of Computer Systems*, 2000.

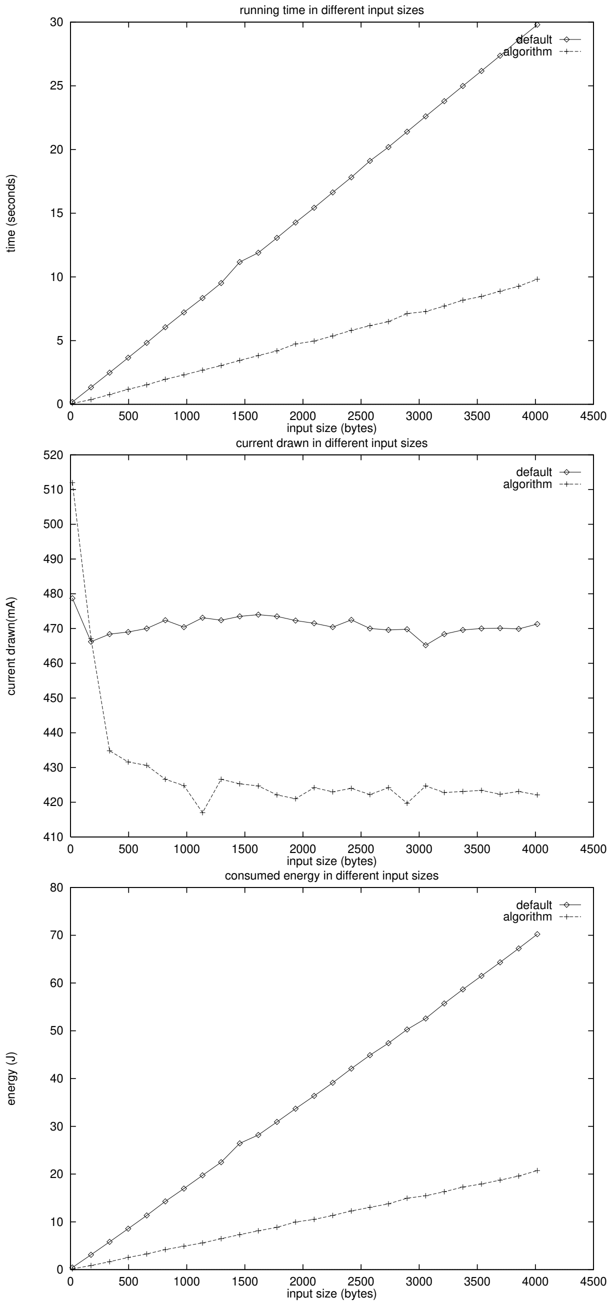


Figure 6: Results of G.721/encode with different input sizes

[3] A.V Goldberg and R.E Tarjan. A new approach to the maximum-flow problem. *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 136–146, 1986.

[4] Ulrich Kermer, Jamey Hicks, and James M. Rehg. A compilation framework for power and energy management on mobile computers . *14th International Workshop on Parallel Computing (LCPC'01)*, August 2001.

[5] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: A partition scheme. *International Conference on Compiler, Architecture and Synthesis for Embeded Systems*, 2001.

[6] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.

[7] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.