MODLEX: A Multi Objective Data Layout EXploration Framework for Embedded Systems-on-Chip

T.S. Rajesh Kumar C.P. Ravikumar
Texas Instruments India Ltd.
C.V. Raman Nagar
Bangalore, 560 091, India

{tsrk,ravikumar}@ti.com

R. Govindarajan
Supercomputer Education & Research Centre
Indian Institute of Science
Bangalore, 560 012, India

govind@serc.iisc.ernet.in

Abstract— The memory subsystem is a major contributor to the performance, power, and area of complex SoCs used in feature rich multimedia products. Hence, memory architecture of the embedded DSP is complex and usually custom designed with multiple banks of single-ported or dual ported on-chip scratch pad memory and multiple banks of off-chip memory. Building software for such large complex memories with many of the software components as individually optimized software IPs is a big challenge. In order to obtain good performance and a reduction in memory stalls, the data buffers of the application need to be placed carefully in different types of memory . In this paper we present a unified framework (MODLEX) that combines different data layout optimizations to address the complex DSP memory architectures. Our method models the data layout problem as multi-objective Genetic Algorithm (GA) with performance and power being the objectives and presents a set of solution points which is attractive from a platform design viewpoint. While most of the work in the literature assumes that performance and power are non-conflicting objectives, our work demonstrates that there is significant trade-off (up to 70%) that is possible between power and performance.

I. Introduction

Today's VLSI technology allows to integrate tens of processor cores on the same chip along with embedded memories, application specific circuits, and interconnect switches. Such devices are being used in feature-rich multimedia applications such as mobile cameras. The key to the success of such systems-on-chip (SoC) which are targeted for commodity market will be to achieve low cost, high performance, and low power dissipation.

One of the key factors that drives the cost and power dissipation of an embedded system-on-chip is the memory architecture. Studies have shown that in many DSP based embedded systems, the area and power consumed by the memory subsystem is up to 10 times that of the data path, making memory a critical component of the design. Further, the memory subsystem constitutes a large part (typically up to 70%) of the silicon area for the current day SoC and it is expected to go up to 94% in 2014 [11]. The main reason for this is that embedded memory has a relatively small per area design cost in terms of both man power and time to market. Another reason is related to power consumption: the heat dissipated per area is lower for memories than that of logic. Hence memory can be used to add functionality with a smaller impact on system heat [12]. As the number of transistors/package keeps increasing along with the increasing share of memory, the future system's performance will depend even more on memory.

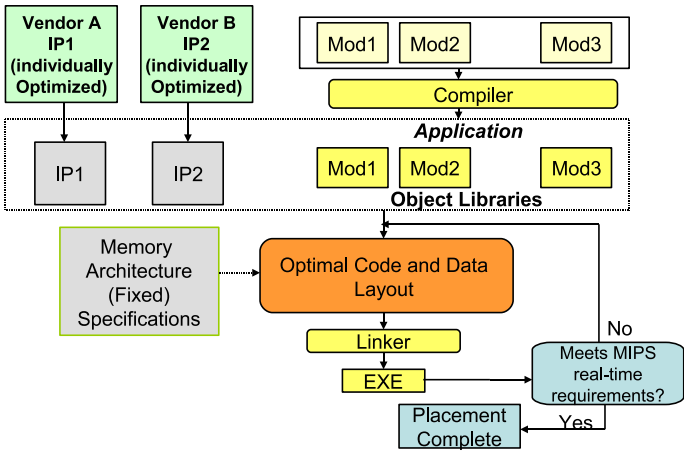

Fig. 1. Embedded Application Development Flow

The memory architecture of embedded DSPs are complex and custom designed to improve the run-time performance and power consumption. The on-chip memory can be SRAM, and/or ROM and/or embedded DRAM and similarly the off-chip memory can be DRAM and/or SRAM. The on-chip memory referred as Scratch Pad memory is organized into multiple memory banks to facilitate multiple simultaneous data accesses. Further on-chip memory bank can be a single-access RAM (SARAM) or a dual-access RAM (DARAM), to provide single or dual access to the same memory bank in a single cycle. This memory architecture is applicable mainly for

1-4244-0630-7/07/$20.00 ©2007 IEEE.

DSPs rather than for microcontrollers as DSPs have multiple on-chip busses and multiple address generation units to service the higher bandwidth needs of DSP applications. Also the on-chip memory banks can be of different sizes. Smaller memory banks consume lesser power per access than the larger memories. This architecture presents opportunities to optimize power consumption by placing the most frequently accessed data variables in the smallest bank size.

The application program in a modern embedded system is complex and has many multimedia components like MP3, AAC and MIDI. This necessitates an IP reuse methodology, where software modules developed and optimized independently by different vendors are integrated. Figure 1 explains the typical flow in embedded application development. This integration is a very challenging job and must be accomplished with short design time, achieving high performance and low cost and power. In order to obtain good performance and a reduction in memory stalls, the data buffers of the application need to be placed carefully in different types of memory; this is known as the **data layout problem**. Typically the data layout is performed manually and hence takes a significant amount of time (approximately 1-2 man months).

The data layout optimization methods varies significantly for applications built for microcontrollers (MCU) and Digital Signal Processors (DSPs) due to the following reasons: (a) DSP applications are more data dominated than the control software executed in MCU. Memory bandwidth requirements for DSP applications range from 2 to 3 memory accesses per processor clock cycle. While that for MCU is at best 1. at best provide only one memory access per cycle. (b) The DSP software for critical kernels is developed mostly as hand optimized assembly code. Whereas the MCU SW is developed in high level languages. Hence compiler based optimizations may not be directly applicable for the DSP kernels. In this paper we consider data layout optimizations for DSP applications and do not consider optimizations that require code modifications.

A. Related Work

The data layout problem [8, 5, 4, 7, 1, 9, 2] has been widely researched in the literature both from a performance and power perspective individually. Further most of the work has addressed the data layout problem for specific memory architectures. Following works addresses the data layout problem with an objective to improve performance. The authors of [5, 4] have formulated data layout problem as Integer Linear Programming (ILP), where they allocate data variables to memory banks based on the access frequency of data variables; the memory architecture they consider is on-chip memory and off-chip memory with different latencies. In [8] the data layout problem is solved for an on-chip memory architecture that has both cache and scratch pad RAM. [7, 1] have addressed the partitioning of simultaneously accessed data variables in multiple single-port memory banks to avoid memory bank conflicts. [10] handles the data partitioning problem for self-conflicting data variables (variable that are accessed multiple times in the same cycle and will result in memory conflict if placed in single-port memory). It proposes partial duplication of data variables to avoid using expensive dual-port memories. [3] proposes a low-energy memory design method, referred as VAbM, that optimizes the memory area by allocating multiple memory banks with variable bit-width to optimally fit the application data. In [2], Benini et al., present a data layout method that aims at energy reduction. The main idea of this paper is to use the access frequency of memory address space as starting point and design smaller (larger) bank size for the most (least) frequently accessed memory addresses.

All these optimizations are very effective individually for the class of memory architecture they target. However for a complete data layout approach one has to combine many/all of these approaches to be able to comprehensively address the problem. Also it may not be optimal to just combine different optimizations and an integrated approach will yield a better result. All the data layout work in the literature to the best of our knowledge either optimizes run-time performance or minimizes power and assumes that power and run-time performance are non-conflicting objectives. But this assumption is not valid for DSPs, where data layout optimization will result in a trade-off between power and performance. In this paper we show that there is a trade-off between performance and power for a given application and a given memory architecture. We propose MODLEX, a Multi Objective Data Layout EXploration framework based on Genetic Algorithm.

The main contributions of this paper are (a) combining different data layout optimizations into an unified framework that can be used for the complex embedded DSP memory architectures. Even though we target the DSP memory architectures, our method also works for microcontrollers as well. (b) Model the data layout problem as multi-objective Genetic Algorithm (GA) with performance and power being the objectives. Our method optimizes the data layout for power and run-time and presents a set of solution points that are optimal with respect to power and performance. (c) Most of the work in the literature assumes that performance and power are non-conflicting objectives with respect to data allocation. But we show that there is significant trade-off (up to 70%) that is possible between power and performance.

The remainder of this paper is organized as follows. In Section 2, the problem definition. In Section 3, we present our MODLEX framework. In Section 4, we present the experimental methodology and results. Finally in Section 5, we conclude and outline some of the future work.

II. Problem Statement

Given a memory architecture with m on-chip SARAM memory banks, n on-chip DARAM memory banks, and an off-chip memory. The size of each of the on-chip memory bank and the off-chip memory is fixed. The access time for the on-chip memory banks is one cycle, while that for the off-chip memory is l cycles. Given an application with d data sections. The simultaneous access requirement of multiple arrays is captured by means of a two-dimensional matrix C where

C_{ij} represents the number of times data sections i and j are accessed together in the same cycle in the execution of the entire program. We do not consider more than two simultaneous accesses, as the embedded DSP core typically supports only up to two accesses in a single cycle. If data sections i and j are placed in two different memory modules, then these conflicting accesses can be satisfied simultaneously without incurring stall cycles. C_{ii} represents the number of times two accesses to data section i is made in the same cycle. Self-conflicting data sections need to be placed in DARAM memory banks, if available, to avoid stalls. The objective of the data layout problem is to place the data sections in memory modules such that the following are minimized:

- Number of memory stalls incurred due to conflicting accesses (parallel and self conflicts) and the additional cycles incurred in accessing off-chip memory
- The total memory power calculated as the sum of the memory power of all memory banks. Memory power of each of the banks is computed by multiplying the number of read/write accesses and the power per read/write access

Since the optimization problem involves multiple objectives the output will be a set of Pareto optimal points. The pareto optimal condition can be formally stated as follows. Let x and y be two n-tuples representing the values of the objective criteria such as execution time, energy consumed and area. Assume that all objectives are minimizing functions. We say $x <_p y$ if

$$(x <_p y) \Longleftrightarrow (\forall_i)(x_i <= y_i) \wedge (\exists i)(x_i < y_i)$$

Using the partial relation $<_p$, we can say if $x <_p y$ then x dominates y or y is a dominated point. If the set of all dominated points are removed from the set of all points in the design, we get the non-dominated set or the Pareto-optimal design points.

III. MODLEX: Multi Objective Data Layout EXploration

A. Method Overview

We formulate the data layout problem as a multi-objective GA [6] to obtain the set of *Pareto optimal* design points. The multiple objectives are minimizing memory stall cycles and memory power. Figure 2 explains our MODLEX framework. Application profile information and a logical memory architecture are taken as inputs. The logical memory architecture contains the number of memory banks, memory bank sizes, memory bank types (single-port, dual-port), and memory bank latencies. The logical memory to physical memory map is explained in the following section. The core engine of the framework is the multi-objective data layout, which is implemented as a Genetic Algorithm (GA). The data layout block takes the application data and the logical memory architecture as input and outputs a data placement. The cost of data placement in terms of memory stalls and memory power are computed for a given data placement. The overall fitness function used by the

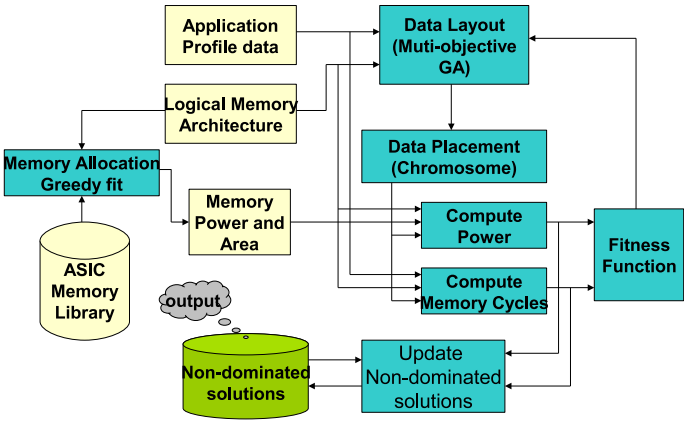

Fig. 2. Embedded Application Development Flow

GA is a combination of memory stall cost and memory power cost. Based on the fitness function the GA evolves by selecting the fittest individuals (the data placements with the lowest cost) to the next generation. Since the fitness function contains multiple objectives, the fitness function is computed by ranking the chromosomes based on the non-dominated criteria (detailed explanation provided in Section C). This process is repeated for a maximum number of generations specified as a input parameter.

B. Mappping Logical Memory to Physical Memory

To get the memory power and area numbers, the logical memories have to be mapped to physical memory modules available in a ASIC memory library for a specific technology/process node. A number of approaches have been proposed for mapping logical memory to physical memories [14, 13]. Each of the logical memory bank can be implemented physically in many ways. For example, for a logical memory bank of 4K*16 bits can be formed with two physical memories of size 2K*16 bits or four physical memories of size 2K*8 bits. The memory allocation problem in general is NP-Complete [14]. However since the logical memory architecture is already organized as multiple memory banks, most of the mapping turns out to be a direct one to one mapping. In this paper a simple greedy technique is used to perform the mapping of logical to physical memory with the objective of reducing silicon area. The memory modules are sorted based on area/byte and the smallest area/byte physical memory is taken to form the required logical memory bank size.

C. Genetic Algorithm Formulation

To map an optimization problem to the GA framework, we need the following: chromosomal representation, fitness computation, selection function, genetic operators, the creation of the initial population and the termination criteria.

C.1 Chromosome Representation

For the data memory layout problem, each individual chromosome represents a memory placement. A chromosome is a vector of d elements, where d is the number of data sections. Each element of a chromosome can take a value in (0 .. m), where 1..m represent on-chip memory banks (including both SARAM and DARAM memory banks) and 0 represents off-chip memory. Thus if the element i of a chromosome has a value k, then the data section is placed in memory bank k. Thus a chromosome represents a memory placement for all data sections. Note that a chromosome may not always represent a valid memory placement, as the size of data sections placed in a memory bank k may exceed the size of k. Such a chromosome is marked as invalid and assigned a low fitness value.

C.2 Chromosome Selection and Generation

The strongest individuals in a population are used to produce new off-springs. The selection of an individual depends on its fitness, an individual with a higher fitness has a higher probability of contributing one or more offspring to the next generation. In every generation, from the P individuals of the current generation, M more offspring are generated, resulting in a total population of $(P + M)$. From this P fittest individuals survive to the next generation. The remaining M individuals are annihilated. Crossover and mutation operators are implemented in standard way.

C.3 Fitness Function and Ranking

For each of the individuals, the fitness function computes M_{pow} and M_{cyc}. The value of M_{cyc} is computed as follows. Each of the chromosomes represents a data placement. The number of memory stalls incurred in a memory bank j can be computed by looking at the number of conflicting data sections that are kept in j. For each pair of the conflicting data sections, the number of conflicts is given by the conflict matrix. The total memory stalls incurred in bank j can be computed by multiplying the number of conflicts and the bank latency. The total memory stalls for the complete memory architecture is computed by summing all the memory stalls incurred by all the individual memory banks.

Memory Power corresponding to a chromosome is computed as follows. If p_j is the power per access of memory bank j, and AF_i is the number of times data variable i is accessed then the total power P_j for memory bank j is

$$P_j = \sum_{i=1}^{d} p_j * AF_i * I_{ij}$$

where $I_{ij} = 1$, if data variable i is placed in memory bank j else $I_{ij} = 0$. Thus the total power P_t for all the memory banks is the sum of all the individual memory bank's power.

$$P_t = \sum_{j=1}^{N_b} P_j$$

where N_b is the total number of banks including off-chip memory.

Once the memory cost and memory cycles are computed for all the individuals in the population, individuals are ranked according to the *Pareto optimality* conditions given in the following Equation. Let (M^a_{pow}, M^a_{cyc}) and (M^b_{pow}, M^b_{cyc}) be the memory power and memory cycles of chromosome A and chromosome B, A dominates B if the following expression is true.

$$((M^a_{pow} < M^b_{pow}) \wedge (M^a_{cyc} \leq M^b_{cyc}))$$
$$\vee((M^a_{cyc} < M^b_{cyc}) \wedge (M^a_{pow} \leq M^b_{pow}))$$

The ranking process in multi-objectctive GA proceeds as follows. All non-dominated individuals in the current population are identified and flagged. These are the best individuals and assigned a rank of 1. These points are then removed from the population and the next set of non-dominated individuals are identified and ranked 2. This process continues until the entire population is ranked. Fitness values are assigned based on the ranks. Higher fitness values are assigned for rank-1 individuals as compared to rank-2 and so on. This fitness is used for the selection probability. The individuals with higher fitness gets a better chance of getting selected for reproduction.

One of the common problems in multi-objective optimization is solution diversity. Basically the search path may progress towards only certain objectives resulting in design points favoring those objectives. Solution diversity is very critical in order to get a good distribution of solutions in the Pareto-optimal front. To maintain solution diversity, the fitness value is reduced for solution that has many neighboring solutions. To maintain solution diversity we use the sharing function method explained in [15]. The GA must be provided with an initial population that is created randomly. In our implementation we have used a fixed number of generations as the termination criterion.

IV. Experimental Results

A. Experimental Methodology

We have used Texas Instrument's TMS320C55X and Texas Instrument's Code Composer Studio (CCS) environment for obtaining the profile data and also for validating the data-layout placements. We have used 3 different applications from the multimedia and communications domain as benchmarks The kernels of the applications are developed in hand-optimized assembly code. The applications are compiled with the C55X processor compiler and assembler. The profile data is obtained by running the compiled executable in a cycle accurate software simulator. For obtaining conflict data we used one large bank of single-access RAM that fits the application data size. This configuration is selected because this does not resolve any of the parallel or self conflicts. The output profile data contain (a) frequency of access for all data sections (b) the conflict matrix. The other inputs required for our method

is the application data section sizes, which are obtained from the C55X linker. We have used TI's ASIC memory library for the memory allocation step. The area and power numbers are obtained from the ASIC memory library.

B. Experimental Results

This section presents the experimental results on the multi-objective data layout. We consider a set of 6 different logical memory architecture listed in Table I. In Table I the corresponding physical memory architecture and the normalized area [1] required by the physical memory for the different architectures are given. The output is a set of Pareto optimal points that presents trade-off between power and memory cycles for a given memory area. From the table it can be observed that the memory area increases with the DARAM size and the number of banks. Note that the architecture A4 has more memory area than A5 even though it has only half of the A5's DARAM. This is due to the higher number of banks in A5.

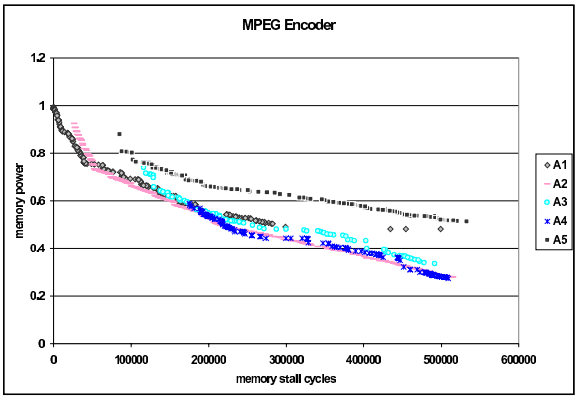

Fig. 3. MPEG Enoder: Performance-Power trade-off

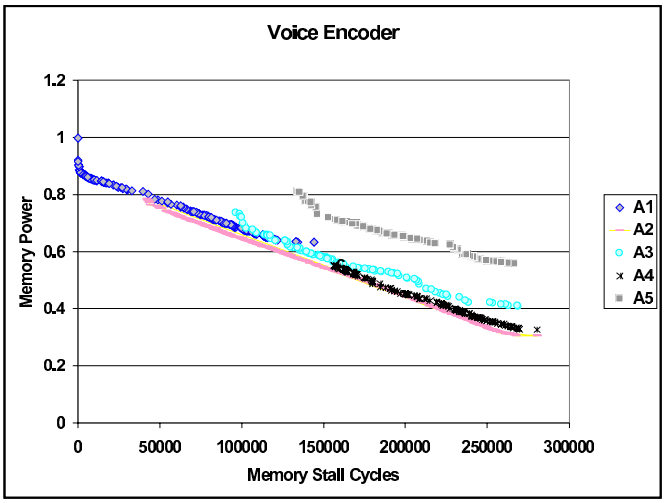

Fig. 4. Voice Enoder: Performance-Power trade-off

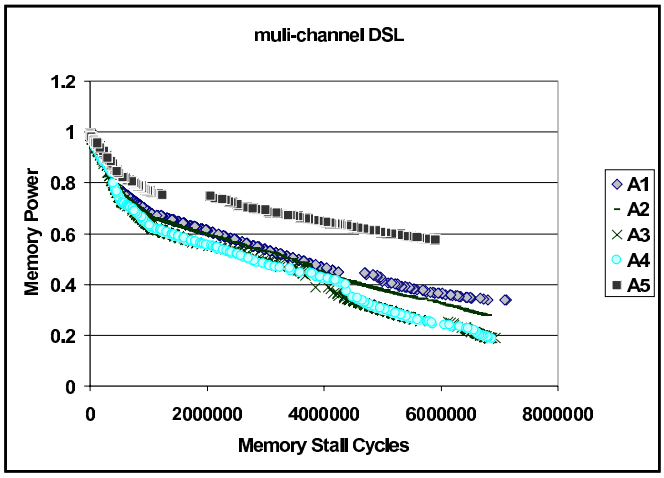

Fig. 5. DSL: Performance-Power trade-off

Figures (3, 4 and 5) shows the set of non-dominated points each corresponds to a *Pareto Optimal* data layout for the 3 applications for architectures A1-A5. It should be noted that the non-dominated points seen by the multi-objective GA are only near optimal, as the evolutionary method may result in another design point in future generations that could dominate. It can be observed from Figures (3, 4 and 5) that the architecture A1 gives the best performance for all the three applications as it offers higher bandwidth due to the larger DARAM size. [2]. It resolves almost all the memory conflicts and latencies. But this solution point is also the worst in power. Also observe that the architecture A2 performs as well as A1 for MpegEnc even though A2 has only 33% DARAM. Also observe that A2 is not performing as well for voice encoder application as it did for Mpeg. This is because the Mpeg encoder has more parallel conflicts and lesser self-conflicts than voice encoder and

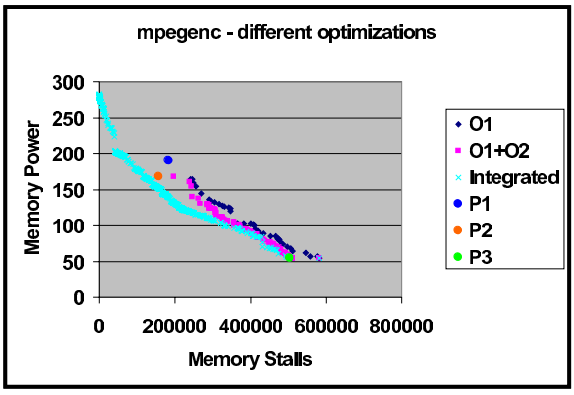

Fig. 6. Individual Optimizations vs Integrated

[1] As the ASIC library is proprietary to Texas Instruments, we present only the normalized power and area numbers

[2] We recommend this page be viewed in color for better readability of the graphs

Arch no	Architecture		memory area
	Logical memory	physical memory	
A1	2x8K(SARAM) 20x4K(DARAM)	2x8192(1P) 20x4096(2P)	1
A2	18x4K(SARAM) 8x4K(DARAM)	18x4096(1P) 8x4096(2P)	0.91
A3	8x4K(SARAM) 1x32K(SARAM) 8x4K(DARAM)	8x4096(1P) 1x32K(1P) 8x4096(2P)	0.82
A4	8x2K(SARAM) 4x4K(SARAM) 3x16K(SARAM) 4x4K(DARAM)	8x2048(1P) 4x4096(1P) 3x16K(1P) 4x4096(2P)	0.77
A5	2x32K(SARAM) 8x4K(DARAM)	2x32K(1P) 8x4096(2P)	0.72
A6	8x2K(SARAM) 4x4K(SARAM) 1x16K(SARAM) 4x4K(DARAM) 1x32K(Off-Chip)	8x2048(1P) 4x4096(1P) 1x16K(1P) 4x4096(2P) 1x32K(1P)	0.57

TABLE I
MEMORY ARCHITECTURES USED FOR DATA LAYOUT

also A3 has large number of banks that resolves higher number of parallel conflicts. With only a small increase in area compared to A5, A3 can achieve much better performance than A5. This is due to the higher number of banks in A3 that resolves more parallel conflicts. Also observe the wide range of trade-off available between power and performance for all the applications. This is very useful for application engineers and system designers from a platform design viewpoint. It takes 18 minutes to obtain one Pareto Optimal plot for an application. This run-time is approximately same for all the application.

Figure 6 presents the results for mpeg for the memory architecture A6 explained in Table I. There are six different plots and each plot represents a specific data layout optimization. Plot *O1* corresponds to performing just on-chip/off-chip data partition similar to the approach in [5, 4]. Plot *O2* represents performing *O1* and resolving parallel memory conflicts by utilizing only multiple memory banks as in [7, 1]. Plot *O3* corresponds to the integrated approach that includes *O1*, *O2* and other optimizations like resolving self conflicts and exploiting multiple memory banks for power optimization. Observe that for the same memory architecture the integrated approach resolves almost all the memory stalls. Also both from power and performance perspective the integrated approach completely dominates the other two plots. The methods like [5, 4] will give power/performance close to point *P1* and the point *P2* corresponds to the works [7, 1, 10] and the data layout that optimizes power [2] is represented by point *P3*. From the results we can conclude that the integrated approach gives better solution points both with respect to power and performance. Also from the experimental results it can be concluded that there is a wide range of design points with respect to power and performance can be obtained from multi-objective data layout optimizations.

V. CONCLUSION AND FUTURE WORK

We presented our framework MODLEX: a Multi Objective Data Layout EXploration Framework. Our approach results in many (Pareto) optimal design points with respect to power and performance which are important from a platform design view point. We demonstrated that there is significant trade-off (up to 70%) that is possible between power and performance. As a future work we plan to extend our framework to explore memory architectures design space along with the data layout.

REFERENCES

[1] M.Ko, S.S.Bhattacharyya. Data Partitioning for DSP Software Synthesis. In Proceedings of the International Workshop on Software and Compilers for Embedded Processors, Sep-2003.

[2] L.Benini, L.Macchiarulo, A.Macii, M.Poncino. Layout Driven Memory Sunthesys for Embededed Systems-on-Chip. CASES 2002.

[3] Y.Cao, H.Tomiyama, T.Okuma, H.Yasuura, "Data Memory Design Considering Effective Bitwidth for Low Energy Embedded Systems," *ISSS 2002*.

[4] J.Sjodin, and C.Platen. Storage Allocation for Embedded Processors. CASES 2001.

[5] O. Avissar, R. Barua, D. Stewart. Heterogeneous Memory Management For Embedded Systems. CASES 2001, Nov 2001.

[6] D.E. Goldberg. Genetic Algorithms in Search, Optimizations and Machine Learning. Addison-Wesley, 1989.

[7] R. Leupers, D. Kotte. Variable Partitioning for Dual Memory Bank DSPs. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Salt Lake City (USA), May 2001.

[8] P.R. Panda, N.D.Dutt, and A.Nicolau. On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems. ACM Trans. Design Automation of Electronic Systems, 5(3):682–704, July 2000.

[9] T. S. Rajesh Kumar, R. Govindarajan, C. P. Ravikumar. Code and Data Layout For Embedded Systems. VLSI Design, Jan 2003.

[10] M.A.R. Saghir, P.Chow, C.G.Lee. Exploiting Dual Data-Memory Banks in Digital Signal Processors. In Proceedings of the 7th Intl Conference Architectural Support for Programming Languages and Operating Systems, pp.234-243, October 1996.

[11] International Technology Roadmap for Semiconductors, SEMATECH, 3101, Industrial Terrace Suite, 106 Austin TX 78758. *International Technology Roadmap for Semiconductors, 2001*.

[12] Eike Schmidt. Power Modelling of Embedded Memories. Phd Thesis, 2003.

[13] H.Schmit and D.Thomas, "Array Mapping in Behavioral Synthesis," *ISSS 95*.

[14] Pradip K.Jha and Nikil D.Dutt, "Library Mapping for Memories,", *EuroDesign 1997*.

[15] Deb.K., Goel T. Controlled elitest non-dominated sorting genetic algorithms for better convergence. In Porceedings of Evolutionary Multi-Criterion Optimization. 67-81, 2001.