Energy Management for Mobile Devices through Computation Outsourcing within Pervasive Smart Spaces

Ahmed A. ABUKMAIL and Abdelsalam (Sumi) HELAL

Abstract— **In this work we explore the opportunity Pervasive Spaces could provide as supplemental energy sources. We utilize the nature of pervasive smart spaces to outsource computation that would normally be performed on a mobile device to a surrogate server within the smart space. The decision to outsource a computation depends on whether its energy cost on the device is larger than the cost of communicating its data to the surrogate and receiving the results back. We propose an approach by which the outsourcing decision is made at runtime, while the intelligence that makes that decision is inserted at compile-time as logic that modifies the application code. The merit of our approach is that it is application-independent and requires minimal programmer energy awareness. We utilized a methodology from real-time systems to aid us in constructing the decision making logic. Additionally, we implemented a runtime support on top of Linux to facilitate for testing and experimenting with the client/server outsourcing approach. Our experimental validation and benchmarks shows significant energy saving on the mobile device, which validates our approach as a viable and novel approach to power saving and management for mobile devices.**

Index Terms—**Computation Outsourcing, Pervasive Computing, Power-aware Computing, Mobile Computing, Smart Spaces.**

I. INTRODUCTION

The emergence of mobile and then pervasive computing (as new computing domains) introduced several new challenges and research opportunities, one of which was energy management. These challenges arose from the mobility of used hardware [30]. Such hardware includes devices such as cellular phones, PDA's, laptop computers, and even MP3 players. The mobility of these devices implies that they are powered by mobile power sources represented by the battery of each device; and that, in turn, implies that the power source is limited.

As these devices become more popular, and their use becomes more apparent and frequent, the need to manage their energy consumption becomes more vital to their operation. This is because

Manuscript received January 28, 2007. (Write the date on which you submitted your paper for review.)

Ahmed Abukmail is Assistant Professor in the School of Computing at University of Southern Mississippi, Hattiesburg, MS 39406, USA (e-mail: ahmed@cise.ufl.edu).

Abdelsalam (Sumi) Helal is Professor at the Computer and Information Science and Engineering Department at the University of Florida, Gainesville, FL 32611, USA (e-mail: helal@cise.ufl.edu).

the more often a battery needs to be charged, the more often the mobile device is rendered immobile (which reduces the pervasiveness of their applications). The problem of energy management has gained a lot of attention in the mobile and pervasive computing research community. This is due to the increased reliance on mobile devices by a wide spectrum of users. This increased reliance on mobile devices stemmed from the increased capability of these devices. This argument was supported by Helal [18] who gave a close look at the market for Java-enabled phones and PDAs from a commercial standpoint to show that the capabilities of these devices are increasing, and will continue to increase over time.

Starner [32] gave a discussion of how much slower advances in battery technology have been than those for the other mobile computer components (the discussion was given for laptop computer, but information for wearable computers, PDAs and cellular phones was deduced to be similar). Starner [32] provided a graph representing the improvement in laptop technology from 1990-2001. As the graph indicated, CPU speed has kept up with Moore's Law, but battery capacity has not. As a matter of fact, battery capacity improvements were extremely small.

The continued increase in reliance and capability of mobile devices indicate that the energy-saving problem is ongoing, and it needs to continue to be addressed on the long run. Due to their capability, mobile device users are ranging from teenagers to the elderly and they span a wide range of backgrounds and mobile device utilization. One of the most widely used feature set of mobile devices is that dealing with multimedia applications, especially among the youth. The young generation uses these devices to play video games; take and edit pictures and videos and record and play sounds and music on them. In addition to this, the medical and dental professions rely heavily on 3-D, and having these capabilities on mobile devices would be

attractive to them. Other applications would also include graphics design, and voice recognition.

Managing the energy consumed by these mobile devices has been an important subject in research and industry communities of both mobile and the pervasive computing. Solutions have been presented at the various levels and layers of the computer system, and often these solutions to the energy problem involve a certain type of tradeoff. One of the most attractive avenues to energy management is high-level energy-management techniques. Such a good argument was made for handling power management at high level that Ellis [8] proposed a power-based API to allow for synergy between the application and the system. One of the most attractive avenues to energy management are high-level energy-management techniques. Such a good argument was made for handling power management at high level that Ellis [8] proposed a power-based API to allow for synergy between the application and the system. One of the most attractive high-level solutions to energy management is a compiler-based solution that alleviates or minimizes the need for programmer power-awareness. This is done via compiler optimization. Velluri et al. [36] studied he effect of the traditional compiler optimization techniques on system power (and therefore energy). Results showed that (except for loop unrolling and function inlining) most optimizations increased the energy consumed by the core of the processor. These results were (at least for loop unrolling) confirmed by Kandemir et al. [20].

II. Overview of the Approach

To solve the problem stated in the previous section, the solution has to be composed of two parts. The first part of the solution is done at compile-time as an optimization technique at the high-level source code. The second part of the solution must provide the necessary support to the

outcome of the first phase. This is due to the fact that the outcome resulting from the compile-time phase is a different formation than that initially developed.

A. Overview of the Compile-Time Solution

We have introduced this work as part of our fine-grain approach to power-aware computing [2]. First at compile-time, an assumption has been made that the source code has been tested and verified in its original form. Although that is done, this solution still validates the source code syntactically to make sure that no inadvertent errors were introduced along the way. In addition to syntax checking, the source code is also disassembled and the outcome of this process is an assembly representation using the mnemonic representation of each instruction of the target architecture. At this point, information about the high-level source code and the low-level instructions will become available for the optimization technique part of this contributed research.

The next step is to recognize basic program blocks (mainly loops) in both the source code and the assembly code, and simply match them. Recognizing loops at the high-level representation of the source code will result in the ability to collect all the data involved in the computation of the loop, and that will yield the energy cost of communication for sending all the data involved in the calculation out, and receiving only the data that changes (L-Values). Also, using the technique mentioned by Healy et al. [17], the number of iterations for each loop is calculated. As for the assembly code, the loops are recognized to determine the instructions involved in each loop, which will yield the entire energy cost of executing a single iteration of the each loop. In addition to instructions, at compile-time, we recognized whole library functions such as those belonging to the math library, and we added the value of their energy cost to the cost of the loop in which

they are executed. This, along with the metric calculated before to find out the number of each loop's iteration construct a good estimate of the total cost of the local execution of each loop. Before calculating the total cost of communication and the total cost of each loop's computation, experiments were done to find out the cost of communicating a single unit of data (a byte), and the cost for executing each machine instruction for the target architecture. As for calculating the cost of communicating a single byte, a client/server application was tested with multiple sizes of data to communicate between two machines, and the measurement for this was recorded and averaged. As far as each instruction's energy cost, a similar approach to that presented by Tiwari et al. [35] was utilized where each supported instruction is isolated via high-level code implementation, and executed multiple times within a loop and the final result is averaged based on the number of instructions used (we used 100 instructions within a loop executing 100 million times). In addition to testing machine instructions and verifying their cost, we tested pre-existing library function and verified their energy cost in a similar manner to the individual machine instructions.

B. Overview of the Runtime Support

To support the ability to outsource code, the application must be able to run in one of two modes: normal mode, or energy-saving mode. So, when an application starts, it will have to get some information based on the resources that are available. If the battery is susceptible to be drained quickly, then the application needs to run in energy-saving mode, the user also has control over this. However, if the user decides to run in normal mode, then the application should not worry about computation outsourcing.

In order for the application to be able to make the right decision, it has to contact the battery monitor at startup. The battery monitor would have already determined if energy saving is

available via outsourcing (this decisions is based on user preference also). Additionally, the battery monitor will contact the network monitor to check if the devices is actually connected to a network and that network contains surrogate servers. If so, then it will run in energy-saving mode listing the appropriate surrogate available for the application to utilize. This monitor is also similar to, but much simpler than, those discussed by Flinn and Satyanarayanan [9] and by Gu et al. [15].

The work done by Flinn et al. [9] suggests that the cost of these monitors is "non-negligible". This is true in their case, as a lot of the intelligence to execute code remotely is done at runtime as opposed to compile-time, and that is why their approach is a coarse-grained approach to energy management. However in our approach, while may utilize an idea presented by Flinn and Satyanarayanan [9] and by Gu et al. [15], the solution is much simpler and that is because the battery monitor is a straightforward inquiry to operating system's advance power management (APM). As far as the network monitor is concerned, it will only be invoked if an energy-saving mode of operation is decided (mainly as an outcome of the battery monitor). Therefore, the cost is negligible for these two monitors. Implementation of the battery monitor was as easy as looking at a single file containing information about the battery at certain increments of time. As for the network monitor, several approaches can be investigated, the simplest of which was proposed by Gu et al. [15] and it is based on wireless broadcast for discovering surrogates.

III. Computation Outsourcing Framework

Outsourcing computation is not a new terminology here. However, the motivation behind outsourcing the computation to a remote server, and the approach under which we are outsourcing the computation is the contribution here. Our goal from this research is to show that

an intelligent runtime decision can be made to decide if it is better to execute a section of code locally on the mobile device, or would it be more energy-beneficial to send its data to a remote server, and get the results back.

A. Overview

The overall framework for outsourcing is described in Figure 1. The idea is that a server machine accessible via a wireless network can serve as a surrogate server for a host of mobile devices such as handhelds, PDAs and laptop computers. This server at runtime will receive requests from client programs running on any of these devices for outsourcing code to the server.

The code that is in charge of making this decision is completely transparent to the programmer. All the programmer is required to do is to compile the code to optimize for energy. This will result in two version of the program being generated which the programmer will eventually have to compile and install. We believe that this is not a burden on the programmer in any way, and it is not a requirement for the programmer to have any knowledge of energy requirements/constraints.

Once an application is compiled, and two versions have been generated (a server version and a client version), and they are installed on their respective machines, the user can then execute a client application on the mobile device. This client application executes normally until it reaches a section of code that has been designated as outsource-able (having the potential for outsourcing), this is what we call the outsourcing candidate. Once this section is reached, then the intelligent code that was inserted at compile-time is executed to make the outsourcing decision. As a matter of fact, the candidate code will not be executed (locally or remotely) until the decision making code is executed.

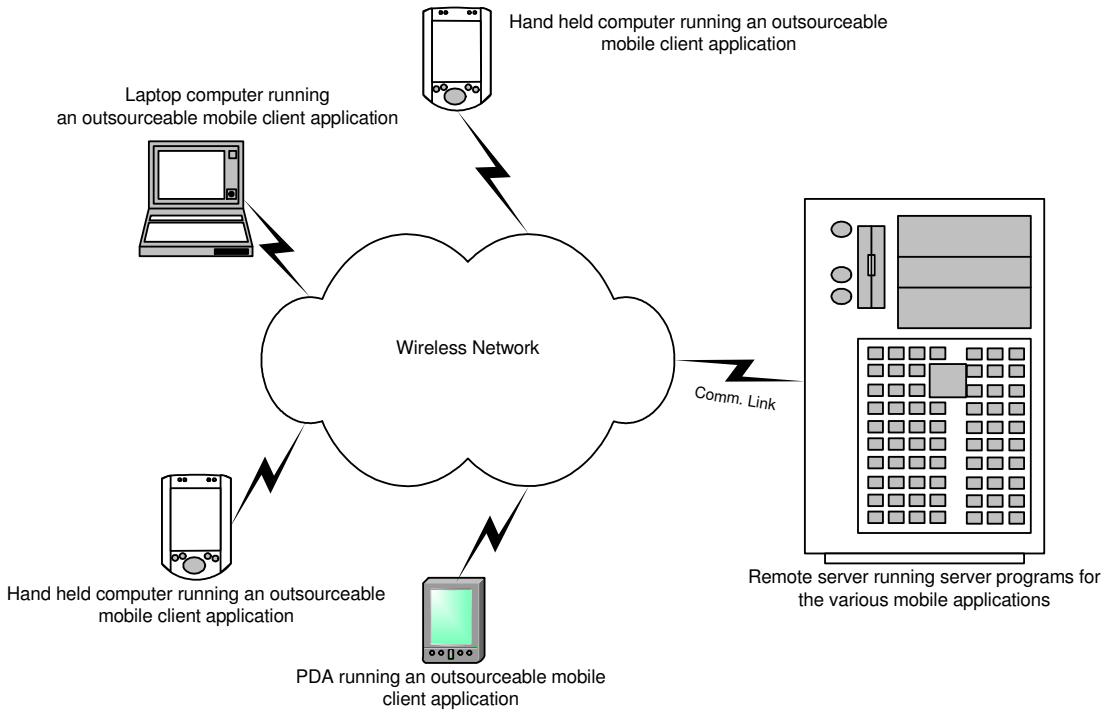

Figure 1. Framework for computation outsourcing at runtime

In figure 2, an illustration of the outsourcing mechanism at runtime is given. The client runs on the mobile device, and once it reaches an outsource-able section of code, it determines if it is more energy beneficial to outsource or is it more beneficial to execute locally. If the determination is made to outsource, then it will send the data to the server and wait for the results back, otherwise it will continue to execute locally until it reaches the next available outsource-able section of code. At all times, the server running on the surrogate machine is waiting for requests from client programs. Once it services the client's request it goes back to waiting for client requests again, which occur once a candidate section of code decides to outsource its computation.

The decision to outsource a section of code is not an arbitrary decision. The mobile machine user must configure it to determine if outsourcing is desirable in the first place. Therefore, there is a battery and a network monitor running on the client machine that will help in making this

determination. The battery monitor will run and ask the user to determine the outsourcing policy that the user chooses. Once that is determined, then the network monitor gets involved to determine the feasibility of outsourcing (if there is no network connection to a surrogate, then outsourcing will not occur. Once the feasibility is determined, applications either run in energy-saving mode, or they will run in normal mode.

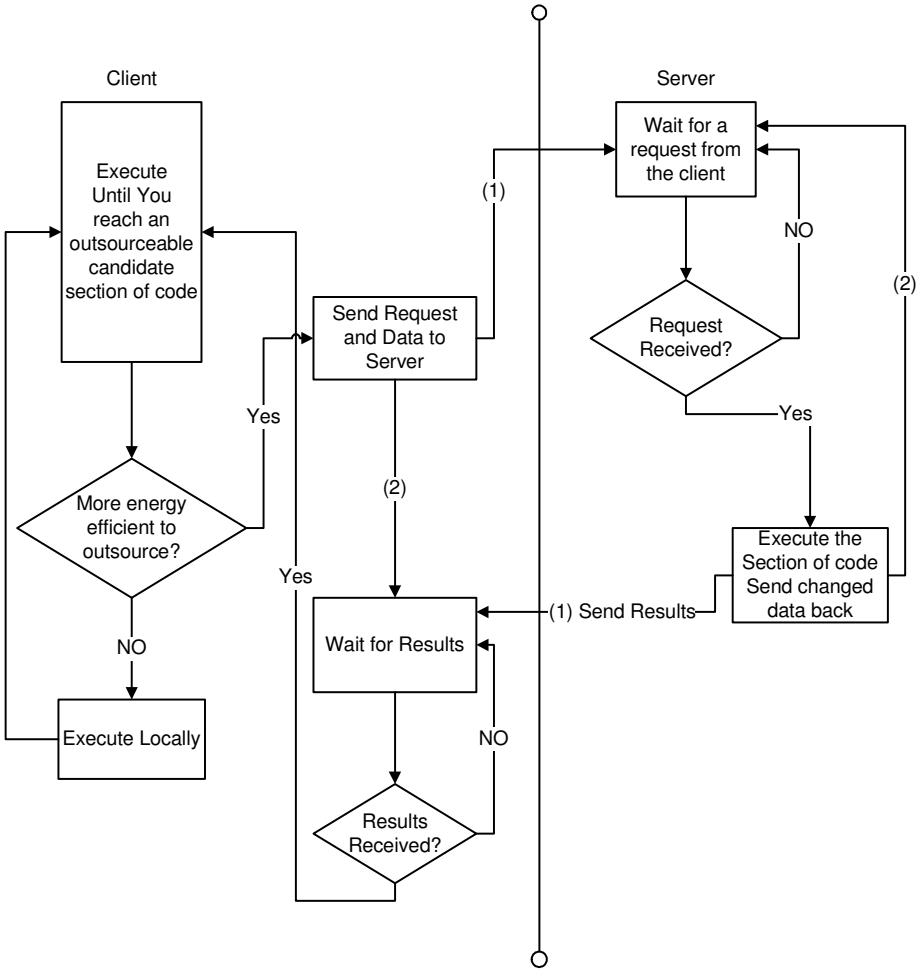

Figure 2. Steps for executing a client program under the outsourcing framework

Using our model, we envision the development and creation of an entity called a computation service provider (CSP). Different mobile users would subscribe to the CSP in order to service their energy-needs. The subscription will be by registering a copy of the server of the energy-aware application with the CSP. Whenever the user is within the proximity of (in the pervasive

smart space containing) the surrogate machine containing the server code, it would be possible for the client to outsource code to the server located on the surrogate. The outsourcing takes place by the client communicating its data to the server, let the server process the data, and then the client will get the results back.

B. Formal Model

When dealing with research that targets computation outsourcing via a distributed computing system, you have to consider grid computing as it is a new model that has a great potential in benefiting this type of research. In looking into grid computing [12] and [23], it looks like it would be a good model for our system on a much smaller scale, and therefore we defined our formal model to be based on one that was presented in the grid computing. Nemeth and Sunderam [26] presented a formal approach for defining the functionality of a grid system. Their approach started by defining distributed systems and showed how a grid system differs from the classic distributed system environment. Our model is a much more simplistic model than that they presented. Our model has a limited number of resources, and a limited number of processes. The resources in our model are the wireless network (WiFi) and the surrogate device. Our two processes are represented by the client and by the server versions of the original code. The model presented is based on an abstract state machine (ASM).

In looking at their model, we realized that their model encompasses a general description of grid and distributed systems. Our model is a simplified representation of theirs. In our model, we define the process universe as *PROCESS = {client, server}*, the resource universe as *RESOURCE = {wireless_net, surrogate}*, and the location universe as *LOCATION = {within-range, out-of-range}*. We use the same functions used in the grid and distributed computing domain, and add two of our own functions which are: *execCost: TASK → VALUE, and comCost: TASK →*

VALUE, where execCost is a function that produces the value of the energy consumed by a specific task of a process. Similarly, the *comCost* produces the value of the energy consumed by communicating the data for a specific task of a process.

As far as the functions that we use from grid and distributed computing are concerned, we use the same exact definition presented by Nemeth and Sunderam [26]. The following functions are defined:

- *user: PROCESS → USER*
- *request: PROCESS × RESOURCE → {true, false}*
- *uses: PROCESS × RESOURCE → {true, false}*
- *loc: RESOURCE → LOCATION*
- *CanUse :USER × RESOURCE → {true, false}*
- *state: PROCESS → {running-normal, running-energy-saving, receive-waiting}*, we modified this function to fit our execution framework.
- *from: MESSAGE → PROCESS*
- *to: MESSAGE → PROCESS*
- *event: TASK → {req-res, send, receive, terminate}*

Upon defining the above functions, and universe sets, the rules for defining our system as a simplified grid computing system can clearly be defined. We present definitions of the rules system in figure 3.

IV. Compile-Time Strategy

Our compiler optimization technique for low energy analyzes a source program at the three different levels of representation (high, intermediate, and low). At the high-level, we collect information about the data involved in each loop. At the intermediate level we utilize an algorithm described by Healy et al. [17] to find out the number of loop iterations. The reason this is an intermediate level analysis is because they analyze the register transfer list (RTL) [6] representation of the source code. At the low level, we determine the machine instructions generated by an assembler to determine which instructions are getting executed within each loop.

All of the three levels of source code analysis are embedded in our algorithm.

A. Overview

This new compilation technique utilizes pre-existing utilities such as the gcc compiler and Metrowerks' assembler and compiler. We first pass the code that needs to be compiled to the gcc compiler to make sure that it is syntactically correct, once that is done, we remove the resulting machine code as it will not be needed. Then we pass the same source code through our

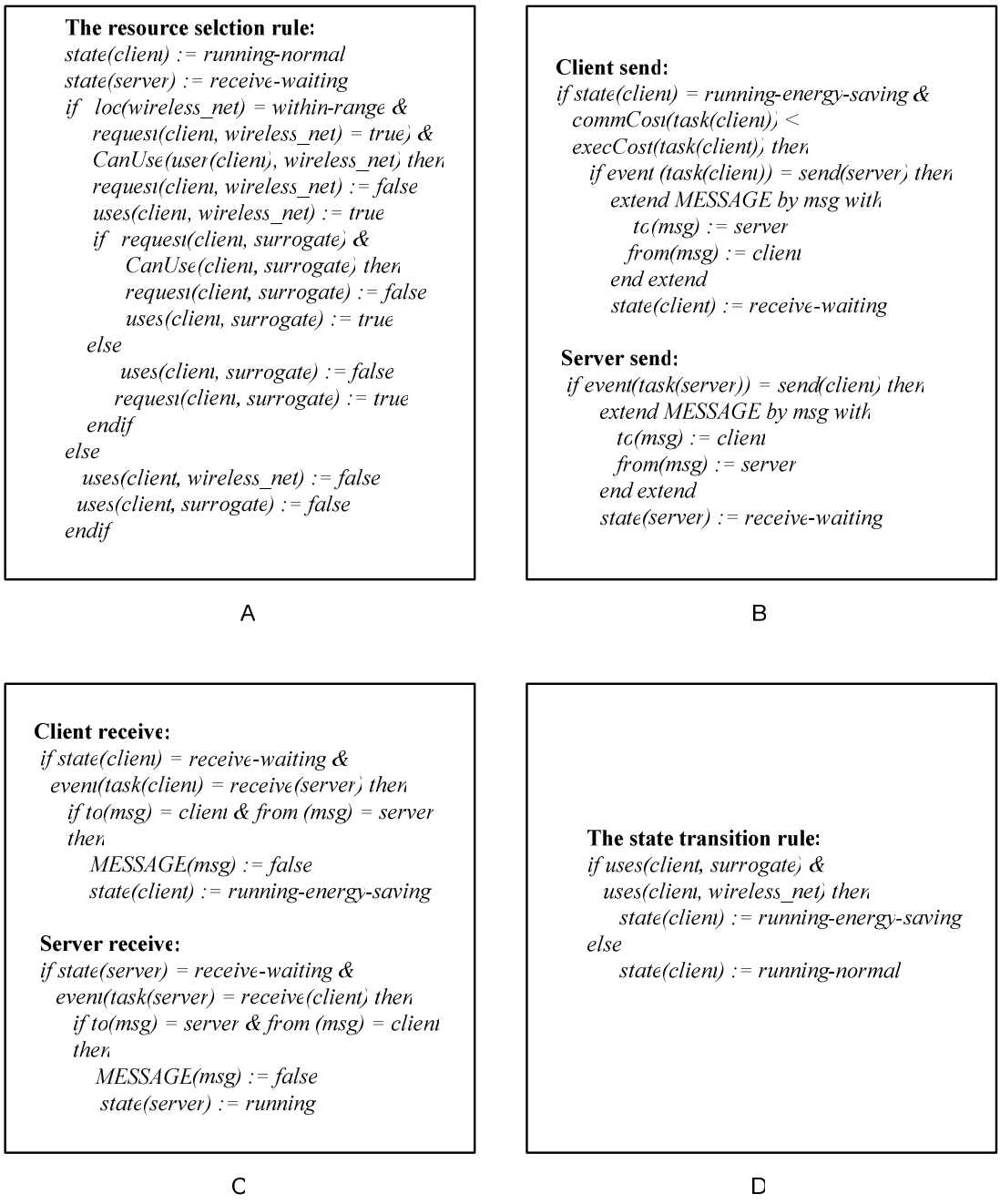

Figure 3. Rule definitions for the formal model. A) The resource selection. B) The send rules. C) The receive rule. D) The state transition rule.

optimization preprocessing, along with the assembly code generated from passing the original source code through the Metrowerks' assembler, to generate the two versions of the code, the client and the server. Once the client and the server codes are generated, then the server is compiled for the target server machine, and the client is compiled for the mobile device. In our

environment, the client is compiled using Metrowerks' Codewarrrior, and the server is compiled on a Linux machine using the gcc compiler. Figure 7 shows the process for optimizing a C program using our technique.

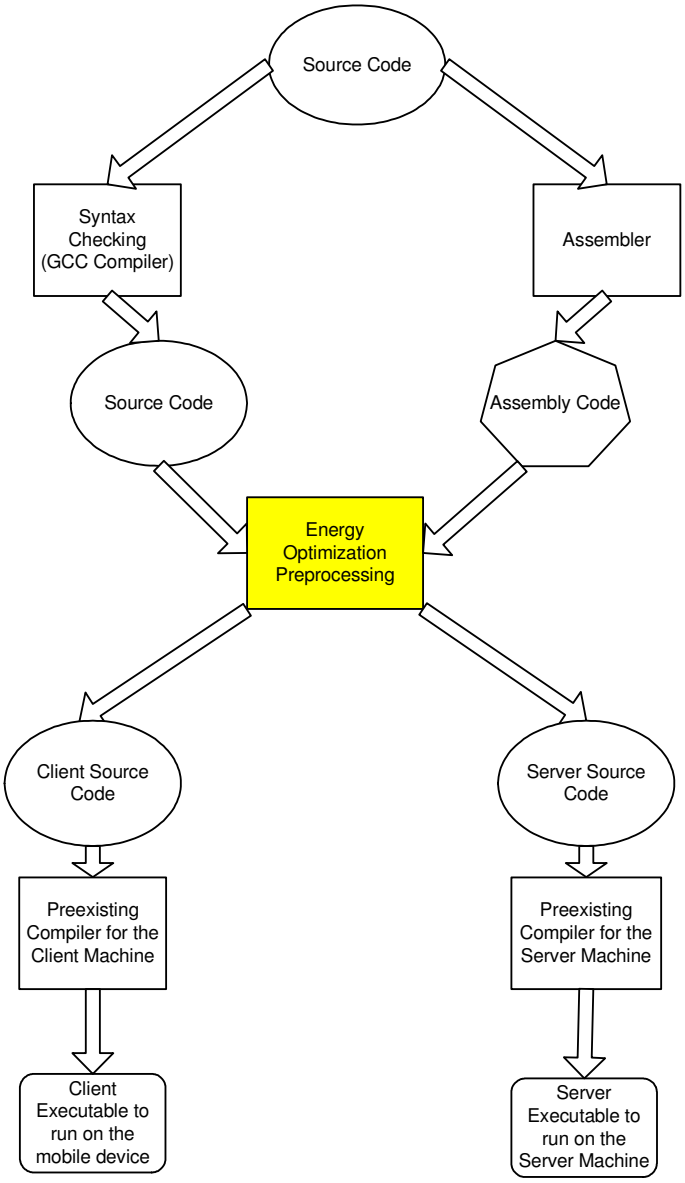

Figure 4. Overview of compilation and optimization process

B. Energy-Optimization Process

Our optimization technique modifies the high-level code (the C source code). The input to this process is a file containing the source code, and a file containing the assembly representation of

his source code. Using the source code program, we determine the number of loop iterations, the size of the data involved in the loop execution. Then we determine, using the assembly representation (low-level representation) of the program, the instructions involved, and we calculate the total energy cost for all of the instructions using the energy cost of each individual instruction. In addition to the machine instructions we handle also library functions (such as the math and standard libraries) called within each loop. The energy cost of each individual instruction was calculated using a methodology similar to that presented by Tiwari et al. [35], and we give an explanation of this in our experimental validation. We calculate the energy cost for library functions in a similar manner to that of the machine instructions. Also, we had already measured the cost of transferring one byte of data using our wireless card. Given all of these metrics, we were able to insert socket code within our source program and conditional statements to determine at runtime if it is more energy-beneficial to outsource a candidate section of code (basic program block/loop) or to execute it locally on the mobile device.

1) Calculating the Number of Loop Iterations

Healy et al. [17] developed a useful utility for predicting the worst-case execution time (WCET) of a program. They provided us with the software that will accomplish this task for C programs. Their algorithm is part of implementing a static timing analyzer for analyzing real-time systems, as predicting the number of loop iteration is essential for analyzing real-time systems. Their approach automatically bounds the number of loop iterations. They handle nested loops, and loops with multiple exits. Their methodology is implemented by analyzing the register transfer list (RTL) [6], which is an intermediate representation of the source program.

First, they identify the branches that can affect the number of times the loop executes. Secondly, they calculate when each branch can change its direction. Third, they determine when

each iteration branch can be reached. Finally, they calculate the minimum and maximum number of each loop's iteration. If the loop invariant is a non-constant, for the purposes of the timing analyzer they are implementing, they allowed the user to input the minimum and maximum values for this variable, and that is not needed for our compiler optimization technique, as our methodology supports non-constant loop invariant as, at runtime, its value will be known, and we can use a formula involving the invariant to be multiplied by the cost of executing each loop once (the energy cost) which gives us as a formula that is easily evaluated at runtime to determine if a section of code should be outsourced.

Their implementation is integrated in the implementation of the Very Portable C Compiler (vpcc) introduced by Benitez and Davidson [6]. The input to the modified vpcc (we will refer to it as vpcc) is a source program with a ".c" extension, and the output is a set of files, only one of which is of interest to us, and that is the file with the same name as the source program, except with a ".inf" extension (the INF file). The INF file contains information about the maximum and the minimum number of loop iteration, and we are only interested in the maximum number of iteration in our research and that is because we do not want to under-estimate, we want to outsource with a high degree of certainty that a benefit will be gained from outsourcing. In figure 5, we give a sample C program that can be compiled with vpcc, and by passing certain switches to it an INF file (Figure 6) will be generated.

2) Loop Data and Iterations Acquisition

The first stage of our technique is to recognize the maximal basic program blocks (most likely these blocks will be loops). These basic program blocks (loops) will constitute the opportunity for optimization (candidate code for outsource-ability). Once these basic program blocks are recognized at the high-level, then we collect all the data elements associated with them, and

```
main()
{
  int i, j;

  for(i = 0; i < 100 - j ; i = i + 3) {;}

}
```

Figure 5. Example of C program passed as input to vpcc

```
-3
main
! loop 0 0 1 1 -1 -1 1 2 3 4 -1 4 -1
! loop 1 1 -4 r[10] 0 r[9] 3 s -2 (100-.1_j-2)/3 (100-.1_j-2)/3 -1 -1 3 -1 3 -1
! block 1 lines 5-5 preds -1 succs 2 4 -1
makes_unknown 3 -1
doms 1 -1
1 82 4 0 8 7 () 1024 7 (100) 8 4 (%o1)
1 90 4 0 8 4 (%o1) 8 4 (%o3) 8 4 (%o1)
1 90 7 1 1024 7 () 8 4 (%o1) 8 7 ()
1 62 4 2 2048 4 () 0 0 () 0 0 ()
1 82 4 0 8 7 () 1024 7 () 8 4 (%o2)
! block 2 lines 5-5 preds 1 -1 succs 3 -1
makes_unknown 3 -1
doms 1 2 -1
2 32 4 0 8 4 (%o2) 1024 7 (3) 8 4 (%o2)
! block 3 lines 5-5 preds 3 2 -1 succs 4 3 -1
doms 1 2 3 -1
3 90 4 1 8 4 (%o2) 8 4 (%o1) 8 7 ()
3 74 4 2 2048 4 () 0 0 () 0 0 ()
3 32 4 0 8 4 (%o2) 1024 7 (3) 8 4 (%o2)
! block 4 lines 5-5 preds 1 3 -1 succs -1
doms 1 4 -1
4 80 4 0 128 4 () 8 7 () 0 0 ()
4 15 4 0 0 0 () 0 0 () 0 0 ()
```

Figure 6. Resulting INF file for the program in Figure 5. The boldfaced expression represents the maximum number of loop iterations

determine the beginning and end file positions of these loops. Additionally, we pass the relevant sections of the original source code to the program that calculates the number of iterations for each loop.

The first part of this stage is to implement a parser-like module (we call it the pseudo-parser) to recognize basic program blocks, collect the data used within each loop, and identify what variables are R-valued (do not change), and what variables are L-valued (change). We did not

need to implement a full parser here as the syntax has already been checked before entering this stage of the algorithm. In addition to acquiring the data elements involved in the calculation of each loop, we also determine which C library functions have been called to determine the contribution of their energy cost to the execution of the loop. Additionally, we determine if certain loops are not outsource-able. All loops that involve Input/Output (I/O) routines are designated as non-outsource-able. This holds true also for those loops that include nested loops with I/O functions.

The second part of this stage is to figure out the number of loop iterations and associate each number with each loop calculated by the pseudo-parser. The algorithm to do this is a very simple one. This algorithm is implemented using a very simple parser that parses only the lines that contain the minimum and maximum iterations for each loop in the INF file. Once it extracts the expression representing the maximum number of iterations, it cleans it up by removing any extra characters such as those in figure 6 where the expression is *"(100-.1_j-2)/3"*. This particular expression is unique also as it contains a *'/'* which could be problematic as if everything used in the expression is an integer then at runtime, integer division might happen, and therefore after we remove the substring *".1_"* from the expression and if we recognize the division operator, we insert the typecasting *"(double)"* right before it. Hence, the resulting expression is *"(100 – j – 2) / (double)3"*.

3) Calculating the Size of Loop Data

The next stage is to calculate the data size for each loop. In this stage, we examine each variable involved in the loop, and based on the size of the variable in bytes (including arrays), we add the value to our sum to calculate the size in bytes. Additionally, if the variable is an L-value (changes), then we add the size of the variable to our sum for the L-valued variable. This is a

very important aspect of this algorithm, as if the data does not change, we only need to communicate it to the server, and we do not need it back, but if it changes, then we will expect it to be sent back to the mobile device. This way, we can minimize the amount of communication needed. Additionally, we check if the loop contains other loops, and if so, then we collect the variables of the nested loops only if these variables have not already been collected by a parent loop.

At this point, we have the data size for each loop which when multiplied by the cost to send one byte of data gives us the total cost to send all the data within the loop added to the cost to receive all the L-valued data within the same loop.

4) Identifying Loop Instructions and Total Loop Execution Cost

Using the assembly code representation of the source program, we can recognize loops within the assembly code. The target architecture (Xscale) has a unique way of identifying loops. Loops can be identified by three consecutive instructions, the first of which is the compare instruction *"cmp"*, followed by a conditional branch *"ble, blt, bge, bgt, bne, beq"*, followed by an unconditional branch. *"b"*. The unconditional branch is quite useful in this regard as it sends the control outside of the loop, and all we have to do is to go to that branch location, and find the other unconditional branch that completes the loop and returns us back two instructions before the *"cmp"*. This way we are able to identify or rather delimit where the assembly code for each loop starts and where it end. However, when loops are nested, we need more information to be able to map loops at the assembly level with those at the high level. The additional information needed is available in the structure containing information about all the loops (we call it the "*loopdata"* data structure). The information needed here is which loop is nested with which loop, and that information was obtained via our pseudo-parser.

Once each loop was delimited, then it was just a matter of going through the instructions that constitute the loop, and adding their pre-measured energy cost. In addition to the cost of each instruction there is a cost for pipeline stalls. This cost was obtained experimentally using multiple instruction sequences once the cost per instruction was determined. Therefore, when we recognize that certain instructions precede others (e.g., *str* before an *ldr*), we add the measured pipeline stall energy cost. This calculation gave us the cost of a single execution of the loop. At this point, we have all what we need to be able to produce the resulting client and server. The total loop execution cost becomes a matter of multiplying the cost of a single execution by the formula representing the number of loop iterations calculated before.

5) Insert Outsourcing Code

The implementation of this code was very large, but it was not difficult. As our pseudo-parser generated for us information of where each loop begins and where it ends. The location of where we need to generate the necessary C code to create a client/server based application becomes a matter of inserting the necessary include files, and variable declaration (we declared them globally). The outsourcing code is only inserted for those loops that are flagged outsource-able.

V. Runtime Support

Here we present the two monitors used as runtime support for the outsourcing mechanism. The two monitors work together and they get executed based on the user preference and the battery condition. The attractive property of these two monitors can be summed in the fact that most of the time they are not consuming any energy. In fact, they consume very little energy when they are doing any work. The way these two monitors work depends on the user preference in the first place, and once that has been determined, the condition of the battery of the mobile device takes

control of the decision making process. These two monitors will run on the client mobile device. In addition to these two monitors, a server program will run on the surrogate device waiting for connections from the client. The battery monitor, network monitor, and the server will together establish the service detection within the wireless network.

A. Battery Monitor

The battery monitor gets executed either by the user or the operating system. This is also a decision to be configured by the user. The user chooses if he/she desires to run in energy-saving mode or in normal mode. If normal mode is selected, then nothing happens and the monitor exits. Otherwise, if energy-saving mode is selected, then the battery monitor will ask the user if the energy saving is to take place immediately, or it should wait until the battery gets below a certain limit. If energy saving is to take place immediately, then the battery monitor will immediately call the network monitor. Otherwise, the battery monitor will sleep (consuming a very negligible amount of energy) and periodically check the status of the battery by contacting the operating system. Contacting the operating system is a very trivial matter as it will only look at a file called "/proc/apm", and extract the remaining percentage of the battery. Once it reaches the limit specified by the user, then will contact the network monitor that will complete the task of setting up the device in an energy-saving mode.

B. Network Monitor and Surrogate Service Discovery Server

Once the network monitor is called, it will send out a broadcast that will be only received by a network server that is providing any service for the client. This server will be running on the surrogate machine that is to service the client. Once the server receives the broadcast it then will establish a handshake with the client device and inform the device of its name, and that it is ready

for servicing the device and it supports outsourcing. At that point, the network monitor will create a configuration file that is to be opened by the application to determine if it would run in energy-saving mode, or normal mode. If any type of error occurs on the way to creating this file, the file will not be created, and hence there will be no energy-saving mode.

At runtime the client application will start running and checks if the energy-saving file exists, and if so, then extract the information about the server from it, establish the connection, and execute in energy-saving mode, otherwise, execute in normal mode.

VI. Experimental Validation

Our measurements, and experiments were done in two stages with our platform setup. The first stage was to estimate as accurately as possible the cost of each supported machine instruction (assembly instruction). Secondly, the second stage is to measure the cost of each benchmark, first without our optimization, and second with our optimization.

A. Setup

Our target architecture is an Intel Xscale which is an integral part of the Intel PCA. We chose the Sharp Zaurus SL-5600, which contains an Intel Xscale PXA-250 processor, and is running Linux, as our mobile device. Installed on the Zaurus, is a Socket's low-power wireless LAN card. The outsourcing server is an Intel x86 machine running RedHat Linux 7.2.

Developing applications on the Zaurus was achieved using Metrowerks Codewarrior for the Sharp Zaurus. This software comes with a packaged executable to run on the mobile device only during development to be able to debug and/or execute the application on the Zaurus from a Microsoft Windows where Codewarrior is installed. The software is called MetroTRK (Target Resident Kernel). We use MetroTRK to transfer executables to the Zaurus via our wireless

network at the Harris Mobile Computing Laboratory at the University of Florida (http://www.harris.cise.ufl.edu). This Microsoft Windows machine on which Codewarrior is installed also happened to be the same machine on which we record our measurements.

For measuring the energy consumed, we used an Agilent 34401A multi-meter which was connected to a Microsoft Windows 2000 desktop computer via an IEEE488.1 General Purpose Interface Bus (GPIB) cable. Installed on the desktop is Agilent's Intuilink plugin which works with Microsoft Word and Excel. We used Excel because its plugin allows for multiple readings as opposed to Word's single reading. The multi-meter has two J-hooks that were placed in series between the AC adapter and the Zaurus to place the multi-meter in series to measure the total current drawn by the Zaurus. The voltage coming from the AC adapter remained at a constant 5 volts. Therefore, the only for as two factors in the energy equation are the current drawn and the time in seconds as energy is given by the equation:

$$E = V * I * T$$

Where E is the energy consumed, V is the voltage, I is the current, and T is the elapsed time.

To measure the energy consumed by either a running process on the Zaurus, or by data communication, we calculate the difference between the current drawn when the Zaurus is idle, and when the Zaurus is either running a process or sending/receiving data. To make this as accurate as possible, the only application that we ran on the Zaurus was the Linux Terminal. We also turned off the light of the Zaurus LCD. These measures that we took, and as our experiments show, resulted in a constant current drawn by the Zaurus while idle, which gave us the ability to get good measurement with as little sampling noise as possible. The multi-meter only allowed us to take samples at one tenth of a second. Therefore, we had to execute code that consumes enough time to allow accurate a measurement as can be obtained.

B. Instruction-Level Energy Cost Estimation

Before applying our optimization to source code, and besides knowing what machine instructions were used, we had to know the energy cost for each machine instruction involved. Our work targeted only a subset of instructions from the Xscale architecture, which was sufficient to testing our benchmarks, and by no means is that a limitation of our approach.

We used a methodology similar to that described by Tiwari et al. [35] to estimate the cost of each instruction. The methodology suggested executing several instances of a single machine instruction within a loop and average the energy consumed to obtain the per instruction cost. Figure 7 shows a small C programs which when executed gives a good estimate of the energy consumed by an empty for loop. Figure 8 shows a second C program that gives a good estimate for the load instruction (LDR), which loads a memory location into a register. Notice, to minimize the affect of the instructions that execute the loop header (as the loop body is the several occurrences of the load instruction) we used a register loop control variable. The ability to declare register variable in a language like C made it very possible for us to be able to isolate single instructions which enhanced the accuracy of our estimation.

In order to minimize the effects of the cache, and to better estimate inter-instruction energy consumption, we developed additional benchmarks that include multiple instances of a series of different instructions (e.g. ldr followed by str) and we observed their behavior, and recording the amount of additional energy consumption recorded, and used these metrics in our code to update the instruction-level energy consumption values calculated before. In addition to minimizing the effect of caching the instructions, our benchmarking of each instruction, also, estimated the effect of the pipeline data hazards involved in the execution of each instruction.

C. Measuring Communication Cost

To measure communication cost per byte, we wrote small UNIX socket programs that sent data back and forth to a server, and our results confirmed the card statistics that were mentioned on the datasheet of our wireless LAN card. The data sheet suggests that this card is active 90% of the time transmitting at 265 *mA* and receiving at 170 *mA*.

```
main()
{
  register int i;

  for(i = 0; i < 100000000; i++)
  { ;}
}
```

Figure 7. The C program used to estimate energy cost of an empty for loop (executing 100M times)

```
main()
{
  register int i;
  registerint x;
  int a, b, c, d, e, f, g, h, m, n;


  for(i = 0; i < 100000000; i++)
  {
    // repeat 10 time the following 10 statement
    // each line represents an ldr instruction
    x = a;      x = f;
    x = b;      x = g;
    x = c;      x = h;
    x = d;      x = m;
    x = e;      x = n;
  }
}
```

Figure 8. The C program used to estimate the energy cost for the ldr instruction

D. Simple Experimental Validation

To test the effect of our approach on energy, we implemented three different simple benchmarks that span three different formations of data and execution complexity. The first of

which was the Fibonacci loop, which contains constant data, but it executes in $O(n)$ time. In other words, the size of the data remains constant, while the execution changes with n, where n is the number to which we are trying to calculate the Fibonacci number. Due to the sampling limitation of our multi-meter, we had to test this 3 times using 3 large numbers to get more accurate results of our measurements. We performed the testing using the numbers, 100000, 200000, and 300000. Another benchmark that we used was a rectangular version of the bubble-sort loop which executes in $O(n^2)$ and the data size is linear. So, as the data size grows so will the computation complexity. We sorted 10000, 20000, and 30000 integers. The last benchmark that we used was a square matrix multiplication loop, which runs in $O(n^3)$ where n is the number or rows and the number of columns of each matrix. For matrix multiplication we used a 200x200, a 300x300, and 400x400 matrices.

Each one of the benchmarks was executed on the Zaurus before our optimization and after our optimization. We estimated the energy saving for the Fibonacci calculation and the matrix multiplication to be between 60% and 88%. However, one interesting observation in our testing was the bubble-sort loop. The smallest energy saving was 98% for sorting 10,000 integers. This result is well expected due to the fact that compared to the amount of computation involved in bubble-sort, the size of the data is very negligible.

E. Large-Scale Benchmark Validation

To show the benefits of our approach, a more realistic benchmark had to be developed to show that this approach has more meaningful and potentially industry-utilizable benefits. Some of the most computationally intensive computations are those involved in generating an image representing a 3-D graphics scene. To generate a 3-D graphics scene, the input and output of the program are extremely non-expensive processes, as even more complex scenes can be described

with a virtually small amount of data. Also, the output is always a 2-D Image. But to get from a 3-D description of the scene to a 2-D Image depicting the scene, a huge amount of calculation has to be done. In this benchmark, the size of the data is $O(n^2)$ and the order of the computation is also $O(n^2)$. However, the amount of constant calculation within each iteration, is huge when compared to the amount of communicating each unit of data involved in the computation. Our experimental results show a significant amount of energy saving for generating a scene by ray-tracing 3 spheres of different sizes and colors in space to generate 3 different images of 50x50, 100x100, and 200x200. Figure 9, shows the input data passed to the ray-tracing process and figure 10 shows the image produced.

The input to the ray tracing application is quite a simple input composed of a few floating point numbers. These numbers represent the description of the world composed of: the observer, the light source, and the parameters describing three spheres in space.

```
200 200
.1
200 200
4 4 4
.8 .8 .8
4 4 4
0 0 0
1 1000
5 5
3
0.1 0.1 0.1 0.3
.5 .4 .3
1 .8 1 10
0.5 0.5 0.8 0.2
.3 .4 .2
1 .8 1 10
0.2 0.8 0.5 0.1
.4 .3 .5
1 .8 1 10
```

Figure 9. Input file for the ray tracing application.

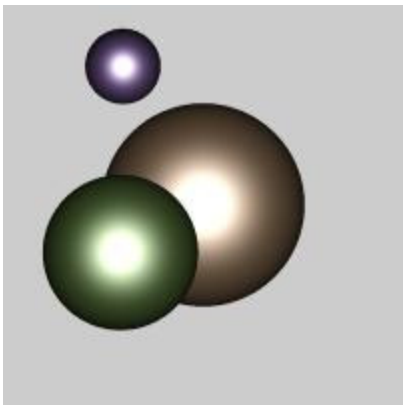

Figure 10. The 2-D image representing the 3-D scene generated by the ray tracing application for a 200x200 image.

We did our experiments on various sizes of data and we generated a 50x50 image, a 100x100 image, and a 200x200 image for the same scene. The amount of computation was so large that in all three cases, the computation was outsourced. Figure 11 shows the results of comparing local execution vs. remote execution for the 200x200 image.

VII. Related Work

Prolonging battery life (often called energy management) has long been the focus of research. This problem has many facets, which can be faced by addressing the various components of a mobile computer system. In [1] we authored a book chapter on power-management techniques. These techniques include reducing energy at the system-architecture level, by targeting (reducing) various components of the power equations $(P=CV^2f)$, where P is power, C is capacitance load, V is supply voltage, and f is switching frequency. Other techniques targeted the operating system by saving energy involved in communication, by caching, by process scheduling, and by having an energy manager. Additionally, we presented software techniques grouped into two categories: specific application techniques, and compiler-based techniques. These software techniques (like those targeting the operating system) are considered higher-level

power-management techniques. Solutions that target energy management usually involve some kind of a tradeoff.

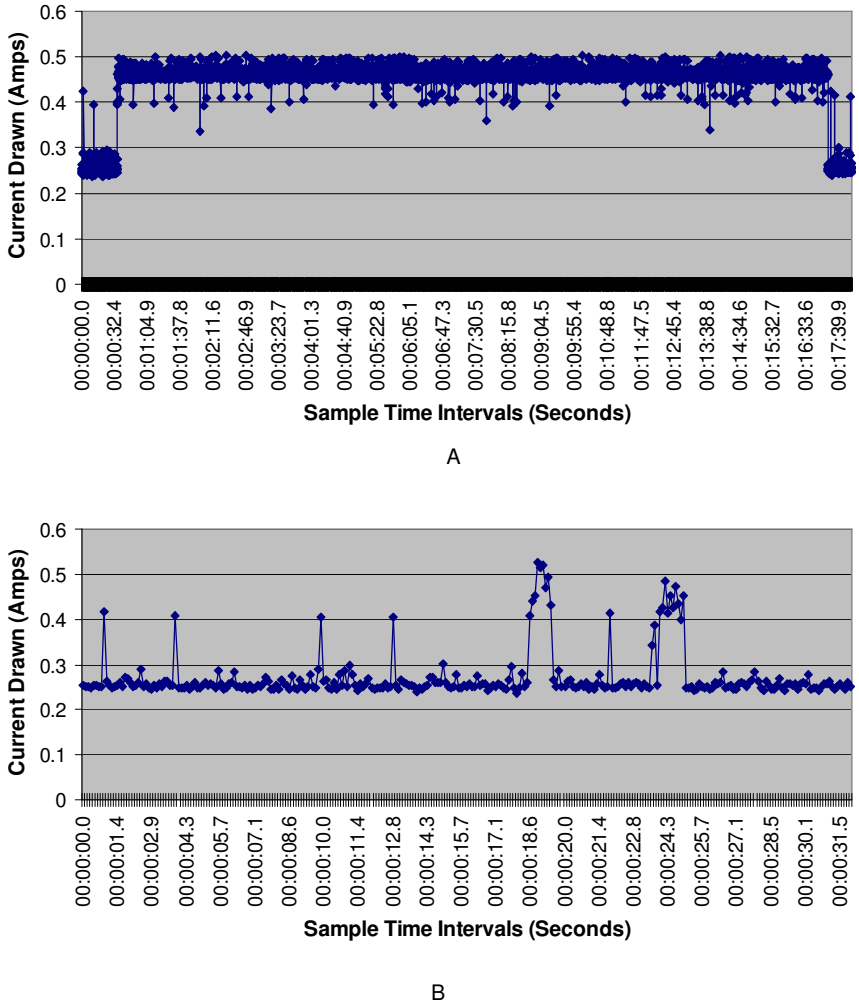

Figure 11. Local vs. remote execution for the ray tracing application of a 200x200 image. A) Histogram for current drawn during local execution. B) Histogram for current drawn during remote execution.

As far as energy management is concerned at the hardware and architecture levels is concerned, a few developments have been introduced. Smart battery systems were introduced to perform intelligent power drainage (http://www.sbs-forum.org/specs/index.html). In addition to batteries, energy-aware processors were also introduced. Various companies introduced their

solutions to in the form of energy-aware processors. Intel introduced the Xscale processor (http://www.intel.com/design/intelxscale), and earlier they shipped their Pentium III with the SpeedStep technology (http://www.intel.com/support/processors/mobile/pentiumiii/ss.htm). Transmeta Corporation has the Crusoe family of processors (http://www.transmeta.com/crusoe/index.html). Additionally, the ARM family of processors is widely popular, and is geared toward reducing power consumption while maintaining a high level of performance (http://www.arm.com).

Reducing any of the variables involved in the power equation will reduce the energy and power consumed. Capacitance load, frequency, and voltage can be managed at the hardware and architectural level. Voltage and frequency scaling have been targeted in [19]. However Smit and Havinga [31] argued that reducing voltage indicates reducing performance, therefore additional hardware is needed to balance it out. Capacitance load reduction was also targettedin [14], and [33]. Hardware solutions augmented by compiler support was also done in [4], and [37] by adding additional caches.

As for operating system solutions, advanced power management (APM), an more recently advanced configuration and power interface (API) have been quite useful in energy management. Additionally, secondary storage (disk) access is very expensive. Therefore, the lower the frequency of disk access is the better it is for energy. Therefore, making fewer incorrect file predictions is a good methodology to save energy [38]. Also, energy saving communication techniques are getting increasingly important. Managing communication devise was done in [21]. In [24] a solution was provided for application-level energy management that can be easily utilized also at the operating system level. Energy-aware scheduling via monitors was also introduced in [5].

The case for higher-level energy management was made by Ellis in [8]. Therefore addressing energy management at the high level is quite an attractive solution. The solutions for higher-level energy management include both application-based solutions, and compiler-based solutions.

Research and experiments have shown that, with the exception of loop unrolling, and function inlining, compiling for performance does not imply compiling for energy [36]. In [34] a few techniques for energy management were introduced. These techniques were proposed to target reduction in frequency of logical state transitions, reordering instructions by utilizing a power metric as opposed to the performance metric suggested in [3], and in [13]. Other compiler-based techniques were introduced in [25], and [29]. Also the work done in [27] migrates the compilation process to a server to save energy. In [22], the work that is closely related to our work was presented, where they perform remote task execution based on the cost of communicating the data. However the work targets specific tasks based on checkpoints that delimit the tasks.

Flinn and Satyanarayanan [10], demonstrated a collaborative relationship between operating systems and applications to meet user-specified goals for battery life. They used PowerScope [11] to validate the measurements of energy consumption for accurate estimation.

As far as the applications are concerned, Haid et al. [16] developed an excellent application with energy awareness in mind. This work presents designing an energy-aware MP3 player. Additionally, Yuan et al. [39] investigated another multimedia application with respect to power-awareness. They present a middleware framework for coordinating the adaptation of multimedia application to the hardware resources.

In addition to previously mentioned Powerscope [11], other research has been done to estimate energy for certain applications, systems, and devices. Cignetti et al. [7] described an energy

model for the Palm™.

VIII. Conclusion and Future Work

Our experimental evaluations showed that computation outsourcing within a pervasive computing smart space has a great potential in energy management. By exporting CPU processing into the network, the mobile device was able to deliver the expected functionality while consuming less energy and lasting for a longer period of time. Therefore, communication should not be viewed as a drain on the battery, but as an opportunity to save energy.

We found that the research done in the domain of real-time systems is quite useful as its aim is always knowing as much as possible about an application before runtime (e.g., compile-time). The reason for that in real-time systems is to execute programs to finish within a deadline. However we believe that the limitations of the utilized methodology which are represented in the inability to handle non-counter-based loops, such as those loops that are pointer-based, logical-expression based, and some non-rectangular nested loops, raise the need to investigate other computer science disciplines, specifically automated software verification.

While the work done in the area of automated verification for loop invariant generation is not quite mature enough, it has a great promise to be combined with the utilized real-time systems methodology to generate accurate estimates for the number of loop iterations. We will investigate the research done in this area to determine the extent of its applicability in our work. Specifically, we will look at the work done by Pasareanu and Visser [28]. We believe that, when combined with the methodology we utilized for determining the loop iterations, the area of loop invariant generation will be quite beneficial to our research.

As part of our future work, we will continue the implementation of the client/server version

that we have already started. However, this implementation will contain additional support for dynamic memory allocation and subroutine-level computation outsourcing. In addition, work needs to be done to evaluate a more accurate communication cost. This part will be an ongoing part of the research as we utilize additional outsourcing techniques. In addition, we will continue to identify specific applications that will benefit from our approach which will be a refinement of defining categories of application areas that we believe will benefit from this approach.

To handle multiple platforms, targeting languages such as Java and C++ will be necessary. This will require porting a compiler to handle both languages while augmenting with the code that we obtained to calculate the number of loop iterations. Additionally, as we will support additional types of basic program blocks we will build the compiler support that will handle estimating total execution cost of these basic program blocks such as additional types of loops than those supported so far, recursive functions (inherently these are loops), library linked functions whose energy consumption is predefined, and functions defined within the code. Additionally, we will investigate in the case of non-Java languages, the possibility to be able to cross-compile them for the most popular mobile devices.

In this research, we assumed that if a basic program block (loop) contained an I/O operation, it is determined as non-outsource-able. However, in our future work we will be looking at opportunities where I/O operations may be performed elsewhere. For input operations that involve files, if the file exists elsewhere, then it could be energy-beneficial to read the file on a remote machine, and utilize its contents remotely. Similarly, if producing the output elsewhere and basically all we are interested in is a display of this output which maybe cheaper than displaying the output locally, then that is another opportunity that needs to be investigated.

Identification of the applications, which contain the computationally expensive basic program

blocks, is an essential part of this research. We will investigate and research the different types of basic program blocks that fall into this category and test them and provide them as benchmarks for our research. In addition, we will build the support for other useful applications that will benefit from our approach.

Java remote method invocation (RMI), and remote procedure call (RPC) based systems will also be applicable in our research where we will let the system make the necessary data communication according to its policies and that will benefit our approach especially at the level of outsourcing specific functions, and subroutines. These two systems are utilized in grid computing environments.

We will weigh the benefits of every outsourcing mechanism with each type of basic program block we investigate, and investigate which mechanism allows us to save more energy with a basic program block. This will lead to a hybrid approach where a single application may contain one, two, or three outsourcing mechanisms.

Also, as far as service discovery is concerned, we will investigate a lighter version of some of the well-know service discovery systems like UPnP (http://www.upnp.org), and Jini (http://www.sun.com/software/jini). We believe that these systems can be utilized without spending a large amount of energy as the inclusion of the intelligence in the modified program.

REFERENCES

[1] A. Abukmail, A. Helal, Power Awareness and Management Techniques, in: M. Ilyas, I. Mahgoub (Eds.), Mobile Computing Handbook, CRC Press, Boca Raton, FL, 2004.

[2] A. Abukmail, A. Helal, A Pervasive Internet Approach to Fine-Grain Power-Aware Computing, in: IEEE/IPSJ International Symposium on Applications and the Internet, Phoenix, Arizona, January 2006.

[3] A. Aho, M. Ganapathi, S. Tjiang, Code Generation Using Tree Matching and Dynamic Programming, ACM Transactions on Programming Languages and Systems 11 (4) (1989) 491-516.

[4] N. Bellas, I. Hajj, C. Polychronopoulos, G. Stamoulis, Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High performance Microprocessors, in: International Symposium on Low Power Electronics and Design, Monterey, CA, February1998.

[5] F. Bellosa, The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems, in: 9th ACM SIGOPS European Workshop, Kolding, Denmark, September 2000.

[6] M. Benitez, J. Davidson, A Portable Global Optimizer and Linker, in: ACM Conference on Programming Language Design and Implementation, Atlanta, Georgia, July 1988.

[7] T. Cignetti, K. Komarov, C. Schlatter Ellis, Energy estimation tools for the Palm, in: 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Boston, MA, August 2000.

[8] C. Ellis, The Case for Higher Level Power Management, in: 7th Workshop on Hot Topics in Operating Systems, Rio Rico, AZ, March 1999.

[9] J. Flinn, S. Park, M. Satyanarayannan, Balancing Performance, Energy, and Quality in Pervasive Computing, in: 22nd International Conference on Distributed Computing Systems, Vienna, Austria, July 2002.

[10] J. Flinn, M. Satyanarayanan, Energy-aware adaptation for mobile applications, in: 17th ACM Symposium on Operating System Principles, Kiawah Island, SC, December 1999.

[11] J. Flinn, M. Satyanarayanan, PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications, in: 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999.

[12] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the Grid, in: T. Berma, G. Fox, A. Hey (Eds), Grid Computing: Making the Global Infrastructure a Reality, Wiley Series in Communication Networking and Distributed Systems, John Wiley and Sons Ltd, West Sussex, England, 2003.

[13] C. Fraser, D. Hanson, T. Proebsting, Engineering Efficient Code Generators using Tree Matching and Dynamic Programming, Technical Report No. CS-TR-386-92, Princeton University, August 1992.

[14] C. Gebotys, Low Energy Memory and Register Allocation Using Network Flow, in: 34th Conference on Design Automation, Anaheim, California, June 1997.

[15] X. Gu, A. Messer, I. Greenberg, D. Milojicic, K. Nahrstedt, Adaptive Offloading for Pervasive Computing, IEEE Pervasive Computing 3 (3) (2004) 66-73.

[16] J. Haid, W. Schogler, M. Manninger, Design of an Energy-Aware MP3-Player for Wearable Computing, in: Telecommunication and Mobile Computing Conference, Graz, Austria, March 2003.

[17] C. Healy, M. Sjödin, V. Rustagi, D. Whalley, Bounding Loop Iterations for Timing Analysis, in: IEEE Real-Time Technology and Applications Symposium, Denver, CO, June 1998.

[18] A. Helal, Pervasive Java Part II, IEEE Pervasive Computing 1 (2) (2002) 85-89.

[19] C-H. Hsu, U. Kremer, M. Hsiao, Compiler-Directed Dynamic Frequency and Voltage Scheduling, in: 1st International Workshop on Power-Aware Computer Systems, Cambridge, MA, November 2000.

[20] M. Kandemir, N. Vijaykrishnan, M. Irwin, W. Ye, Influence of compiler optimizations on system power, in: 37th Conference on Design Automation, Los Angeles, CA, June 2000.

[21] R. Kravets, P. Krishnan. Power Management Techniques for Mobile Communication, in: 4th ACM/IEEE International Conference on Mobile Computing and Networking, Dallas, Texas, October 1998.

[22] U. Kremer, J. Hicks, J. Rehg, Compiler-Directed Remote Task Execution for Power Management, in: Workshop on Compilers and Operating Systems for Low Power, Philadelphia, PA, October 2000.

[23] C. Lee, D. Talia, Grid programming models: current tools, issues and directions, in: T. Berma, G. Fox, A. Hey (Eds), Grid Computing: Making the Global Infrastructure a Reality, Wiley Series in Communication Networking and Distributed Systems, John Wiley and Sons Ltd, West Sussex, England, 2003.

[24] R. Loy, A. Helal, Active Mode Power Management in Mobile Devices, in: 5th World Multi-Conference on Systematics, Cybernetics, and Informatics, Orlando, FL, July 2001.

[25] D. Marculescu, Profile-Driven Code Execution for Low Power Dissipation, in: International Symposium on Low Power Electronics and Design, Rapallo, Italy, July 2000.

[26] Z. Nemeth, V. Sunderam, A Formal Framework for Defining Grid Systems, in: 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, Germany, May 2002.

[27] J. Palm, J. Eliot, B. Moss, When to use a compilation service?, in: ACM Joint Conference on Language Compilers and Tools for Embedded Systems and Software and Compilers for Embedded Systems, Berlin, Germany, June 2002.

[28] C. Pasareanu, W. Visser, Verification of Java Programs Using Symbolic Execution and Invariant Generation, in: 11th International SPIN Workshop on Model Checking of Software, Barcelona, Spain, April 2004.

[29] A. Rudenko, P. Reiher, G. Popek, G. Kuenning, The Remote Processing Framework for Portable Computer Power Saving, in: ACM Symposium on Applied Computing, San Antonio, TX, February 1999.

[30] M. Satyanarayanan, Pervasive Computing: Vision and Challenges, IEEE Personal Communication 8 (4) (2001) 10-17.

[31] G. Smit, P. Havinga, A Survey of energy saving techniques for mobile computers, Internal Technical Report, University of Twente, Enschede, Netherlands, 1997.

[32] T. Starner, Powerful Change Part 1: Batteries and Possible Alternatives for the Mobile Market, IEEE Pervasive Computing 2 (4) (2003) 86-88.

[33] C-L. Su, C-Y. Tsui, A. Despain, Low Power Architecture Design and Compilation Techniques for High-Performance Processors, Technical Report No. ACAL-TR-94-01, University of Southern California. February 1994.

[34] V. Tiwari, S. Malik, A. Wolfe, Compilation Techniques for Low Energy: An Overview, in: International Symposium on Low Power Electronics and Design, San Diego, CA, October 1994.

[35] V. Tiwari, S. Malik, A. Wolfe, T. Lee, Instruction Level Power Analysis and Optimization of Software, VLSI Signal Processing Systems 13 (2) (1996) 1-18.

[36] M. Velluri, L. John, Is Compiling for Performance == Compiling for Power?, in: 5th Annual Workshop on Interaction between Compilers and Computer Architecture, Monterrey, Mexico, January 2001.

[37] E. Witchel, S. Larsen, C. Ananian, K. Asanovic, Direct Addressed Caches for Reduced Power Consumption, in: 34th Annual International Symposium on Microarchitecture, Austin, Texas, December 2001.

[38] T. Yeh, D. Long, S. Brandt. Conserving Battery Energy through Making Fewer Incorrect File Predictions, in: IEEE Workshop on Power Management for Real-Time and Embedded Systems, Taipei, Taiwan, May 2001.

[39] W. Yuan, K. Nahrstedt, X. Gu, Coordinating Energy-Aware Adaptation of Multimedia Applications and Hardware Resources, in: 9th ACM Multimedia Middleware Workshop, Ottawa, Canada, October 2003.