Journal of Advance Computational Research
Vol. 1 No. 1 (January-June, 2016)

Energy Outsourcing for Mobile Devices in Pervasive Spaces

Ahmed Abukmail
School of Computing, University of Southern Mississippi
E-mail: ahmed.abukmail@usm.edu

Abdelsalam (Sumi) Helal
Computer and Information Science and Engineering Dept., University of Florida
E-mail: helal@cise.ufl.edu

Abstract: *We explore the opportunity Pervasive Spaces provide as supplemental energy sources surrounding visiting mobile devices. 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. Our experimental validation also shows that the capabilities of mobile devices can be enhanced to accommodate more computationally intensive applications to further realize the concept of a computation service provider (CSP).*

Keywords: *Computation Outsourcing, Pervasive Computing, Power-aware Computing, Mobile Computing, Smart Spaces*

1. INTRODUCTION

The emergence of mobile and pervasive computing introduced several new challenges and research opportunities, one of which is energy management. These challenges arose from the mobility of the used hardware [31]. Such hardware includes devices such as cellular phones, PDA's, laptop computers, and MP3 players. The mobility of these devices implies that they have a mobile power source represented by the battery which is a limited resource. As these devices continue to gain popularity, the need to manage their energy becomes even more vital. The more these devices have to be charged the more they are rendered immobile, hence reducing their pervasiveness.

The problem of energy management has gained and continues to gain attention in mobile and pervasive computing due to the increased reliance on mobile devices as a result of their increased capabilities. This argument was supported by Helal [19] by giving a closer look at the market of Java-enabled phones and PDAs from a commercial standpoint. Moreover, Starner [33] gave a discussion on how battery technology has not kept up with Moore's Law. The energy problem is an ongoing problem and must be addressed on the long run as usage of mobile devices is increasing among the young as well as the elderly and spans a wide range of backgrounds. One of the most used feature of mobile devices is their multimedia capabilities which are extremely energy (as well as computational power) intensive applications. Not only is this a concern with respect to entertainment, but it also is a concern in the medical as well as the dental professions as doctors now rely heavily on their mobile devices for various uses. Other applications that would benefit from energy saving are those that relate to graphics design, and voice recognition applications.

Solutions to the energy problem have been studied significantly. Solutions were presented at the various layers of a computer system. Often, these solutions have involved a certain tradeoff. At the high level, a good case was made for this solution by proposing power-based APIs [9]. Compiler based solutions, as an attractive solution, have also been proposed as they alleviate the need for programmer's energy awareness [21, 37].

In this work we devised a solution composed of two parts: a compile-time optimization of the high-level code, and a runtime support for the resulting optimized software. The result of the first phase is a client/server

version of the software where the client runs on the mobile device and the server runs on a surrogate machine. We utilized work from real time system research to help facilitate the compiler optimization phase. Our results showed significant savings in energy consumed by our benchmark applications.

2. COMPUTATION OUTSOURCING

We present a novel utilization of computation outsourcing in this work. We utilize pervasive smart spaces as well as the wireless network surrounding a mobile device to save energy. We initially would have compiled our code using our new compile-time energy optimization process, and generated a client and a server version of the software. We would have installed the server on a surrogate machine within the pervasive space, and the client would have been installed on the mobile device entering the pervasive space. Moreover, the mobile device has battery and network monitors running in the background to facilitate the client/server execution environment.

The mobile device would make the decision once entering the pervasive space whether to go into energy saving mode or normal mode. This policy can either be left to the mobile device to intelligently make the decision by pre-configuring the battery monitor, or it can be left up to the user to decide.

Once the device is placed in energy saving mode, the battery monitor would dictate to the network monitor to look for wireless networks within range, and if found it will go into energy saving mode, otherwise it will remain in normal mode and no outsourcing will take place. Once in energy saving mode, the applications compiled using our energy optimization process (pre-processor), would send requests and receive service from the surrogate server in the case that it is more energy-beneficial to do so, otherwise local application execution will take place. Figure 1 depicts the communication process that takes place between the client and the server. This work was introduced in [2].

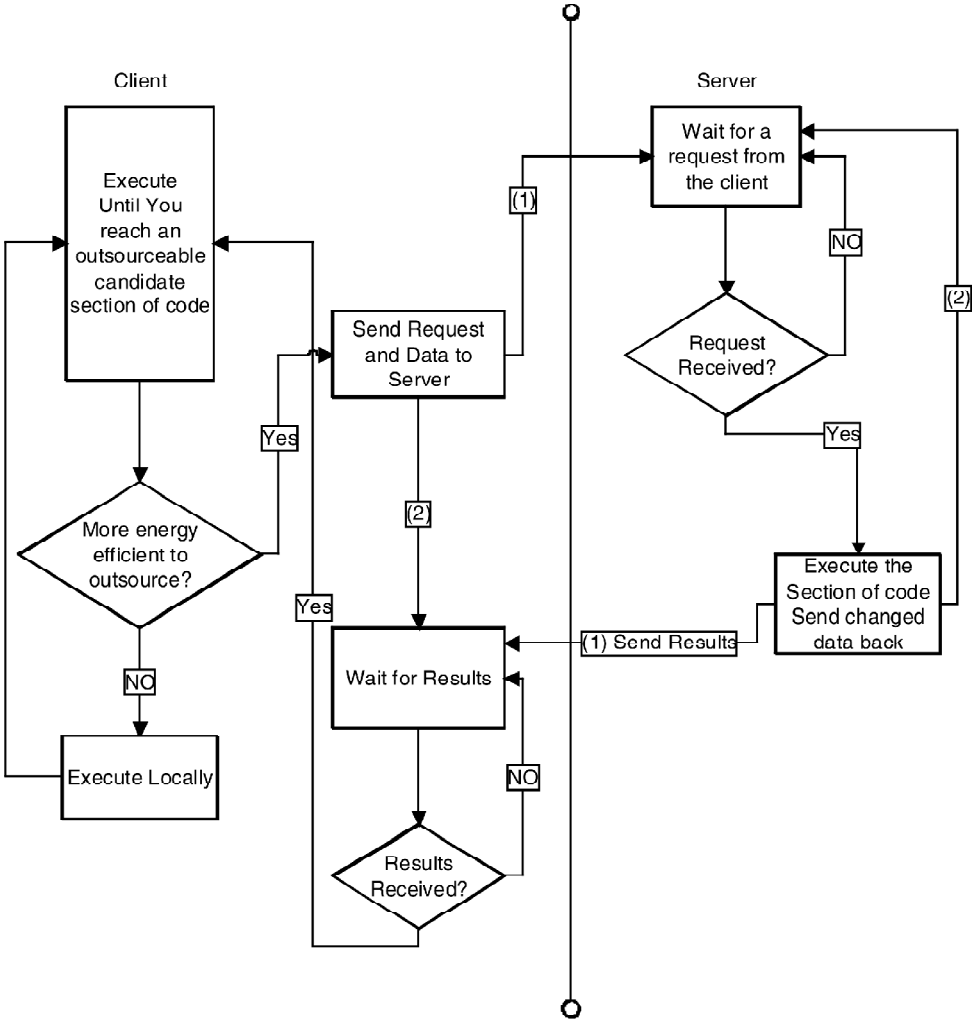

Figure 1: Client/Server Communication

Using our model, we envision the development of an entity describe as a computation service provider (CSP). Different mobile device users can subscribe to this service to facilitate energy savings on their mobile device. We also envision securing this service via setting subscription policies.

2.1 Formal Model for Computation Outsourcing

We present a formal model for computation outsourcing based on the research done by Nemeth and Sunderam [27] who presented a formal approach for defining the functionality of grid systems. We adopted their definition and simplified it for our purposes as grid and distributed computing [13, 24] can benefit a great deal from our research. The model they presented is based on an abstract state machine (ASM). However, the resources in our case consist of WiFi and surrogate servers.

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 [27]. 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 2.

3. COMPILE-TIME SOLUTION

Our compile-time approach presented in [2] is quite novel. We started the energy optimization process under the assumption that the program has been tested and verified in its original sequential form. The following steps are followed for our compile-time optimization:

(1) Verify the syntax of the code (*gcc* compiler)

(2) Produce the assembly code for the source program in mnemonic form.

(3) Use both the original program as well as the assembly code as input to the energy optimization process.

(4) The result will be two versions of the code (client, and server)

(5) Compile each version using its respective compiler on the target machine.

In steps 1, and 2 we used pre-existing compilers and assemblers to generate the required input to the later steps. We also used pre-existing compilers for step 5 of the process. Our major contribution in the optimization process was in step 3 which resulted in the generation of the two versions of the program mentioned in step 4.

Step 3 is divided into multiple subtasks, the first of which is to recognize basic program blocks (at this point we handle *for* loops). While recognizing the *high-level* program blocks, we will be able to collect the data about each loop. We also utilize a technique developed by Healy et al. [18] that produces the maximum number of loop iterations. In their work they analyze the register transfer list (RTL) representation of the code [7]. Using the assembly code generated in step 2, we will be able to know what assembly instructions are used in the execution of each loop. This required us to also discover loops in the assembly code and find their delimiters.

By calculating the number of loop iterations, finding out the size of loop data, determining which instructions were used in each loop, we would have the necessary metrics to determine if it is more beneficial to execute locally or remotely. The only remaining information that would be needed is the cost of communicating a single byte of data across the network, and the cost of executing each assembly instruction. We found this information experimentally for the cost of communication, and we utilized a similar approach to Tiwari's [36] to calculate the energy cost of each assembly instruction. This was

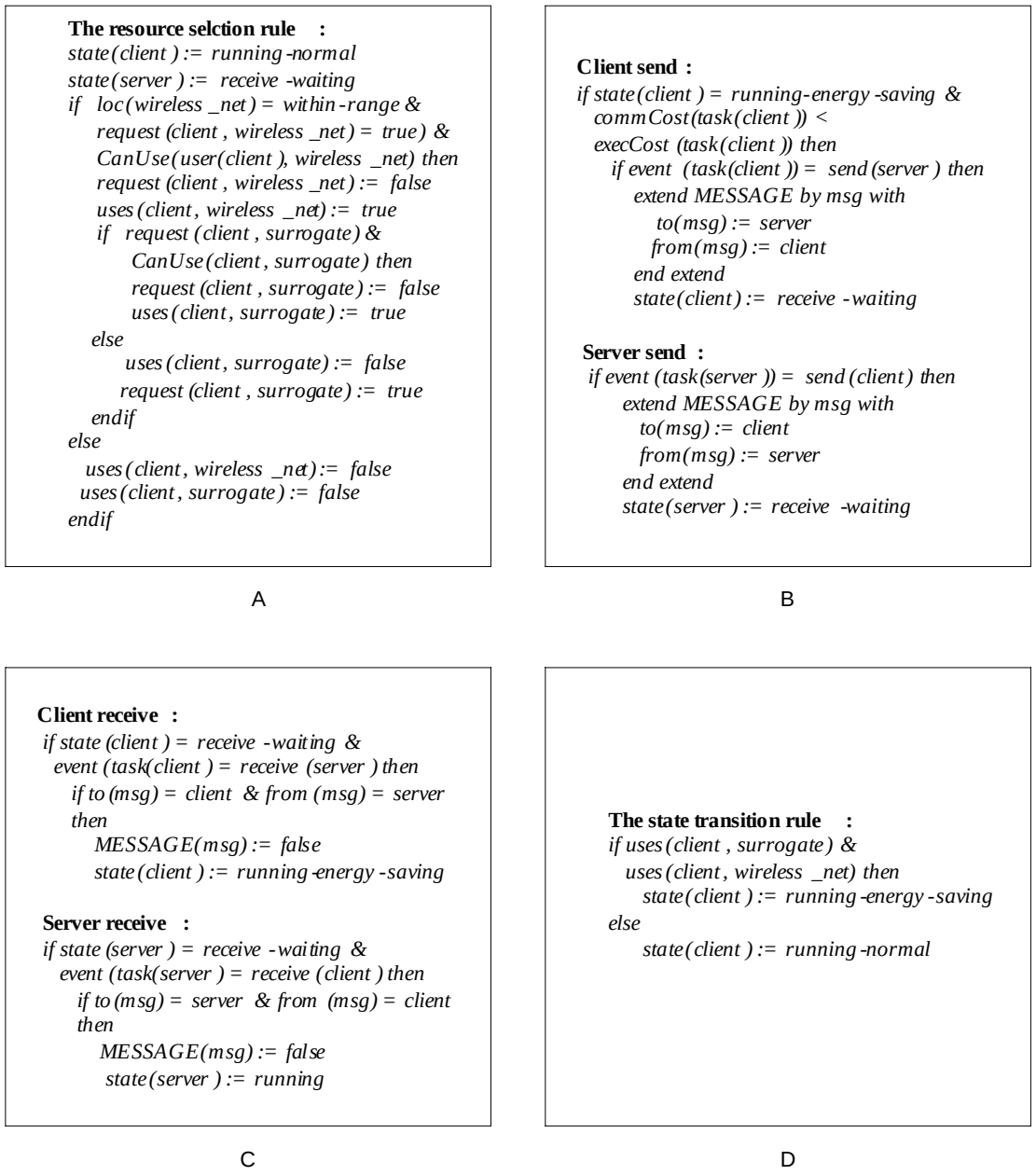

Figure 2: Rule Definitions for the Formal Model. (A) The Resource Selection. (B) The Send Rules. (C) The Receive Rule. (D) The State Transition Rule

done by placing code that corresponds to each assembly instructions (single or multiple) in an empty *'for'* loop and experimentally measure the energy cost differential of each instruction. We used the same method to instrument library function calls such as those in the math library.

3.1 Calculating the Number of Loop Iterations

Healy *et al.* [18] developed a highly useful methodology to predict the worst-case execution time of a program. They implemented this work to be able to statically analyze real-time systems. Their implementation analyzes the RTL representation of a program. They integrated their work in the *vpcc* compiler [7]. The input to their compiler is a C program, and one of the outputs (the one of interest to us) is a file with ".inf" extension. This file contains various types of information one of which is the maximum number of each loop's iteration. Figure 3 shows a C program, and figure 4 shows its ".inf" file.

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

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

}
```

Figure 3: A C Program Compiled with vpcc

3.2 Loop Data and Iteration Acquisition

In this phase of the compilation process, we recognize the maximal CPU blocks (loops) as they are the opportunities for outsourcing. Recognizing the loops at the high-level constitute collecting the data associated with them and size of this data. Additionally, we have designated L-valued data and R-valued data as R-valued data is that data that would be sent to the server and need not be received as it didn't change. This is an optimization within our compiler optimization. We also determined that some loops can not be outsourced because they contain IO operations. This implementation was handled via our parser-like module, we called it the pseudo-parser presented in figure 5.

Once the loop data has been collected, and the loops are identified, the next stage would be to run the code implemented in [18] to calculate the number of loop iterations. Once this code produces the aforementioned ".inf" file, we then parse this file to extract the exact

```
-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 4: The ".inf" File Corresponding to the C Program in Figure 3.

expression to make it adhere to C syntax. For instance the emphasized expression in figure 4 has extra characters associate with it such as the '.', and the '_'. In our code we strip these extra characters out to produce (100 - j - 2) / 3.

3.3 Calculating Size of Loop Data

Once the data of each loop has been collected, we examine each variable involved in the calculation of each loop to see if it is L-valued or R-valued to minimize communication cost. In this part of the code we also recognize if the data involved is an array or not. For arrays, we just multiply the size of the data type by the size of the array.

3.4 Identifying Loop Instructions and Total Loop Execution Cost

In this phase we parse the assembly code associated with the C program. We detect the code that delimits each loop by finding loop entry and exit points. Our target architecture was Intel's Xscale's. A loop can be identified by 3 consecutive instructions, *"cmp"* followed by one of the branch *operations "ble, blt, bge, bgt, bne, and beq"*, followed by an unconditional branch *"b"*. Using the unconditional branch, we were able to find the end of the loop which consists of another unconditional branch to take you back to the beginning of the loop.

At this point we were able to scan the assembly code to find the instructions in each loop and add their total cost to estimate the execution cost of a single iteration of the loop. Inter-instruction costs as well as the cost of pipeline stalls was estimated and added to the total cost to better estimate the energy cost of the entire loop executing once. We were able to handle multiple loops as well as nested loops.

Given the number of iterations of each loop, as well as the cost of a single execution of each loop, in addition to the nesting structure, we were able to generate the formulas that represent the total cost of each loop with respect to its level of nesting.

3.5 Inserting the Outsourcing Code

When reaching this step, we would have generated the formulas the correspond to the cost of each loop as well as the formulas corresponding to the cost of communicating the loop's data. At this point, we insert the necessary socket connection related to the server in the file that is to be installed on the surrogate server, and we insert the corresponding code in the client that is going to run on the mobile device. This created a socket based client/server program.

```
Pseudo-Parser
Input : A C program
Output: An array of loop data structures containing:
        a. The beginning and ending file position of the loop
        b. The variables and their sizes
        c. The L-Valued variable and their sizes

get a token
nestLevel = 0
loopNumber = 0
initialize a stack of loops to empty
while there are tokens in the program do
 if the token is a loop
  loopNumber = loopNumber + 1
  nestLevel = nestLevel + 1
  assign loopNumber to the loop just entered
  assign nestLevel to the loop just entered
  push the current loopNumber on the stack
 else if the token is part of an expression including functions and structures
  if nestLevel > 0
    if you encounter a variable
      if the variable is an L-value
        insert the variable as an L-value in the loop on top of the stack
      else
        insert the variable as a variable in the loop on top of the stack
      end if
    end if
  if you encounter and I/O function then
     The loop becomes non-outsourceable.
  end if
  end if
 else if you reached an end of a loop
  nestLevel = nestLevel - 1
  pop the top of the stack
 end if
  get the next token
 Done.
```

Figure 5: The Algorithm Implementing the Pseudo-parser

Once the socket coded is inserted, we insert an "*if*" statement surrounding each one of the loops that checks if the cost for communication is smaller than the cost of computation for each loop by comparing the two formulas generated before. The statement basically adds the logic that "at runtime" decides if outsourcing is to take place or not.

4. RUN-TIME 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 will not outsource computation.

We created two monitors that support computation outsourcing [3]: a battery monitor and a network monitor. The battery monitor examines the file /proc/apm to determine the amount of battery charge left. This gives it an informed decision on when to go or suggest to go into energy saving mode. The network monitor does surrogate server discovery by sending out a broadcast and receive confirmation from a server under which it is registered to outsource. Once the handshake is established between the client and the server, energy saving mode is established and code may be outsourced.

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 [10] and by Gu *et al.* [16].

The work done by Flinn *et al.* [10] 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 have the same idea presented by Flinn and Satyanarayanan [10] and by Gu *et al.* [16], the solution is much simpler and that is because the battery monitor is a straightforward inquiry to operating system's advanced 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.* [16] and it is based on wireless broadcast for discovering surrogates.

5. EXPERIMENTAL RESULTS

We performed our measurements and experiments in two stages, the first of which was to determine the cost for each assembly instruction. The second was to measure the energy cost of our benchmarks.

Our target architecture was an Intel Xscale (PXA-250) processor installed in a Sharp Zaurus (SL-5600) which is running Linux. We used a low-power wireless LAN card (Socket's low-power wireless LAN) card. The surrogate server was a Linux machine running RedHat 7.2. We developed the applications using Metrowerks Codewarrior for the Zaurus. Measurement was done via

an Agilent Technologies 34401A multi-meter and its supporting software. We measured the current drawn by the device while maintaining a constant voltage (5 volts). Therefore, our measurements compares mainly time and current in amps.

In order to do the measurements, we measure the current drawn by the Zaurus in the idle state. Once we run the program we calculate the difference between the current drawn in the idle state and that drawn by running our benchmark. Then multiply by 5 volts, and by the total time it took to run the program to get the total energy consumed $E = VIT$, where E is the energy consumed, V is the voltage, I is the current, and T is the elapsed time.

To measure each instruction's cost, we first measured the cost of running an empty *'for'* loop, then via the declaration of register variables which the C language allows for *'register int x;'* we were able to isolate the load and the store operations by inserting C code corresponding to them in the loop and measure the difference in energy consumed. If a variable is declared as a register, then assigning another variable to it means that we're only doing a load, and if we assign it to another variable then we're doing a store. Once we were able to measure the cost of the load 'ldr', and the store 'str' operations, then we started adding different operations to the loop and measured their energy cost similarly. This is similar to the work done in [36].

Measuring the communication cost per byte, we wrote a small UNIX socket program that established communication between the client and the server to confirm the statistics provided about the network card. The card data sheet mentioned that the card is active 90% of the time and transmitting at 265 *mA* and receiving at 179 *mA*.

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 200×200, a 300×300, and 400×400 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% respectively. 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. Figure 6 shows the comparison between local and remote matrix multiplication execution on a 400×400 matrix.

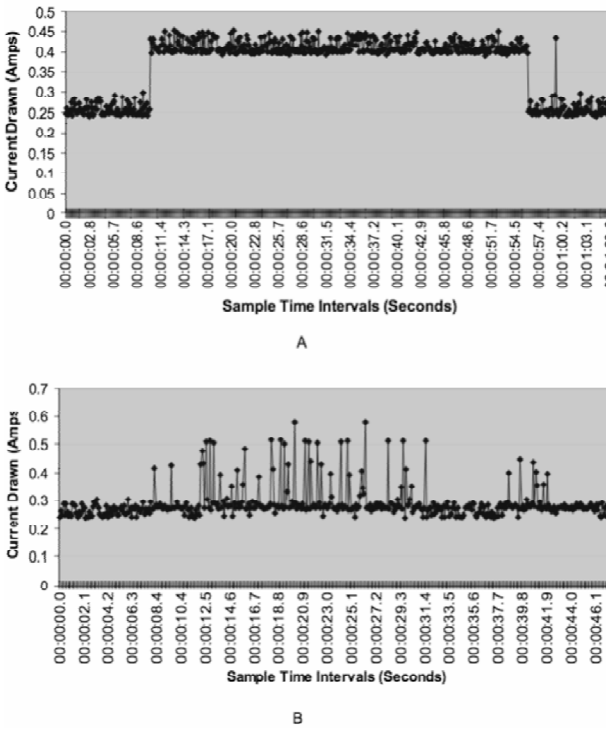

Figure 6: Local vs. Remote Execution for Matrix Multiplication of a 400x400 Matrix

In [3], we presented a benchmark that shows 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 extremely large 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 7, shows the input data passed to the ray-tracing process and figure 8 shows the image produced.

```
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 7: Input File for the Ray Tracing Application

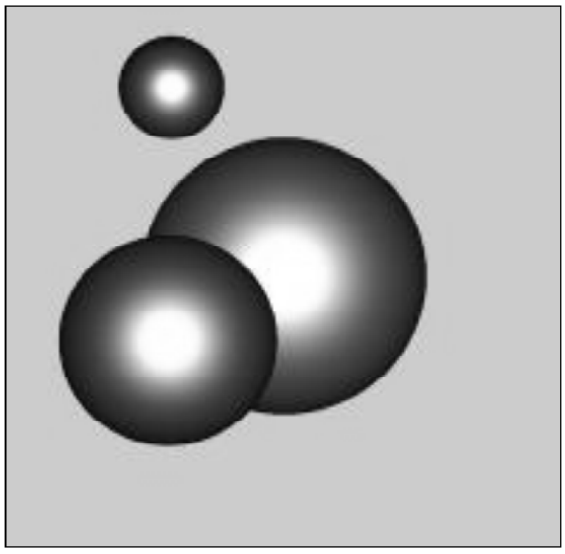

Figure 8: The 2-D Image Representing the 3-D Scene Generated by Ray Tracing

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.

We did our experiments on various sizes of data and we generated a 50x50 image, a 100×100 image, and a 200×200 image for the same scene. The amount of computation was so large that in all three cases, the computation was outsourced. Figure 9 shows the results of comparing local execution vs. remote execution for the 100×100 image.

6. 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 equation $(P = CV^2f)$, 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

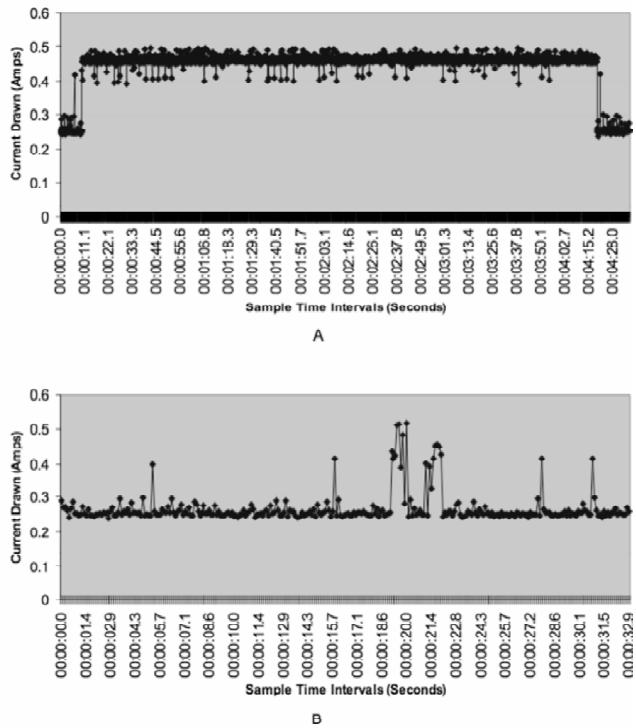

Figure 9: Local (A) vs. Remote (B) Execution for the Ray Tracing Application of a 100×100 Image

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 a tradeoff.

As far as energy management 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.sbsforum.org/specs/ index.html*). In addition to batteries, energy-aware processors were also introduced. Various companies introduced their solutions 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 [20]. However Smit and Havinga [32] argued that reducing voltage indicates reducing performance, therefore additional hardware is needed to balance it out. Capacitance load reduction was also targeted in [15], and [34]. Hardware solutions augmented by compiler support were also done in [5], and [38] by adding additional caches.

As for operating system solutions, advanced power management (APM), and more recently advanced configuration and power interface (ACPI) 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 [39]. Also, energy saving communication techniques are getting increasingly important. Managing communication devise was done in [22]. In [25] 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 [6].

Research and experiments have shown that, with the exception of loop unrolling, and function inlining, compiling for performance does not imply compiling for energy [37]. In [35] 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 [4], and in [14]. Other compiler-based techniques were introduced in [26], and [30]. Also the work done in [28] migrates the compilation process to a server to save energy. In [23], 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 [11], demonstrated a collaborative relationship between operating systems and applications to meet user-specified goals for battery life. They used PowerScope [12] to validate the measurements of energy consumption for accurate estimation.

As far as the applications are concerned, Haid *et al.* [17] developed an excellent application with energy awareness in mind. This work presents designing an energy-aware MP3 player. Additionally, Yuan *et al.* [40] 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 [12], other research has been done to estimate energy for certain applications, systems, and devices. Cignetti *et al.* [8] described an energy model for the Palm.

7. CONCLUSION AND FUTURE WORK

Our experimental results showed that pervasive smart spaces can provide opportunities to outsource computation. By exporting CPU processing over the network, the mobile device delivers expected functionality while consuming less energy and lasting for a longer period of time.

We found that our research benefited and will continue to benefit from research done in the area of real-time systems as it aims at knowing as much about an application before running it. Moreover, we believe that we can also benefit from the area of automated verification for loop invariant generation such as the work done in [29].

In the future we will continue to enhance the implementation of our client/server solution. However, we will augment the work by adding support for dynamic memory allocation, and subroutine outsourcing. 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. Moreover, 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 which 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. Abukmail, A. Helal, A near-Zero Run-time Energy Overhead within a Computation Outsourcing Framework for Energy Management in Mobile Devices, in: 5th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada, April 2008.

[4] 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.

[5] 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.

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

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] G. Smit, P. Havinga**,** A Survey of Energy Saving Techniques for Mobile Computers, Internal Technical Report, University of Twente, Enschede, Netherlands, 1997.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.