High-Level Power-Reduction Heuristics for Embedded Real-Time Systems*

Osman S. Unsal, Israel Koren and C. Mani Krishna

Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003

Abstract. Power constrained real-time systems are of increasing importance in defense, space, and consumer applications. In this paper, we focus on high-level power issues in large scale real-time systems. In particular, we apply our approach to loosely coupled and tightly-coupled systems. We specifically show that significant power savings are possible through simple high-level application-oriented heuristics.

1 Introduction

High-level power estimation and modeling is a well established research area [9, 10]. However, most of the research concentrates on saving energy within individual nodes [3, 5, 20], while other approaches are mostly concerned with routing [16, 17] or topology [13]. Although there are a few papers that deal with high-level power issues in real-time systems [2, 4], there is, to the best of our knowledge, no effort that brings together intra- and inter-node concerns for large-scale real-time systems. In our previous work, we developed such a combined system-level approach unifying inter- and intra-node level power issues. [18].

This paper explores the feasibility of using high-level power saving heuristics in large scale real-time systems. Since this domain is historically divided into loosely-coupled and tightly-coupled systems, we will be looking at each of the two subdomains separately. Loosely-coupled real-time systems, also termed distributed systems, are often characterized by the lack of a shared memory, with each node's operating system responsible for its own private memory. Tightly-coupled real-time systems, on the other hand, are typically shared-memory systems with a single operating system. This architectural difference has implications for power management.

Due to the lack of shared memory, the bottleneck in loosely-coupled systems is the latency in accessing data residing in another processor's private memory. Therefore, the focus in loosely-coupled systems is on replication [1, 11, 19]. In Section 2, we look at selective replication of data structures for power optimization. On the other hand, the bottleneck in tightly-coupled systems is the processor-to-shared-memory interface. Typically, partitioned memory banks are used to mitigate this bottleneck [7]. In Section 3, we will study the use of memory/processor sleep modes in such a power-constrained real-time multiprocessor system. We conclude our analysis with a brief discussion of our results in Section 4.

2 Loosely-Coupled Systems

In this section we consider distributed real-time systems and examine the correlation between power consumption and various system attributes such as task assignment and network topology. Within this framework, our particular aim is to study the problem of positioning copies of shared data structures to reduce power consumption. Each node has its private memory and each task has an associated worst-case execution time and deadline. Shared data originates from, and is updated by, a single source task. This shared data is then consumed by multiple tasks in a read-only fashion. If two tasks reside on different processors then the communication power cost depends on the routing algorithm and topology. To save energy, a source remote task's data structures may be replicated closer to the consuming task(s). The objective is to find the ideal degree of replication. Increasing the replication increases local memory size and its energy consumption, while decreasing the volume of network transfers and the associated power consumption. Therefore a "sweet spot" may exist, beyond which increasing the degree of replication increases the overall energy consumption [18]. Memory

* This work is supported in part by DARPA through contract No. F30602-96-1-0341. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force or the U.S. Government.

consistency is preserved by updating all the replicated copies of a datum when a source task writes a shared datum to its private memory. In typical applications, reads dominate writes. For example, store instructions comprised 26% of data operations, while load instructions took 74% in the study of a subset of the SPECint92 benchmarks [7]. This characteristic renders replication useful.

In this work, we consider selective replication. By selectively replicating the data structure of the source task only at some nodes, we seek to reduce the energy consumption. The reads at the consuming node are done from the closest replicated copy. The framework for the above system already exists at the OS or application level, so this extension can be easily implemented.

In our previous work (which involved replication at each consuming node) [18], we saw that task assignment has a major impact on energy consumption. We analyzed various task assignment strategies such as round-robin, first-fit and best-fit. In this paper we also present results for a new energy efficient *closest-neighbor-first* heuristic that assigns data consuming tasks topologically close to the source task.

The optimal location and number of replicated copies changes for each degree of replication. Since the problem is NP-complete, an adaptive simulated annealing heuristic was used. The details of this heuristic are provided in the next section. There, we also compare selective replication with full replication.

Exhaustive (Optimal) Examples For the results in this section, the task execution times, periods as well as intertask communication sizes are randomly selected. The number of nodes is 16, the write-to-read power ratio is 1.22, 15% of the memory operations are writes and 85% are reads, and per-hop remote access energy cost is three times that of a local access energy cost.

We concentrate on the impact of task allocation. The system here is a 16-node mesh with 40 tasks. The results of our previously mentioned *closest-neighbor-first* heuristic are compared against round-robin allocation in Figure 1. Fewer replicated copies are needed when using the *closest-neighbor-first* algorithm and the optimal energy is lower.

Non-exhaustive Heuristic The previous exhaustive heuristic, while giving optimal results, runs in

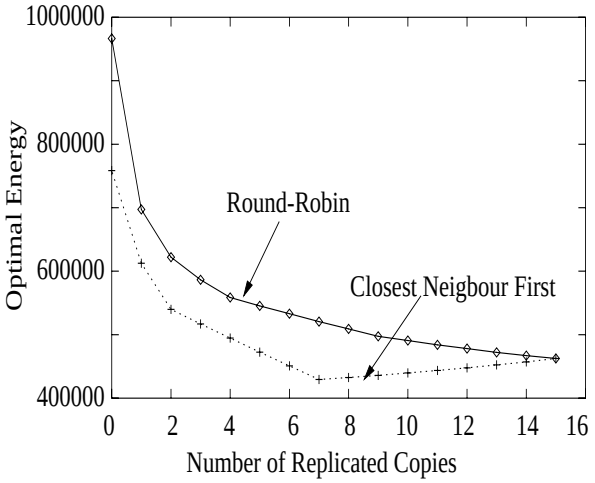


Fig. 1. Comparing the allocation schemes for a 16-node mesh.

$O(2^{number\_of\_nodes})$ time. This becomes very expensive when the number of nodes is large. Therefore, an intelligent search heuristic is needed. We modified an adaptive simulated annealing scheme [14] for our purposes. The pseudo-code of the heuristic is given in Figure 2. Here we start with a sufficiently high temperature, where we allow liberal scanning of the configuration space; and slowly decrease the temperature to concentrate on a minimum. We either transfer a replicated point from a node to another node (in the function transport_one) or we change the number of replicated points by one (in the function incdec_by_one). The metrop function in Figure 2 is the well-known metropolis operator, $exp(-E/kT)$, where E is the energy difference with previous step, k is the Boltzmann constant and T is the current temperature. Success in the metropolis operator function returns 1 for the value of variable *ans*. We executed our simulated annealing heuristic for a 16-node system with different parameters and compared the results against exhaustive runs for the same inputs. Our heuristic was able to find the optimum result for all cases. Then, for the 32-node 40-task system, we compared our selective replication scheme to the case of no replication for different parameter values. The results in Table 1 show significant power savings. The results also show the relative sensitivity of the selectively replicated results to changes in parameter values, whereas cases with no replication are relatively insensitive.

	2X WR		30%WR		5%WR		L.HOP		BASE	
	FR	NR	FR	NR	FR	NR	FR	NR	FR	NR
Hypercube	75837	132500	75275	127250	60895	124750	55102	59750	67587	125750
Mesh	80540	206750	82700	201500	63865	199000	55570	68000	72290	200000
Ring	94647	231500	110915	226250	68320	223750	57137	70750	86397	224750
Torus	72125	97850	68345	92600	59905	90100	54387	55900	63875	91100
Chordal	72867	97850	69335	92600	60152	90100	54772	55900	64617	91100

Table 1. Comparing the minimum energy of no replication to full replication. 2xWR means that write energy is two times of the baseline, 30% WR that 30 percent of the data operations are writes, 5% WR that 5 percent of operations are reads, L.HOP that per-hop energy cost low, i.e. it is one third of the baseline local access cost and BASE is the baseline configuration. FR denotes full replication and NR no replication.

```
main()
{
   nover=100*no_of_nodes;
   nlimit=10*no_of_nodes;
   tfactr=0.9;
   t=tstart;
   create_random_assignment();
   for (jj=1, jj<=100; jj++)
   {
      ldec=random();
      if (ldec>0.5)
      {
         incdec_by_one();
         metrop();
      }
      else
      {
         transport_one();
         metrop();
      }
      if (ans==1)
      {
         nsucc++;
         adjust_assignment();
      }
      if (nsucc>=nlimit) go to end_iter;
   }
end_iter: t=t*tfactr;
   if (nsucc==0) go to end_anneal();
   }
end_anneal();
}
```

Fig. 2. Pseudo-code for Simulated Annealing Heuristic

3 Tightly-Coupled Systems

Most modern memory and processing devices have a sleep mode. A big difference exists in power consumption between the active and sleep mode of devices. This is especially the case for memories. As an example consider a 4M $\times$ 16 EDO DRAM, Micron MT4LC4M16R6 [12]: it has an active-mode power consumption of 580 mW, and a sleep-mode power consumption of 1.65 mW: a ratio of over 350. (By contrast, the corresponding ratio for processors tends to be around 5).

To see how task allocation to memory modules can make a difference to power consumption, consider the following contrived and oversimplified example. Suppose we have a five-processor multiprocessor with two multiport memory modules, running two task sets A and B. Suppose precedence requirements dictate that all the tasks in B can run only after all tasks in A finish. No other precedence conditions exist. Suppose the size of each memory module is large enough to hold one of the two task sets completely; however, neither can hold *all* the tasks in both A and B.

If we allocate all of task set A to memory module M_1 and all of set B to module M_2, M_1 will be active while tasks in A are running, and asleep when they are not. A similar case holds for M_2 with respect to task set B. By contrast, if we allow some tasks of A to be assigned to M_1 and some to M_2 and similarly for tasks of B, both modules will have to be active for a longer time. Hence, the former arrangement permits power savings over the latter. (We have not taken memory interference into account in this example, but the argument will hold as long as memory interference remains a second-order effect).

Task assignment to memory modules to reduce energy consumption was considered before, in [6], for general-purpose systems. In this paper, we study the problem specifically for real-time workloads. The potential for energy savings is higher for overdesigned systems since the devices can be put to sleep mode more often. This is especially the case for real-time multiprocessor systems since memory-wise and compute-wise these systems are typically overdesigned [8].

The system model is shown in Figure 3. We assign tasks to memory banks, i.e., choose one memory bank to hold all of the the data and text sections of a task. If only X banks are used by the application tasks at a given time, the remaining $P-X$ banks can then be switched to sleep mode to conserve energy.

More precisely:

$$\frac{d(Energy(t))}{dt} = N \cdot P_{cpu} + X(t) \cdot P_{ma} + (P - X(t)) \cdot P_{ms} \quad (1)$$

where $Energy(t)$ is the cumulative system energy at time t, P_{cpu} is the CPU power consumption, P_{ma} is the memory power consumption in active mode and P_{ms} is the memory power consumption in sleep mode.

The numerical results in this paper are for a

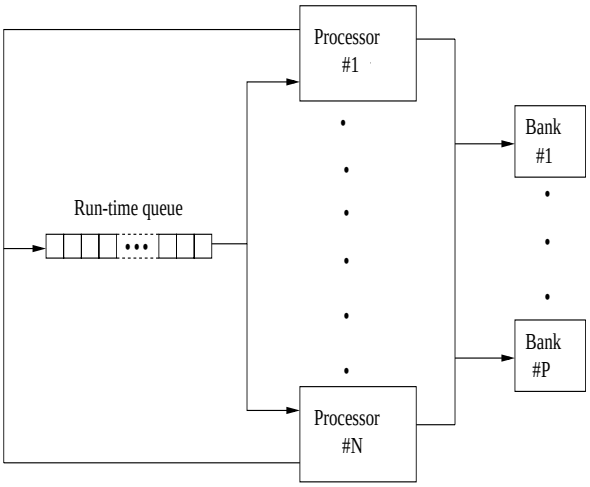


Fig. 3. Closely-coupled system model

shared memory multiprocessor with two memory banks and two processors with a shared run-queue. All tasks have a deadline and a worst-case execution time associated with them. The periods are randomly chosen. The execution times are random as well; however they are chosen so that the total processor utilization is equal to some specified quantity. Each task has a memory size requirement, which is randomly chosen in such a way to satisfy a given memory load. Then, the energy expression for the 2-bank 2-CPU system case is:

$$\frac{d(Energy(t))}{dt} = 2 \cdot P_{cpu} + X(t) \cdot P_{ma} + (2 - X(t)) \cdot P_{ms} \quad (2)$$

X is 0 if there are no tasks in the run queue, 1 if there is one task in the run queue or if there are two tasks in the run queue that are assigned to the same bank, 2 if there are two tasks in the run queue that are assigned to different banks. Our multiprocessor scheduling heuristic is the well-known earliest-deadline-first adapted to the closely-coupled system. In this variant, the chosen task is allocated to the first available processor. We run our simulation over the least common multiple of task periods and check for deadline misses. Since our focus is on real-time systems, we reject runs with deadline misses. Each application is assumed to have an energy budget. Each successful execution of a periodic task contributes to the reward, which continues to accrue until the system energy budget is depleted. All our results, unless otherwise noted, are averaged over 50 runs.

We start by utilizing memory sleep mode only. Our analysis is done across different values of system parameter loads. The system parameters used are the CPU and memory, therefore we need to define the load metrics:

$$CPULoad = (\sum_{k=1}^{n\_tasks} Exe_k/Per_k)/N_p \quad (3)$$

$$MemoryLoad = (\sum_{k=1}^{n\_tasks} Mem_k)/(N_b * S) \quad (4)$$

where Exe_k is the execution time, Per_k the period and Mem_k the memory size of the kth task. N_p is the number of processors, N_b the number of memory banks and S the memory size of each bank. For each energy budget, we plot the reward rate for different CPU loads in Figure 4. As the CPU load is increased, the reward becomes less sensitive to it, and ultimately saturates.

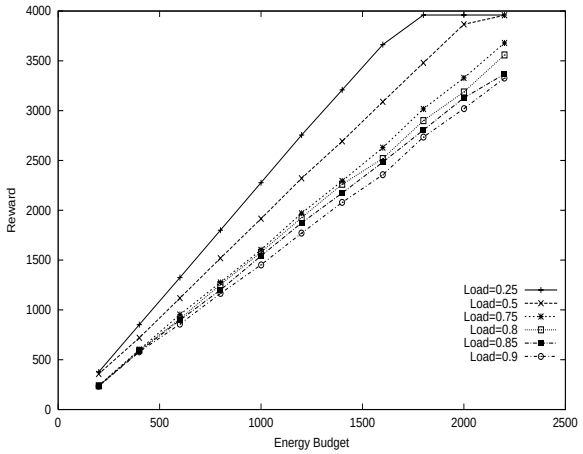


Fig. 4. The system reward as a function of the energy budget using only the memory sleep mode.

We now define overlap to be the time period when both memory banks are active. If we can utilize this

overlap by reassigning the overlapping tasks to the same memory bank as much as possible, then the other memory bank can be switched to sleep mode for the duration of the overlap and save power. We compare such an optimized allocation (obtained by using an exhaustive search) with round-robin allocation. The resulting rewards are shown in Figure 5. The optimized allocation has minimum overlap of tasks, and is significantly more energy efficient than the round-robin allocation.

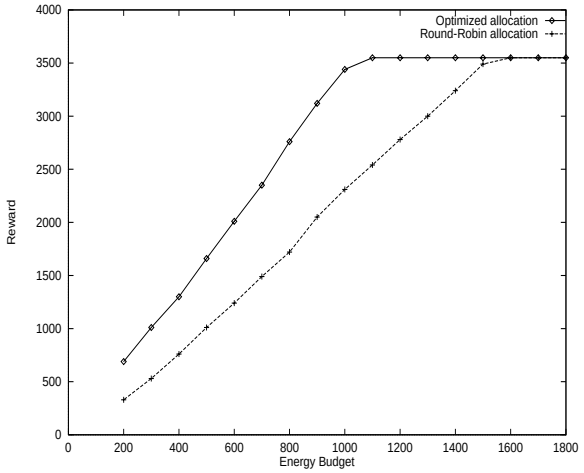
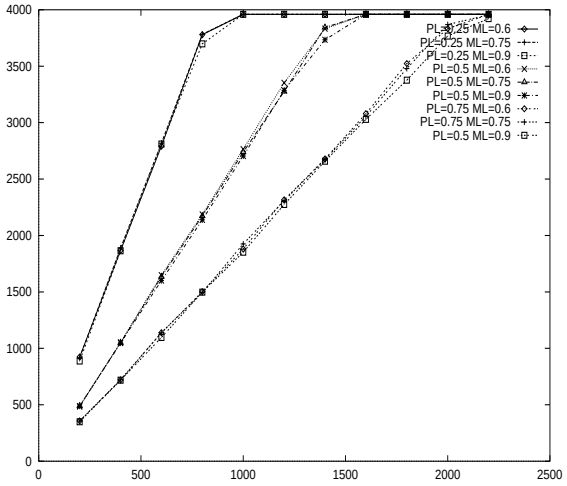


Fig. 5. Comparing the rewards due to the optimized and unoptimized allocation schemes.

An analysis of the effects of memory loading (i.e., the memory requirements of each task) are given in Figure 6 (PL denotes the *CPULoad* and ML the *MemoryLoad*). We require that no task needs to be split between two banks. The results show that the effects are negligible. This leads to the conclusion that under this restriction, CPU load has more effect on the energy reward than memory load.

The additional delay caused by simultaneous access to a memory bank by two tasks must also be considered. We model this penalty by defining interference. We introduce an interference parameter, A, which ranges between 0 and 1. During interference, x cycles of simulation time are translated into Ax units of useful cycles. Therefore, the closer A is to 1, the less the interference penalty. Figure 7 shows the results and as expected the effect of the interference becomes more pronounced at high CPU loads.

Next, we turn our attention to task-period adjustment. Task-period modification is a well-known method in the context of improving control system performance in real-time systems [15]. This method however, has not been used before in a high-level power saving setting. We now make this extension. Readjusting the periods to make them more harmonic can be expected to generate a smaller overlap, and thus lead to lower energy consumption. We are only allowed to *contract* periods, since extending them may violate application-imposed timing constraints. We take random task periods between 20 to 140 time units and contract them to multiples of 20. The results are shown in Figure 8, demonstrating that higher CPU loads can, in some cases, actually lead to higher savings.



PL = processor load; ML = memory load

Fig. 6. Effect of varying memory loads

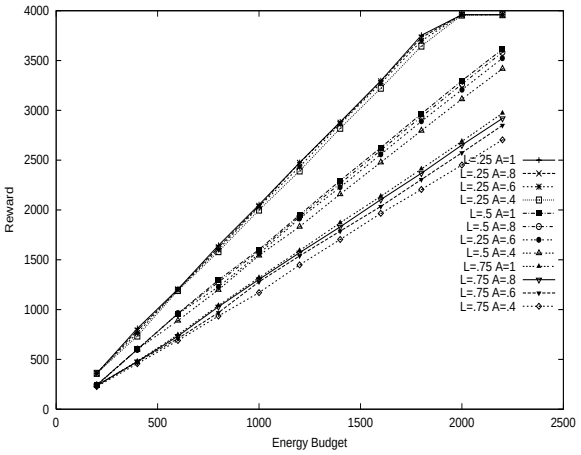


Fig. 7. Effect of memory interference.

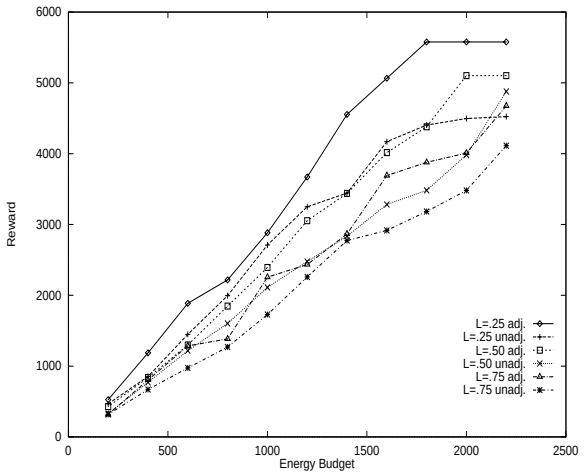


Fig. 8. The effect of period adjustment.

4 Conclusion

We have constructed a framework to gauge the energy impact of various application and OS level heuristics to energy consumption. Our model unifies the inter- and intra-node approaches and is applicable to large-scale loosely and tightly coupled real-time systems. We have shown that relatively simple heuristics can lead to considerable energy savings.

References

1. Ahmad I., Kwok Y.-K., "On Exploiting Task Duplication in Parallel Program Scheduling", *IEEE Transactions on Parallel and Distributed Systems*, September 1998, pp. 872-891
2. Cheng S.-T., Chen C.-M., Hwang, J.-W., "Low-Power Design for Real-Time Systems", *Real Time Systems*, Kluwer Academic Publishers Vol. 15, 1998, pp. 131-148
3. Danckaert K, Masselos K., Catthor F., De Man H. J., Goutis K., "Strategy for Power-Efficient Design of Parallel Systems", *IEEE Transactions on VLSI Systems*, June 1999, pp. 258-265
4. Darko K., Miodrag P., "System-Level Synthesis of Low-Power Hard Real-Time Systems", *Computer Science Department Report*, University of California, Los Angeles, 1996
5. Diguet J. P., Wuytack S., Catthoor F., De Man H., "Formalized Methodology for Data Reuse Exploration in Hierarchical Memory Mappings", *International Symposium on Low-Power Design*, 1997, pp. 30-35
6. Farrahi A. H., Tellez G. E., Sarrafzadeh M., "Memory Segmentation to Exploit Sleep Mode Operation" *32nd ACM/IEEE Design Automation Conference*, 1995
7. Hennesy J., Patterson D. A., "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, 1996, pp 434-435, pp. 106,434-435
8. Krishna C. M., Shin K. G., Real Time Systems, Mc Graw Hill,1997
9. Landman P., "High-Level Power Estimation", *ISLPED* 1996
10. Lee M.T.-C., Tiwari V., Malik S., Fujita M., "Power Analysis and Minimization Techniques for Embedded DSP Software", *IEEE Transactions on VLSI Systems*, March 1997
11. Levy E., Silberschatz A., "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*, December 1990, pp.332-364 *Report No. UCB/CSD-96-914*, University of California, Berkeley, Computer Science Division, September 1996
12. Micron Technology Inc., "MT4LC4M16R6 Data Sheet", www.micron.com/mti/msp/html/data-sheet.html
13. Patel C. S., Chai S. M., Yalamanchili S., Schimmal D. E., "Power Constrained Design of Multiprocessor Interconnection Networks", *IEEE Conference on Computer Design*, 1997, pp 408-416
14. Press W. H., Flannery B. P., Teukolsky S. A., Vetterling W. T., "Numerical Recipes, The Art of Scientific Computing", Cambridge University Press, 1989, pp. 326-334
15. Shin K. G., Meissner C. L., "Adaptation of Control System Performance by Task Reallocation and Period Modification", *11-th Euromicro RTS*, June 1999
16. Singh S., Woo M., Raghavendra C. S. "Power-Aware Routing in Mobile Ad Hoc Networks", *ACM MOBICOM*, 1998, pp. 181-190
17. Sivalingam K. M., Chen C., Agrawal P., Srivastava M. B., "Design and Analysis of Low-Power Access Protocols for Wireless and Mobile ATM Networks", *ACM/Baltzer Mobile Networking and Applications Journal*, March 1998
18. Unsal, O. S., Koren, I., and Krishna C. M., "Power-Aware Replication of Data Structures in Distributed Embedded Real-Time Systems", *Proceedings of IPDPS*, May 2000
19. Vijay S., "Flexible Use of Memory for Replication/Migration inCache-Coherent DSM Multiprocessors", *Technical Report: CSL-TR-99-789*, Computer Science Department, Stanford University, November 1999
20. Wuytack S., Catthoor F., De Man H., "Transforming Set Data Types to Power Optimal Data Structures", *International Symposium on Low-Power Design*, April 1995