Compiler-Directed Remote Task Execution for Power Management

Ulrich Kremer*
Rutgers University
Department of Computer Science
New Brunswick, NJ 08854
uli@cs.rutgers.edu

Jamey Hicks and James M. Rehg
Compaq Computer Corporation
Cambridge Research Lab
Cambridge, MA 02142
{hicks,rehg}@crl.dec.com

Abstract

This paper discusses the potential benefits of application-specific power management through remote task execution. Power management is of particular importance for mobile devices that have to rely on battery power for extended periods of time. Image processing is a class of applications important in mobile environments. Image processing can be used in autonomous robot navigation, target acquisition/classification, keyboard-less input, and aerial surveillance (Micro Air Vehicles), just to mention a few. Experimental results on an image processing application, namely a human face detection and recognition system, indicate the power savings that can be achieved for this important class of applications.

We discuss a compilation strategy that generates two versions of the initial application, one to be executed on the mobile device (client), and the other on a machine connected to the mobile device via a wireless network (server). The client and server codes have to be able to deal with disconnection events. Our proposed compilation strategy uses checkpointing techniques to allow the client to monitor program progress on the server, and to request checkpoint data in order to reduce the performance penalty in case of a possible server and/or network failure.

The reported results have been obtained by actual power measurements on the StrongARM based low-power SKIFF system developed at Compaq's Cambridge Research Laboratory. Initial experiments show that energy consumption can be reduced significantly, in some cases up to one order of magnitude, depending on the selected characteristics of the mobile device, remote host, and wireless network.

*This work was done while the first author was a visiting researcher at the Cambridge Research Lab.

1 Introduction

Power dissipation has become one of the crucial design challenges of current and future computer systems. In a mobile environment, power savings are important to prolong battery life. For a desktop "wall-powered" system, heat emission has become a severe design limitation with respect to transistor densities and clock frequencies. Power management tries to address both of these issues. However, in the context of this paper, prolonging battery life is the main objective.

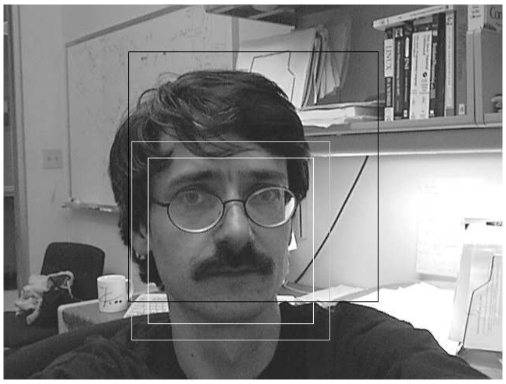
Mobile devices come in many flavors, including laptop computers, Webphones, pocket computers, Personal Digital Assistance (PDAs), and intelligent sensors. Many such devices already have wireless communication capabilities, and we expect most future systems to have such capabilities. There are two main differences between mobile and desktop computing systems, namely the source of the power supply and the amount of available resources. Mobile systems operate entirely on battery power most or all the time. The resources available on a mobile system can be expected to be at least one order of magnitude less than those of a "wall-powered" desktop system with similar technology. This fact is mostly due to space, weight, and power limitations placed on mobile platforms. Such resources include the amount and speed of the processor, memory, secondary storage, and I/O. With the development of new and even more power-hungry technology, we expect this gap to widen even more.

Image processing will be one of the key applications for low-power mobile devices. Low power single-chip imagers, such as the Photobit PB-0100, are becoming available that will allow mobile devices to capture high-quality images. Image processing can be used in the context of robot control and navigation, wide-scale video surveillance, guidance systems for blind people to identify and classify objects, keyboard-less human-computer interfaces, and Micro Air Vehicles (MAVs) to support aerial surveillance, target acquisition and target classification.

Face detection and recognition in video images is an important key technology for several of these applications.

The compilation approach for power and energy management discussed in this paper is complementary to operating systems and hardware techniques. The latter techniques rely on observed past program behavior and resource requirements to predict future program characteristics. A compile-time whole program analysis is often able to determine future behavior and requirements since the entire program is available to the compiler and not just a limited window of recent events. In addition, a compiler may perform high-level transformations and thereby change program behavior and resource requirements, allowing further program optimizations. In cases where compile-time information is not available to perform a particular optimization task, the compiler may generate code that will make optimization decisions at run time based on run-time values of compile-time determined variables and conditions.

In this paper, we give a first assessment of the potential benefits of compiler-directed remote task mapping as an optimization to save energy on mobile devices. We use the tasks of face detection and recognition, in the context of a TourGuide system for visitors to our lab, as an experimental vehicle for our investigation. The main challenges in remote task mapping is the identification of suitable remote tasks where the communication overhead is more than compensated for by the expected power savings. We discuss a compilation framework for remote task execution, and give experimental results for a hand-simulated compilation of our face detection and recognition program. The framework addresses the problem of network disconnection. Power measurements are reported for two platforms: a low-power single-board system called Skiff [3], which was developed at CRL, and a PentiumII based laptop computer.



`Ulrich Kremer:  match distance 192`

Figure 1: Example Image and TourGuide 's output.

2 Remote Task Mapping

The input to the envisioned compilation system is a program written in C or Java that does not contain any user annotations, for instance, to specify tasks for remote execution. Compiler-directed remote task mapping for power and energy management identifies program tasks that (1) can be safely executed on a remote server, and that (2) will lead to power savings on the mobile client system. The profitability of remote task execution will depend on the amount of communication needed between the mobile client and remote server, the time it takes to complete the remote task, and the communication time itself. The latter two points are of particular importance in mobile systems that have a significant power consumption during the time the system is waiting for the response of the server.

Our proposed power management strategy considers disconnectivity. The system will be able to continue program execution while being disconnected from the server, possibly with a degradation of program performance such as an increase in execution time. This is an important feature for any mission critical program with real-time response requirements. Simply putting the mobile system into a hibernation state until the connection to the server has been reestablished is not acceptable. The goal of power management is to save as much energy as possible while running the program to completion within a set of real-time constraints required by the given application. For example, leaving a robot or a MAV without image processing capabilities for extended periods of time may lead to total system loss.

3 TourGuide : A Face Detection and Recognition System

We have developed a prototype application based on face detection and recognition, called TourGuide , which identifies researchers and staff members at Compaq's Cambridge Research Laboratory (CRL). The TourGuide system will consist of a mobile client and a network of embedded servers. The client can be given to visitors to our lab. It allows them to more easily identify and keep track of the members of our laboratory that they encounter during their visit. The system is designed to run on a portable low-power device with wireless communication capabilities that is equipped with a digital camera.

The face detection component of the TourGuide architecture utilizes a neural network-based detector for frontal, upright faces developed at Carnegie Mellon Univ. by Rowley et. al. [7]. The detector uses two networks which were trained to detect faces in windows of 20×20 and 30×30 pixels. These networks are scanned across an image pyramid in a coarse-to-fine fashion. The resulting system can detect any number of frontal, upright faces in a single image, regardless of their size or position. Faces which are tilted in the plane of the image or away from the camera will not be detected. However, if the detector is applied to a sequence of video frames taken from an interaction with a person, it is quite likely that there will be several correct detections.

The face recognition module in TourGuide is applied whenever a face is successfully detected. It is based on the ARENA [11, 10] system developed by Sim et. al. from CMU and JustResearch. ARENA uses a nearest neighbor classifier to identify faces by comparing their distance using an L1 norm to labeled example faces (prototypes) taken from a training set. Input faces are correctly identified when they fall within some predefined distance threshold of the correct prototype face. On relatively small populations the ARENA system has generated impressive results using fairly unconstrained input imagery.

An example input image and the resulting output is shown in Figure 1. A 320×320 central section of the 640×480 input image is passed through the face detector. If successful, the detected face is first enlarged by 20%, summarized, and then processed by the face recognizer. In the shown example, the recognizer reports a successful match and a confidence level for that match.

The structure of the face detection and recognition system is discussed in detail elsewhere [4]. In the envisioned application environment, the system may process several input images before successfully recognizing a person. The entire system has been implemented in C and runs under the Linux operating system. The source code is approx. 6000 lines, including comments. Since the current family of StrongARM processors does not have a floating-point unit, we have developed a fixed-point package for efficient floating-point emulation.

The TourGuide system is an interesting application for our benefit analysis of remote task execution for power management purposes since it exhibits important aspects of mobile applications, including

- *necessary mobility*: The system will be carried by the laboratory visitor who is meeting with researchers inside and outside the laboratory space.
- *potential disconnection*: Some part of the laboratory may not be covered by the wireless communication network. In addition, the communication is interrupted while the visitor is leaving the laboratory space.
- *continuous operation*: Even if the visitor is not within the range of a base station, the system should continue to work, although not at the same performance level as within the range of the network.

Dealing with all or just a subset of these aspects is a challenge, and we believe that compiler support will play a crucial role for effective power management of such applications.

4 Compilation Strategy

The basic idea of our compilation strategy is to identify program tasks that are safe to be executed remotely, and that will lead to power savings on the mobile architecture due to remote execution. During remote execution, the mobile machine will enter a hibernation state in order to save as much energy as possible. It is the compiler's responsibility to insert code that will initiate power state transitions on the mobile machine. While waiting for the completion of a remote task, the mobile client and server may get disconnected. If after a predetermined waiting time the remote server has not finished executing the assigned task, the local client will reexecute the task locally.

The proposed compilation strategy is based on a client/server model and consists of several steps. In the first step, program tasks have to be identified that can safely be executed remotely, i.e., on the server. In the second step, the profitability of remote task execution is examined for remote task candidates. In the final step, two versions of application are generated, one that will be executed on the mobile machine, and one on the remote server accessible from the mobile machine via a wireless connection.

Disconnection events are dealt with through a checkpointing mechanism that provides on-demand data exchange at compile-time determined program points. This checkpointing mechanism serves two purposes, namely (1) informing the client about the progress of the remote task on the server, and (2) allowing the client to receive partial results in order to avoid entire task reexecution in the event of a server or network failure. The version of the application running on the remote server can be considered a specialized server for that application.

objects	life range[1]	size in bytes	comments
		replicated	
neuralNets	2 - 7	300K	neural networks
faceMask	2 - 7	412	20×20 `pnm` image
facesDB	2 - 7	2.8M	faces data base of CRL staff members
		non-replicated	
image	3	60K	640×480 `jpeg` image
imageBW	4 - 5	310K	640×480 `pnm` image
imagePyramid	4	153K	10 levels, based on cropped 320×320 "imageBW"
faceDetected	5	12	location & size of det. face
faceSD	6	269	16×16 scaled-down image
finalAnswer	7	200	output of recognizer

[1] life on entry to node(s).

Table 1: Data objects and their life ranges for the `main` routine of the TourGuide system program.

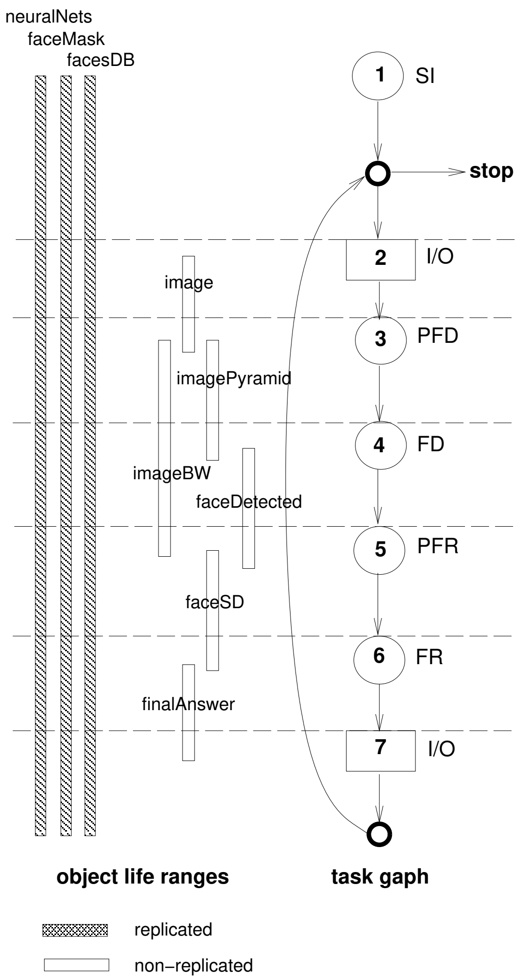


Figure 2: Task graph with object life ranges for the TourGuide system.

The compilation strategy assumes that the entire source program is available during compilation. The compiler will classify I/O operations as performed on *replicated* data or *client* data. Replicated data can be made available to the client and remote server before program execution, i.e., does not have to be communicated during program execution. Client data is acquired by the client, and may lead to communication to the remote server. We will refer to I/O operations such as image acquisition as client I/O. Figure 2 shows the task graph of our TourGuide system, together with its object life ranges. The task graph nodes represent system initialization (SI:1), image acquisition (I/O:2), face detection preprocessing (PFD:3), face detection (FD:4), face recognition preprocessing (PFR:5), face recognition (FR:6), and final answer (I/O:7). Table 1 summarizes the data objects and their life ranges relative to the subtask nodes in the task graph.

A strategy for remote task execution may evaluate the profitability of remote execution for each individual task between a pair of client I/O operations. Our basic strategy is not as general, and is based on the assumption that the network and server is available most of the time. In such a scenario, the client may want to initiate remote execution as early as possible, and once initiated, may want to execute as many tasks as possible remotely, i.e., execute all subsequent tasks remotely until the next client I/O operation is reached. During remote execution, the client may enter a hibernation state. The client may wake up periodically to check the progress on the server, and may request checkpoint data from the server before entering the next hibernation interval. Suitable policies for wake-up and on-demand checkpointing are currently under investigation [4]. For example, a client may request the execution of a checkpoint if the wireless signal strength is below a predefined threshold, indicating a potential disconnection event in the near future. The wake-up and checkpointing policies will be implemented through a runtime library and code generated by the compiler as part of the client and server programs. No communication is necessary between the client and the server once the remote execution has been initiated, except for checkpointing purposes and the final answer computed by the server. The resulting execution model has strong similarities with the concept of program continuations in the sense that once a remote task execution is initiated, it will be able to execute the task to completion without any further communication.

Due to space limitations, a discuss of our checkpointing mechanisms is beyond the scope of this paper.

A *candidate thread* starts with the initial client I/O operation and ends with the final client I/O operation. Once a remote execution has been initiated, the thread will execute all remaining tasks remotely, returning the final answer to the client just before the final I/O operation. The first task executed remotely is called the *remote entry point*. It is important to note that this remote task execution strategy serves as a base-line strategy, and may be refined further [4].

The communication necessary before the initiation of each remote execution corresponds to the non-replicated data objects life at the remote entry point. A candidate thread is safe if all data object instances that need to be communicated to the remote server are known. For example, an object instance that cannot be uniquely identified at compile time must not be live at a remote entry point. Such a life range will force the compiler to eliminate this particular candidate thread. Compile-time techniques such as *points-to* analysis (e.g.: [1]) and *escape analysis* (e.g.: [6] will be used to determine the set of objects that need to be transmitted between client and server.

For example, Figure 3 shows the candidate thread PFD+FD+PFR+FR (entry point 3) of our TourGuide application with its three compile-time determined on-demand checkpoints. The compiler determines the data objects that need to be communicated at a checkpoint in order to resume program execution on the mobile client. The decision whether or not to execute a checkpoint will depend on the selected checkpointing policy and will be evaluated at runtime. The only life object at entry point 3 is `image`. The object is communicated to the server as part of initiating remote execution. The remote server will send object `finalAnswer` to the mobile client once the task FR has been successfully executed on the server.

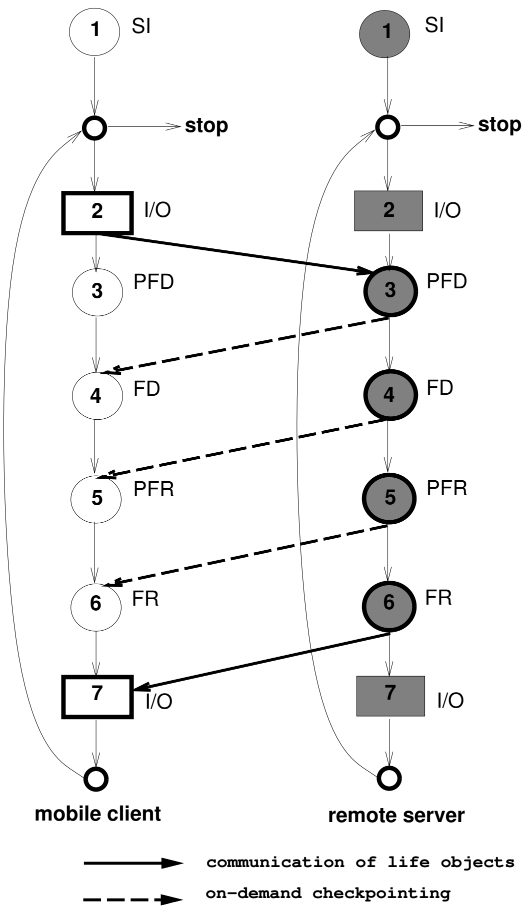


Figure 3: Client and server task graphs for remote thread PFD+FD+PFR+FR. The remote thread has three possible checkpoints.

5 Experiments

The TourGuide face detection and recognition system serves as a test case for the discussed compilation strategy. The three target configurations for this study are shown in Table 5. All machines run the Linux operating system and used Compaq's WL100 11 Mbps wireless LAN card as their wireless network connection. The experiments focus on the execution time reductions obtained through remote task execution for the different configurations.

5.1 Skiff

Skiff has been designed as a low-power computing appliance and is based on a 233MHz StrongARM processor (SA-110). It has 32MB of SDRAM, 16KB I-cache, 16KB D-cache, 8MB flash memory, one 10Mbps Ethernet port, and two PCMCIA card slots, but no floating point unit. A full system specification is given elsewhere [3]. The system has separate power planes (2V, 3.3V, and 5V) and allows power measurements for individual system components such as the processor, memory subsystem, and controllers. The power measurements were taken using a Tektronix oscilloscope at each of these power measurement points. In addition, the power supply reported the overall power consumption of the entire system, allowing power measurements for system components such as the power

mobile client	remote server
SA110 (SKIFF) 233 MHz	PentiumII 300 MHz
SA110 (SKIFF) 233 MHz	Dual Proc. PentiumIII 450 MHz
PentiumII 300 MHz	Dual Proc. PentiumIII 450 MHz

Table 2: The target systems for the case study.

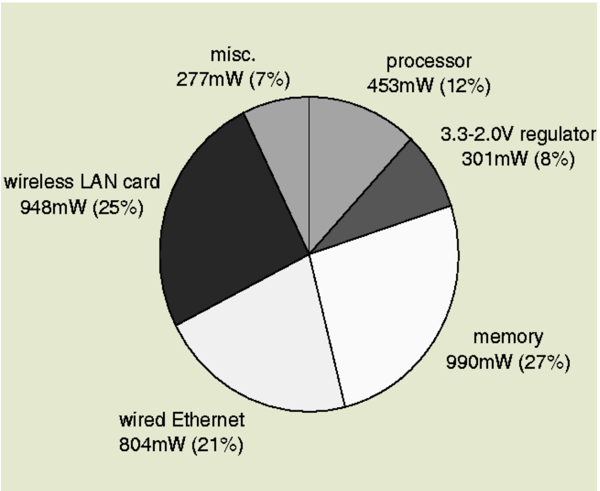


Figure 4: Average power measurements for TourGuide on Skiff.

regulators.

Although Skiff has been designed to serve as a low-power computing appliance and not a mobile device, its system characteristics are similar to other handheld computers such as Compaq's Itsy [2] and Compaq's iPAQ H3600, which are also based on a StrongARM processor (SA-1100). Figure 4 shows the average power dissipation and distribution of Skiff for the four main tasks of TourGuide in a stand-alone mode. The TourGuide system program was compiled using `gcc -O3` and then executed and measured for several input images. Overall, Skiff dissipates 3.77W with a wireless PC card, and 2.82W without such a card. Note that even though the wired Ethernet connection is not "in use", the Ethernet controller consumes significant power. The same holds for the wireless LAN card. We were only able to take an overall power measurement for the PentiumII based laptop. Without the display, the laptop dissipates 19W.

candidate thread	execution time (in seconds)
Skiff local only	8.691
PII local only	0.7083

	Skiff & PII	Skiff & PIII	PII & PIII
PFD+FD+PFR+FR	0.877	0.624	0.469
FD+PFR+FR	4.586	4.347	0.655
PFR+FR	7.716	7.543	0.655
FR	8.702	8.632	0.626

Table 3: Execution times of four candidate threads.

5.2 Initial Benefit Analysis

In the initial experiment, we measured the execution times of the basic tasks in the task graph and the cost of communicating non-replicated life objects at each of the four remote entry points. The communication was performed by reading (receiving) and writing (sending) the life data objects to and from files residing on the remote host. In other words, communication was done through the network file system (NFS). The reported results are based on four input pictures. Three measurements were performed for each picture, and the reported numbers are the medians over all runs and images. Timings are wall clock times in seconds.

Mainly due to the fact that Skiff does not have a floating point unit, floating point intensive computations such as in PFD and FD are executed on the order of $35\times$ faster on the dual processor PentiumIII system than on Skiff. For this case study, all four possible candidate threads, namely PFD+FD+PFR+FR (entry point 3) , FD+PFR+FR (entry point 4), PFR+FR (entry point 5), and FR (entry point 6) are safe. Table 3 shows the expected overall execution times for the different candidate threads and target systems. The figures were computed as the sum of execution times for tasks and communications performed on the client and host. Communicating a data object involves writing the object into a file on the sender side, and reading it out of the file on the receiver side. The reported figures exclude system initialization times (SI), and the time needed to acquire the input jpeg image.

Execution time reductions correspond to the expected energy savings if the mobile client does not support hibernation states for its components. The wireless PC card leads to additional power dissipation of 948mW as shown in Figure 4. A Skiff board without the wireless LAN card dissipates 2.82W. Figure 5 shows the expected energy consumption for Skiff given the additional power costs of the wireless connection. The stand-alone version does not initiate any remote execution and does not use the wireless PC card, i.e., the PC card is not inserted into one of Skiff's PCMCIA slots and therefore does not consume any power. For the PentiumII laptop, the additional power consumption of the wireless card is not significant (5% increase). For both target systems, we did not observe a significant difference in power consumption due to program initiated communication over the wireless Ethernet connection, i.e., the costs for sending and receiving were comparable.

The results show speedups of up to 13.9x, and resulting power savings of up to 10x on Skiff. However, the energy consumption can be larger for some candidate threads as compared to the version of the code that is executed only on the mobile client due to the additional energy consumed by the plugged-in wireless LAN card. On the laptop, energy savings on the order of 30% are possible. In contrast to Skiff, all candidate threads lead to energy savings. The reported figures do not consider power savings due to transitions into low-power hibernation states on the mobile client.

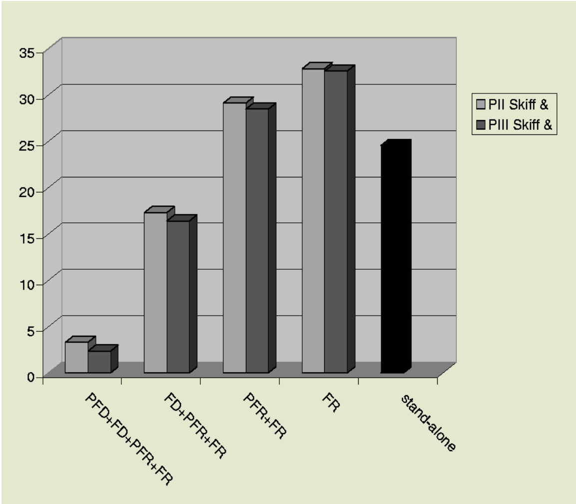


Figure 5: Expected energy consumption on Skiff in Joules for the four remote candidate threads of the TourGuide system and two server target architectures. Reported figures are based on candidate threads execution times and Skiff's average power dissipation (3.77W with wireless LAN card, 2.82W without).

6 Related Work

Executing task on a remote server for the purpose of power savings is not a new idea. Researchers at UCLA and the University College London have investigated the profitability of remote task execution for operating system level tasks [9, 5]. Rudenko et al. have developed a remote processing framework that supports remote task mapping [9]. Files are the objects that are communicated between a client and server. For a set of compilation tasks, the framework achieved power reductions on the order of 3 to 6 times as compared to the compilation done on the client machine. On a task set that includes text processing and Gaussian Elimination tasks in addition to compilation tasks, they report power consumption savings of up to 50% [8]. Battery lifetime extensions of up to 21% were reported by Othman and Hailes [5]. These results were obtained by simulation, and different assumptions about the available network bandwidth and CPU processing power of the client vs. the server machines. To the best of our knowledge, no compiler work has been done to support remote task execution for the purposes of energy savings.

Our proposed compile-time techniques identify remote subtask and determine when their execution may be profitable in terms of overall energy savings. For many applications, compilers are able to analyze entire programs and predict their future behavior and resource requirements. Operating systems and hardware approaches rely on the past observed program behavior to predict future program characteristics. In addition, our work addresses the important issue of network disconnection and failure recovery. Finally, our power measurements are more precise than previously reported figures since we are able to measure instantaneous power directly at the hardware level.

7 Conclusions and Future Work

Remote task execution can be an effective compiler optimization for image processing applications. Preliminary experiments on a StrongARM based system show potential energy savings of one order of magnitude, mainly due to the reduction in overall execution times. On a PentiumII based laptop computer, up to 30% of the energy was saved. However, the experiments also show that remote tasks have to be selected carefully since a poor choice can lead to a significant increase in overall energy consumption.

We are currently investigating power dissipation models that can be used within a compiler. In addition, compiler techniques are needed to initiate transitions into hibernation states, allowing additional energy savings. The ability to dynamically "shut-down" and "wake-up" different system components will lead to dynamically changing power dissipation distributions across the active system components. As a result, cost/benefit tradeoffs of compile-time power optimizations will change. An implementation of the discussed compilation strategy and an initial strategy for compiler-directed power-state transitions is underway.

References

[1] R. Chatterjee, B. Ryder, and W. Landi. Relevant context inference. In *ACM SIGPLAN Symposium on the Principles of Programming Languages (POPL)*, pages 133–146, January 1999.

[2] K. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson. Quantifying the energy consumption of a pocket computer and a Java virtual machine. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Santa Clara, CA, June 2000.

[3] J. Hicks and U. Kremer. Skiff - a platform for pervasive computing. Technical Report 2000-3, Compaq Cambridge Research Laboratory (CRL), 2000. To appear.

[4] U. Kremer, J. Hicks, and J. Rehg. Compiler-directed remote task execution for power management: A case study. Technical Report 2000-2, Compaq Cambridge Research Laboratory (CRL), 2000. To appear.

[5] M. Othman and S. Hailes. Power conservation strategy for mobile computers using load sharing. *Mobile Computing and Communications Review*, 2(1):44–50, 1998.

[6] Y. Park and B. Goldberg. Escape analysis on lists. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'92)*, pages 116–127, June 1992.

[7] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[8] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[9] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning. The remote processing framework for portable computer power saving. In *Proceedings of the ACM Symposium on Applied Computing (SAC99)*, San Antonio,TX, February 1999.

[10] T. Sim, R. Sukthankar, M. Mullin, and S. Baluja. High-performance memory-based face recognition for visitor identification. Technical report, Just Research, 1999.

[11] T. Sim, R. Sukthankar, M. Mullin, and S. Baluja. Memory-based face recognition for visitor identification. In *Proc. 4th Intl. Conf. on Face and Gesture Recognition*, pages 214–220, Grenoble, France, March 2000.