In Proceedings of the International Workshop on Hardware/Software
Co-Design, Copenhagen, Denmark, April 2001. To appear.

Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors

Neal K. Bambha
Shuvra S. Bhattacharyya
ECE Department and UMIACS
University of Maryland, USA
{nbambha, ssb}@eng.umd.edu

Jürgen Teich
Computer Engineering
University of Paderborn
Paderborn, Germany
teich@date.uni-paderborn.de

Eckart Zitzler
Computer Engineering
Swiss Federal Institute of Technology
Zurich, Switzerland
zitzler@tik.ee.ethz.ch

ABSTRACT

In this paper, we explore a hybrid global/local search optimization framework for dynamic voltage scaling in embedded multiprocessor systems. The problem is to find, for a multiprocessor system in which the processors are capable of dynamically varying their core voltages, the optimum voltage levels for all the tasks in order to minimize the average power consumption under a given performance constraint. An effective local search approach for static voltage scaling based on the concept of a *period graph* has been demonstrated in [1]. To make use of it in an optimization problem, the period graph must be integrated into a global search algorithm. *Simulated heating*, a general optimization framework developed in [19], is an efficient method for precisely this purpose of integrating local search into global search algorithms. However, little is known about the management of computational (compile-time) resources between global search and local search in hybrid algorithms, such as those coordinated by simulated heating. In this paper, we explore various hybrid search management strategies for power optimization under the framework of simulated heating. We demonstrate that careful search management leads to significant power consumption improvement over add-hoc global search / local search integration, and explore alternative approaches to performing hybrid search management for dynamic voltage scaling.

Keywords

simulated heating, dynamic voltage scaling

1. INTRODUCTION

Dynamic voltage scaling [13] in embedded processors is an important advancing technology that allows the tuning of power/performance behavior as software execution evolves. For example, in non-iterative execution, the operating voltage can be increased or decreased depending on whether a task lies on the critical path; similarly, for iterative systems, the decision might be based on examining the critical cycles. By tuning power/performance behavior at the task level, this can allow the designer to meet a performance constraint with significantly lower energy consumption than if all tasks are required to execute at the same voltage [2].

In iterative embedded multiprocessing systems, such as those found in digital signal processing applications, optimized application of dynamic voltage scaling, and any sort of complex design space exploration in general, is greatly complicated by communication resource contention. One example of this is a shared bus. A processor must first gain access to the bus before it can execute an interprocessor communication (IPC) operation. One consequence of this contention is that under iterative execution that is *self-timed* (processors synchronize with one another only based on interprocessor communication requirements [11]), there is no known method for deriving an analytical expression for the throughput of the system, and thus, simulation is required to get a clear picture of application performance. However, simulation is computationally expensive, and it is highly undesirable to perform simulation inside the innermost optimization loop during synthesis. To avoid such inner-loop simulation, it has been shown that a data structure called the *period graph* can be used as an efficient estimator for the system throughput [1]. In particular, the reciprocal of the maximum cycle mean of the period graph can be used as an efficient estimate of the throughput. The maximum cycle mean is the maximum over all directed cycles C of the sum of the task execution times in C divided by the sum of the edge delays in C. A variety of efficient, low polynomial-time algorithms have been developed for computing the maximum cycle mean (e.g., see [5]).

The mean error and fidelity of the period graph estimator (i.e., its maximum cycle mean) have been quantified, and this analysis has demonstrated the accuracy of the estimation technique [1]. In this technique, the maximum cycle mean of a period graph derived from a simulation (including contention effects) is used to estimate the throughput for the system as task execution times are varied. Specifically, relatively small variations in task execution times are translated to corresponding changes in the node weights of the period graph, and the new maximum cycle mean is used to estimate the performance impact of the given execution time variations.

This period-graph based approach is accurate if the task execution times are varied around a limited region (local search). Thus, in general, it must be combined with a global search strategy that navigates the sequence or “populations” of local regions to explore. In the context of power optimization through voltage scaling, two hybrid search strategies using, respectively, genetic algorithms and simulated annealing in conjunction with period-graph-based local search were shown to yield significant benefit [1]. However, this work was limited in its focus on static, rather than dynamic, voltage scaling, and its use of a *fixed local search config-*

uration throughout the optimization process.

In this paper, we examine the more complex domain of dynamic voltage scaling, and explore adaptive (variable-configuration) *parameterized local search* formulations of period-graph-based local search to optimize use of compile-time resources. For this purpose, we apply the *simulated heating* concept of [19], which provides a general approach to controlling parameterized local search from within an enclosing global search process. We demonstrate that careful search management leads to significant power consumption improvement over add-hoc global search / local search integration, and explore alternative approaches to performing hybrid search management for dynamic voltage scaling.

In this work, we assume that a schedule has been computed beforehand so that the ordering of the tasks on the processors is known. We also assume that the system hardware supports dynamic voltage scaling for the purpose of power reduction. We address the problem of finding a set of voltages for the tasks (i.e., a mapping from tasks into valid voltage values) in order to minimize average power (energy per computation period) while satisfying a prespecified throughput constraint.

2. RELATED WORK

2.1 Task level scheduling

Estimates for task execution times can be obtained through several methods. The most straightforward is for the programmer to provide them as annotations while developing a library of primitive blocks. Analytical techniques also exist. Li and Malik [12] have proposed algorithms for estimating the execution time of embedded software in an efficient manner. Much work has been done on scheduling and binding methods for high level synthesis [15][6][8][4]. These techniques attempt to optimize the schedule makespan, which is a suitable performance metric for non-iterative applications or fully-static implementations, but is not ideally suited to the iterative, self-timed context that we address in this paper.

2.2 Voltage scaling

If processor voltages can be adjusted during run-time, then, by slowing down tasks that are not on the critical path or critical cycle — by reducing the operating voltage in effect when each those tasks executes — an overall power reduction can be achieved. This exploits the quadratic dependence of power on voltage in CMOS technology. Reducing the supply voltage also has the effect of decreasing the clock speed and increasing circuit delay. The circuit delay can be modeled by

$$delay = k \times \frac{V_{dd}}{(V_{dd} - V_t)^2}, \quad (1)$$

where V_{dd} is the supply voltage, V_t is the threshold voltage, and k is a constant [3]. We use a value of $0.8 volts$ for the threshold voltage. The power consumption is given by

$$P = \alpha C_L V_{dd}^2 f \quad (2)$$

where f is the clock frequency, C_L is the load capacitance, and α is the switching activity. The optimization problem we address consists of finding the voltage vector $V = (v_1, v_2, \ldots, v_n)$ for the n tasks in the graph, such that the energy per computation period (average power) is minimized and the throughput satisfies some pre-specified constraint (e.g., as determined by the sample period in a DSP application). Dynamic voltage scaling requires extra hardware, which uses extra power. This was estimated at 10% for the system given in [14]. The time required to switch between voltages was estimated at $10\mu s$ for this system.

2.3 Integrating global search and local search

For many useful optimization problems that arise in hardware-software codesign, efficient algorithms exist for refining arbitrary points in the search space into better solutions. Such algorithms are called *local search algorithms* because they define neighborhoods, typically based on initial coarse solutions, in which to search for optima. Many of these algorithms are parameterizable in nature. Based on the values of one or more algorithm parameters, such a *parameterized* local search algorithm (*PLSA*) can trade-off time/space complexity for optimization accuracy (quality of the optimized result).

Local search techniques can often be incorporated naturally into global search algorithms (*GSA*s) in order to increase the effectiveness of optimization. This has the potential to exploit the complementary advantages of GSAs like evolutionary algorithms (generality, robustness, global search efficiency), and problem-specific PLSAs (exploiting application-specific problem structure, rapid convergence toward local minima).

Such hybridization of optimization algorithms arises naturally in many contexts [7]. For instance, in the field of evolutionary computation, many evolutionary algorithm/local search method combinations can be found in the literature, e.g., [9, 16]. When employing PLSAs in the context of embedded system implementation, however, a critical issue is how to use computational resources most efficiently under a given compile-time budget (e.g., a minute, an hour, a day, etc.), which translates into a problem of appropriately reconfiguring successive PLSA invocations to achieve appropriate accuracy/compile-time trade-offs as optimization progresses.

For example, a PLSA formulation of period-graph-based local search might involve N randomly-determined sets of execution time variations, which are each evaluated using the period graph to select the "best" variation. As N is varied, the local search consumes more time (drawing time away from the global search under a given compile-time budget), but in general, produces better results. Under a fixed configuration, N would be fixed before optimization begins, and would remain the same across all PLSA invocations; whereas, in an adaptive approach, the PLSA parameter would be varied in an attempt to streamline the allocation of resources between global and local search.

3. Hybrid global/local search

The general hybrid global/local search scenario can be described as follows [19]. Suppose that we have a GSA G operating on a set

of solution candidates and a PLSA $L(p)$, where p is the parameter of the local search[1]. Let $\Re$ and $\aleph$ respectively denote the set of real numbers, and the set of natural numbers, and let:

- $C(p)$ denote the complexity (worst-case run-time) of L for the parameter choice p;
- $A(p)$ be the average accuracy (effectiveness) of L with regard to p; and
- $R \subseteq \Re$ denote the set of permissible values (the *parameter domain*) for parameter p (for example, R may be described by an interval $[p_{\min}, p_{\max}]$).

Furthermore, suppose that for any pair (p_1, p_2) of parameter values we have that

$$(p_1 \le p_2) \Rightarrow (C(p_1) \le C(p_2)) \textbf{ and } (A(p_1) \le A(p_2)). \qquad (3)$$

That is, increasing parameter values in general result in increased consumption of compile-time, as well as increased optimization effectiveness.

Generally, it is very difficult, if not impossible, to analytically determine the functions $C(p)$ and $A(p)$, but these functions are useful conceptual tools in discussing the problem of designing cooperating GSA/PLSA combinations. The techniques that we explore in this paper do not require these functions to be known. The only requirement we make is that the monotonicity property (3) be obeyed at least in an approximate sense (fluctuations about relatively small variations in parameter values are admissible, but significant increases in the PLSA parameter value should correspond to increasing cost and accuracy). Consequently, a tunable trade-off emerges: when $A(p)$ is low, refinement is generally low as well, but not much time is consumed ($C(p)$ is also low). Conversely, higher $A(p)$ requires higher computational cost $C(p)$.

In Fig. 1, the *GSA/PLSA* hybrid that is taken as the basis for the optimization scenario in this paper is explained:

The GSA G operates on a set of N solution candidates (N may be equal to one for, e.g., simulated annealing or greater than one for, e.g., an evolutionary algorithm); per optimization step, it creates a new set of solution candidates depending on the previous solution set and the quality function associated with it. The PLSA L is used to refine and/or to evaluate the solution candidates generated by G; its parameter p is adapted in each iteration according to a predefined scheme. Furthermore, a fixed time limit determines how many iterations of the main loop of the hybrid may be performed. At the end, when the given time limit is exceeded, L is applied to the best solution in S using maximum accuracy $A(p_{\max})$.

4. SIMULATED HEATING

The idea of simulated heating [19] can be summarized as follows: Instead of keeping the PLSA parameter value p constant for the entire optimization process, we start with a low value (leading to a low $C(p)$ and $A(p)$) and increase it at certain points in time. The intention is to focus on the global search at the beginning and to find promising regions first; for this phase, the PLSA L runs with low accuracy. Later, more time is spent by L in order to improve the solutions found so far and/or to assess them more accurately. As a consequence, fewer global search operations are possible during this phase of optimization. Since $A(p)$ is steadily increased in the course of time, we use the term simulated heating by analogy to simulated annealing where a temperature is continuously decreased according to a given cooling scheme.

Definition: Let $H : \aleph \rightarrow R$, where R is the parameter domain of a PLSA A, be a function that specifies the PLSA parameter value $p = H(t)$ to be used during local search at iteration t of a GSA/PLSA hybrid involving A. The function H is called a **heating scheme** of the associated hybrid search algorithm if it is monotonically non-decreasing — that is, for all $t_1, t_2 \in \aleph$ with $t_1 \le t_2$, we have $H(t_1) \le H(t_2)$.

We can distinguish heating schemes according to whether they are computed at compile-time (*static*) or at run-time (*dynamic*). Another orthogonal classification is whether i) an equal number of iterations is performed for each parameter in the set $R' \subseteq R$ of parameters considered during optimization, or ii) constant optimization time is spent for each member of R'. We call the first class of schemes FIP (*fixed number of iterations per parameter*) and the second class FTP (*fixed time per parameter*). With FTP, the optimization time is spread equally for each parameter. As a consequence, the number of iterations that may be performed for each fixed parameter decreases for higher values of p.

5. OBJECTIVE FUNCTION

The quality function F will take as input a voltage vector V and a period graph PG. Each node execution time is scaled by its corre-

Input: N (size of solution candidate set)
$T_{\max}$ (maximum time budget)
Output: s (best solution found)
Step 1: *Initialization:* Create an initial multi-set S containing N randomly generated solution candidates. Set $T = 0$ (time used) and $t=0$ (iterations performed).
Step 2: *Parameter adaptation:* Choose $p \in R$ according to a given heating scheme H: $p = H(t)$.
Step 3: *Local search:* Apply L with parameter p to each $s \in S$ and assign it a quality (fitness) $F(s)$.
Step 4: Set T = time elapsed since Step 1.
Step 5: *Termination:* If $T > T_{\max}$ then go to Step 7.
Step 6: *Global search:* Based on S and F, generate a new set S' of solution candidates using G. Set $S = S'$ and increase the iteration counter t. Go to Step 2.
Step 7: *Output:* Apply L with parameter $p_{\max}$ to the best solution in S regarding F; the resulting solution s is the outcome of the algorithm.

Figure 1. Global/local search hybrid

1. For clarity, we assume here that p is a scalar rather than a vector of parameters.

sponding voltage. Let M be the maximum cycle mean of PG with the node voltages scaled by V. The energy consumed by each task (node) is equal to the power times its execution time. The average power is the total energy divided by the period $T_{solution}$ (reciprocal of M). If $T_{solution}$ violates the period constraint ($T_{solution} > T_{constraint}$), the power consumption is multiplied by a large penalty factor $\exp(100(T_{solution} - T_{constraint}))$.

6. LOCAL SEARCH ALGORITHMS

We implement two different local search strategies - hill climbing and Monte Carlo. The benefit of using a local search algorithm is that within a restricted voltage range we can use the period graph estimator for the throughput, which is much faster than performing a simulation. For the hill climbing algorithm, we define a parameter δ which is the voltage step, and we define a resimulation threshold r, which is the maximum amount that the voltage vector can vary from the point at which the period graph was calculated. The algorithm is run for I iterations. So for this case, the PLSA L has 3 parameters I, r, and δ. One iteration of local search consists of changing the node voltages, one at a time, by $\pm\delta$, and choosing the direction in which the objective function is minimized. From this, the worst case cost $C(I, r, \delta)$ for I iterations would correspond to evaluating the Objective function $3I$ times, and resimulating $(I/\lceil r/\delta \rceil)$ times. For our experiments we fix I and δ and define the local search parameter (from section 3) $p = 1/r$. Then for smaller p (corresponding to larger resimulation threshold) the voltage vector can move a greater distance before a new simulation is required. For a fixed number of iterations I in the local search, a smaller p will correspond to a shorter running time $C(p)$ for $L(p)$. The accuracy $A(p)$ will be lower, since the accuracy of the period graph estimate decreases as the voltage vector moves farther away from the simulation point.

In the Monte Carlo algorithm, we generate N random voltage vectors within a distance D from the input vector. For all points within a resimulation threshold r, we use the period graph to estimate performance. We use a greedy strategy to evaluate the remaining points. Specifically, we select one of the remaining points at random, we simulate and construct a new period graph, and we then use the resulting estimator to evaluate all points within a distance r from this point. If there are points remaining after this, we choose one, resimulate, and repeat. For our experiments we fix N and D and define the local search parameter $p = 1/r$. As for the hill climbing local search, smaller values of p correspond to shorter run times and less accuracy for the Monte Carlo local search. The pseudo-code for the local search algorithms is given below in Figures 2 and 3.

7. SIMULATED HEATING ALGORITHM

For the following experiments, we consider a GSA/LSA hybrid using a dynamic heating scheme and assume that the parameter domain R takes the form of an interval $[p_{min}, p_{max}]$, and that n parameters are uniformly chosen over R to form the set R' of parameters to consider. The paramter p is initially set at p_{min}. It is increased when for a user-given time of T_{stag} seconds the quality of the best solution in the solution candidate set has not improved (stagnation). As a consequence, for each parameter a different number of iterations may be considered until the stagnation condition is fulfilled. The amount of increase is based on the time T_p spent with the last parameter, the time limit for the overall optimization T_{max}, the current elapsed time T_{cur}, the last parameter p, and p_{max}. If $T_{cur} > T_{max}$, the optimization is terminated. Otherwise, let $N_{step} = \lceil (T_{max} - T_{cur})/T_p \rceil$. Then p is increased by $(p_{max} - p)/N_{step}$.

8. EXPERIMENTS

We ran experiments with the following application graphs: fft1, fft2, fft3, karp10, qmf4, and meas. The fft graphs are different implementations of the fast fourier transform from [10], and contain 28 nodes. Karp10 refers to the Karplus-Strong music synthesis algorithm with 10 voices (21 nodes), qmf4 is a quadrature mirror filter bank (14 nodes), and meas is a measurement application (14 nodes). From the application graph, we constructed a schedule using the dynamic level scheduling algorithm given in [17]. We used the estimate of 10% for the power overhead and 10µs for the switching time from [14]. The global search algorithm was an incremental genetic algorithm using one-point crossover, mutation probability of 0.1, crossover probability 0.9, and population size 50. This genetic algorithm is similar to those based on the GENITOR [18] model. It uses overlapping populations, with the worst two individuals being replaced each generation. We compare results obtained for a fixed compile time using simulated heating with results obtained without using simulted heating (keeping PLSA parameter p constant for the entire optimiztion process). We use the dynamic heating scheme outlined in section 7. The throughput constraint was calculated by setting all the task volt-

Input to hill climbing local search: Voltage vector V, δ, r, I, period graph PG.

```
Pseudo -code for Hill Climbing LocalSearch(V, δ, r, I, PG):
    copy V into another vector V_init
    for (k = 0; k < I; k = k + 1)
      for(i = 0; i < length(V); i = i + 1)
        V0 = V[i]
        V[i] = V0(1 + δ)
            f1 = F(V, PG)
        V[i] = V0(1 – δ)
            f2 = F(V,PG)
        V[i] = V0
            f = F(V, PG)
        if (f1 < f)
            V[i] = V0(1 + δ)
        else if (f2 < f)
            V[i] = V0(1 – δ)
        end if
      end for

      distance = Vector distance between V and V_init
      if (distance > r)
        Resimulate(PG)
```

Figure 2. Hill climbing local search

ages to a fixed reference voltage of 5 volts and calculating the period. The average power was calculated by summing the energy for each task (power from equation 2 multiplied by the task execution time) and dividing by the period. The total compile time alloted to the optimization was 1200 seconds.

```
Pseudo - code for Monte Carlo Local Search
Input: Voltage vector V, N, r, D where N is the number of random vectors to generate, r is the resimulation threshold, D is the distance within which the random vectors are generated.
Generate N random vectors within a distance D from V
Create list<vector> L1, L2, L3
Initially, L1 holds all N vectors, L2 and L3 are empty
while (size(L3) < N)
    Simulate at V and create period graph
    while (L1 is not empty)
        Remove a voltage v from L1
        Calculate distance d from v to V
        if (d < r)
            Evaluate v using period graph estimate
            Place v in L3
        else
            Place v in L2
        end if
    end while
    Swap L1 and L2
    Pick first element v from L1, and set V = v
```

Figure 3. Monte Carlo local search

Table 2. No simulated heating, hill climbing local search, compile time 1200 seconds

Application	**PLSA param** p	P/P_0
fft1	1.7	0.78
fft1	5.0	0.93
fft2	2.2	0.71
fft2	5.0	0.87
fft3	2.5	0.67
fft3	5.0	0.75
karp10	1.67	0.66
karp10	5.0	0.82
qmf4	3.3	0.62
qmf4	5.0	0.79
meas	4.7	0.61
meas	5.0	0.64

Table 1 shows the results for a hill climbing local search keeping PLSA parameter p fixed during the entire optimization (no simulated heating). Optimization runs were performed for values of p between 1.1 and 5.0. The first entry for each application corresponds to the parameter p which yielded the best results. The next entry corresponds to the highest accuracy parameter (5.0 in this case). The heading P/P_0 refers to the ratio of the average power of the best solution found by the optimization to the initial average power. Table 2 shows results under the same conditions for Monte Carlo local search. From tables 1 and 2, it can be seen that when not using simulated heating, the optimum parameter p is different in general for different applications and different local search techniques. It is hard to predict in advance which parameter should be used. For these applications, the hill climbing local search produced slightly better results overall.

Table 1. No simulated heating, Monte Carlo local search, compile time 1200 seconds

Application	**PLSA param** p	P/P_0
fft1	1.67	0.81
fft1	5.0	0.96
fft2	2.4	0.74
fft2	5.0	0.88
fft3	1.67	0.74
fft3	5.0	0.94
karp10	2.7	0.69
karp10	5.0	0.79
qmf4	3.4	0.71
qmf4	5.0	0.75
meas	4.0	0.66
meas	5.0	0.72

Table 3. Dynamic heating, Monte Carlo local search, $T_{\text{stag}} = 200$ seconds, $p_{\min} = 1.1$, $p_{\max} = 5$, compile time 1200 seconds

Application	P/P_0
fft1	0.72
fft2	0.65
fft3	0.62
karp10	0.58
qmf4	0.60
meas	0.63

Next, simulated heating experiments were performed with the dynamic heating scheme, using a range of parameters from $p_{min} = 1.1$ to $p_{max} = 5$ with $T_{stag} = 200$ seconds. Table 3 summarizes results for the Monte Carlo local search, and table 4 summarizes results for the hill climbing local search. Comparing tables 3 and 4 with tables 1 and 2, it can be seen that the dynamic heating scheme produced better overall results for fixed compile time than those obtained by keeping p constant.

9. CONCLUSION

In this paper, we have explored the efficient exploitation of dynamic voltage scaling technology to minimize the average power consumption of an embedded multiprocessor system under a given throughput constraint. To address the complex underlying design space, we have explored hybrid global/local search strategies for this problem using the previously-developed tools of period-graph-based performance estimation, and simulated heating for integrating parameterized local search algorithms (PLSAs) into global search. Our approach systematically allocates compile-time resources between the global search and parameterized local search processes — this is done by adaptively determining the accuracy/run-time settings with which successive PLSA invocations should be configured to attain maximum search efficiency. Our experiments show that 1) the best parameter setting for fixed-configuration PLSA optimization is highly application-dependent, and 2) even with the best (application-specific) configuration setting, fixed-configuration PLSA use is outperformed by our dynamically-reconfigured PLSA approaches, which yield significantly reduced power consumption.

10. REFERENCES

[1] N. K. Bambha and S. S. Bhattacharyya. A joint power/performance optimization technique for multiprocessor systems using a period graph construct. In *Proceedings of the International Symposium on Systems Synthesis*, pages 91-97, September 2000.

[2] T. Burd and R. Broderson, "Design Issues for Dynamic Voltage Scaling", In *Proceedings of 2000 International Symposium on Low Power Electronics and Design*, pages 76-81, July 2000.

Table 4. Dynamic heating, hill climbing local search, $T_{stag} = 200$ seconds, $p_{min} = 1.1$, $p_{max} = 5$, compile time 1200 seconds

Application	P/P_0
fft1	0.75
fft2	0.61
fft3	0.59
karp10	0.54
qmf4	0.56
meas	0.58

[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid State Circuits*, 27(4):473-484, 1992.

[4] J. M. Chang and M. Pedram, "Register allocation and binding for low power," *Design Automation Conf.*, June, 1995.

[5] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):889-899, October 1998.

[6] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," in *Proceedings of the International Symposium on Low Power Design*, April 1995.

[7] D. E. Goldberg and S. Voessner. Optimizing global-local search hybrids. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 220-228, 1999.

[8] L. Goodby, A. Orailoglu, and P. M. Chau, "Microarchitectural synthesis of performance-constrained low-power VLSI designs," in *Proceedings of the International Conference on Computer Design*, Oct. 1994.

[9] H. Ishibuchi and T. Murata. Multi-objective genetic local search algorithm. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 119-124, 1996.

[10] A. Kahn, C. McCreary, J. Thompson, and M. McArdle, "A Comparison of Multiprocessor Scheduling Heuristics", in *Proceedings of 1994 International Conference on Parallel Processing*, vol. II, pages 243-250, 1994.

[11] E. A. Lee and S. Ha. Scheduling strategies for multiprocessor real time DSP. In *Proceedings of the Global Telecommunications Conference*, November 1989.

[12] Y. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the Design Automation Conference*, 1995.

[13] D. Marculescu, "On the Use of Microarchitecture-Driven Dynamic Voltage Scaling", *Proceedings of ISCA* 2000.

[14] T. Pering, T. Burd, and R. Broderson, "The simulation and Evaluation fo Dynamic Voltage Scaling Algorithms", In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 76-81, August 1998.

[15] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," in *Proc. Intl. Conf. Computer Design*, Oct. 1994.

[16] M. Ryan, J. Debuse, G. Smith, and I. Whittley. A hybrid genetic algorithm for the fixed channel assignment problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1707-1714, 1999.

[17] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2): 75-87, February 1993.

[18] D. Whitley. The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In *Proceedings of 3rd International Conference on Genetic Algorithms*, pages 116-121. Morgan Kauffman, 1989.

[19] E. Zitzler, J. Teich, and S. S. Bhattacharyya. Optimizing the efficiency of parameterized local search within global search: A preliminary study. In *Proceedings of the Congress on Evolutionary Computation*, pages 365-372, San Diego, California, July 2000.