System Level Distributed Cooperative Design of Media SoC Using Application Profiling

Dawei Wang
College of Computer Science, University of Defense Technology, Changsha, P.R.China
Email: daweiwang@nudt.edu.cn

Peng Zhao and Sikun Li
College of Computer Science, University of Defense Technology, Changsha, P.R.China
Email: pengzhao@nudt.edu.cn, lisikun@263.net.cn

***Abstract*—Heterogeneous multi-core architectures of System-on-Chip can support various embedded real-time applications well. SoC design is very complex for multi-fields experts to collaborate on application analysis, system decision and hw/sw co-design. However, existing SoC design methods and environments can only support human-computer interaction, ignoring the collaboration interaction between multi-field experts. This paper presents a collaborative approach of media SoC design using application profiling. We create a distributed collaborative design environment for system decision engineers, software designers, hardware designers and application developers. The method not only utilizes the advantages of application profiling for SoC design, but also pro-vides a framework for multi-field experts to work collaboratively. Experimental results show that the method effectively improves the quality and speed of media SoC design.**

***Index Terms*—Cooperative design, System-on-Chips, application profiling, system level design, embedded media**

I. INTRODUCTION

Existing EDA (Electric Design Automation) systems support single designer to de-sign with human-computer interaction only. With the emergence and rapid growth of CSCW (Computer Supported Collaborative Work) [1] [2] [3], EDA is not only the tool for functional computing, design decision and performance analysis, but also for multi-field experts to communicate and collaborate with human-human interaction.

Heterogeneous SoC design is very complex for multi-field experts to collaborate on application analysis, hw/sw co-design and system decision, while hw/sw partition is the key problem of media SoC design. Existing methods are well for hw/sw co-design, while not for distributed collaborative design with multi-field experts.

By analyzing typical embedded media applications, we find that some critical codes in the program, such as loop and function, contribute most system resource. So it is a must to find them for speed up and optimization. Besides, existing SoC design methods consider only specific algorithm realization, which get high performance on the specific algorithm on the loss of adaptability, thus they can't adapt various algorithms of media applications.

This paper puts forward a novel collaborative system level design approach of media SoC using application profiling. We developed DisCoFrame (Distributed Collaborative Framework) Environment [4], in which we build a distributed collaborative framework for multi-field experts. Orienting to typical MediaBench applications [5], we use application profiling to find critical blocks in program automatically, such as loop, function. Use these profiling results we build TFG (Task Flow Graph) to direct for hw/sw partition. Experimental results show improvement on design efficiency and quality.

The rest of the paper is organized into 6 sections. Section II overviews the previ-ous work. Section III introduces the collaborative design framework. In Section IV we present system level design of media SoC using application profiling. In Section V, we apply our approach on typical media benchmarks and give the obtained results. Finally, in Section VI we present the conclusions.

II. RELATED WORK

CSCW design is a novel product design method, which supports product designers and related experts at different locations to design the product by use of network and various computer aided tools. During this process, each user is aware of the existence of other users, and interacts with them.

Cutkosky [2] [3] advanced the concept of distributed cooperative design. Since then, researchers apply many techniques to implement distributed cooperative design, such as network and communication, distributed computing, computer supported cooperative design, agent and web service. Currently, the combination of intelligent agent and web service techniques has gained better application effect, and it supports distributed cooperative design of complex products efficiently.

Many profiling tools have been developed. FLAT can provide loop/function level information for a wide variety of platforms, which can support hw/sw partition, but it cannot support architecture design and compiler optimization [6]. Besides, it provides profiling information on execution time only. ATOM is specific to a particular mi-coprocessor family, which provides a toolset that let the user track a program's behavior [7]. Some analysis routines are inserted at interesting parts of the program and when the program is executed,

© 2009 ACADEMY PUBLISHER

analysis routines gather information about the program for further analysis. The tool gprof can only provide function level profiling and do not provide sufficient information about loop. HALT provides a flexible way to add routines to program produced by SUIF (Stanford University Intermediate Format) compiler, which provides the SUIF annotations for labeling the interesting parts of the program [8]. It also provides many analysis routines, which is helpful for code layout, instruction scheduling and register allocation.

While our approach is orienting to embedded media applications and we develop a tool called CBPT (Critical Block Profiling Toolset). By using SUIF toolkit we profile some application information, such as execution time, storage capacity, and natural loop analysis. The purpose of CBPT is to support some key work in SoC design, such as task partitioning, compiler optimization and architecture design. For collaborative system level design of media SoC, we in general assume that the styles of collaboration between multi-field experts are synchronous and remote. The experts, such as application developer and SoC designer, work on application analysis, hw/sw design and system decision making respectively.

III. Distributed Collaborative Design

A. Collaborative Design Framework

We use a Client/Server structure for distributed collaborative design, namely DisCoFrame. It supports data share, communication among group and the control of parallel operation. The client consists of some semi-intelligent agents for hw/sw designer, application developer to collaboratively design, as shown in Figure 1. It supports state watching, action reacting and TFG configuring. The server supports system decision maker to control the course of partition. It sees after global information storage, communication transfer, harmony and decision.

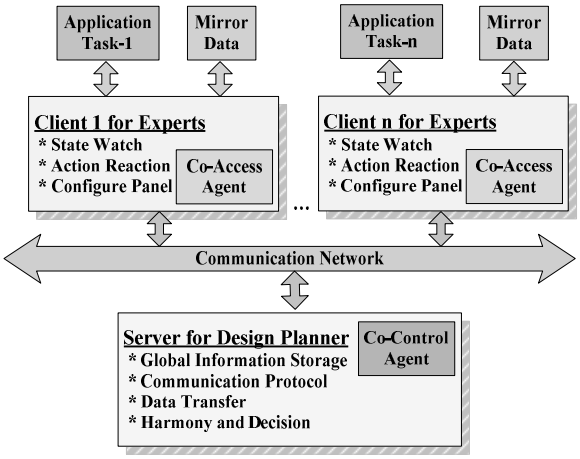

Figure 1. Distributed collaborative design framework for media SoC design.

In DisCoFrame, system decision maker on server informs the experts on clients to configure the parameters of TFG. Then every expert submits the configuration results of TFG to server by co-access agent. After receiving all the configuration results, the server starts ant colony optimization algorithm to do partition. The experts adjust the configuration according to the feedback of runtime simulation information from server until some objectives are met. The experts can stop partition if they find some constraints are not met in the process of partition. Besides, some requirement information is exchanged between application profiling, task partition and architecture design and co-compiler optimization by DisCoFrame.

B. Communication among Multi-experts

In CSCW system, experts want to know about not only the results of collaborative operation, but also the whole course of collaborative operation. They should know about the activity of expert, the change of state and the log of operation. Experts can adjust their behavior according to current actions and states of collaborative environment. So it is convenient for experts to collaborate with each other.

In DisCoFrame, we define the communication protocol of multi-field experts, which includes:

- Task protocols, such as query of current task, requisition for operation of task, user register, user login and logout.
- Data transfer protocol, including transfer mechanism of the configuration of TFG parameters, the operations of experts and the state of simulation.
- Parallel and cooperation control protocol, which maintain the consistency of global share data.
- Notification protocol, including aware mechanism of the state of tasks and the chatting among experts.

C. Parallel and Cooperative Control

Conflicts are inevitable for DisCoFrame supports multi-experts access global share data. To avoid the conflicts, we designed a logic clock based concentrative parallel control method. The basic idea is that: according to the global logic clock in server and the global exclusive integer allocated to each client, we define the order of operation events that send to the server. So it can ensure the consistency of the order of operation events running in client.

IV. Application-concerned System Level Design

A. Application-Concerned Design

We build media SoC hierarchical platform based on SoC hardware/software co-design environment developed by our research group.

Figure 2 shows the overall structure of media SoC, which comprises three main parts:

1). Hierarchical platform and the mapping, including system level design, transaction level design, RTL (Register Transfer Level) design, and design plan, co-synthesis. In system level, the main work is application profiling and task modeling. In transaction level, the main work is platform design space exploration, performance analysis and estimation. In RTL, the main work is FPGA (Field-Programmable Gate Array) simulation, RTOS (Real-Time Operating System) configuration and optimization. Design plan maps system level design onto transaction level design, the main work of

© 2009 ACADEMY PUBLISHER

which is task partitioning. Co-synthesis maps transaction level design onto RTL design, the main work of which is communication synthesis and compiler optimization. In each design level, modeling and simulation is needed.

2). Platform manager and maintain, including algorithm library and design tem-plate library in the field of embedded media process, which can support design reuse. Various application field platforms are saved in the field platform library. When de-sign new SoC, we select suitable field platform from the library according to the functions and characteristic of design. The platform has design templates of three design level and mapping results in hierarchical platform. After designing new SoC, platform manager and maintain can optimize the field platform and save well design results as new design template of field platform for further reuse.

3). Platform assist tools, including the tools of algorithm library, transaction level IP library, RTL IP library, modeling and simulation in each design level, design planning, co-synthesis, design space exploration and other commercial back-end tools. These tools can improve the design quality and efficiency of Media SoC hierarchical platform.

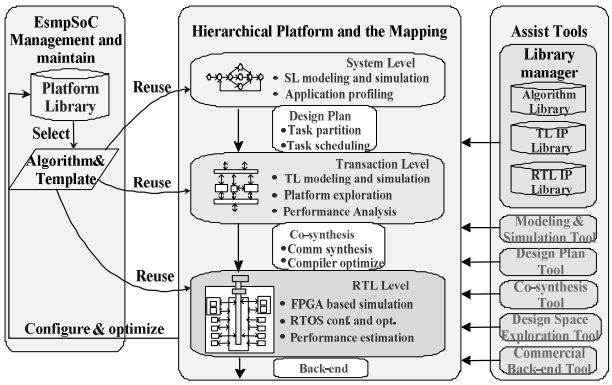

Figure 2. The overall structure of Media SoC hierarchical platform

We use a novel design method, AC-SLD (Application-Concerned System Level Design), in SoC hierarchical platform. This is a kind of application/platform co-design approach, which uses the idea of concern separation in MDA (Model Driven Architecture). AC-SLD concerns the applications in system level design, and use application profiling to direct for system level design, such as architecture design, task partitioning and scheduling, co-compiler optimization. AC-SLD can improve the suitability of SoC platform design to various applications.

It is a need to explain the relationship between AC-SLD and SoC hierarchical plat-form. AC-SLD belongs to the SoC hierarchical platform, which concerns the work about applications. Application profiling is in system level design, which is executed by system level design engineers. Besides, architecture design, task partitioning and scheduling, co-compiler optimization are executed by different design engineers in the SoC platform.

B. Application Profiling

There are two categories of instruction profiling tools: simulation based instruction profiler and compilation profiler. Simulation based instruction profiler uses an instruction set simulators, which can be further classified into static profiler or dynamic profiler. While compilation based profiler instruments the program by adding some counters to the interesting parts of program. During execution the statistic counter values are gathered. In this paper, we use the latter method based on SUIF toolkit.

Application profiler can collects some information for task partition, architecture design and co-compiler optimization of media SoC design, as shown in Figure 3. It shows the framework of SUIF based application profiler, which concludes mainly two parts: character identification and character merging. The framework provides the basis for specific needs of application analysis.

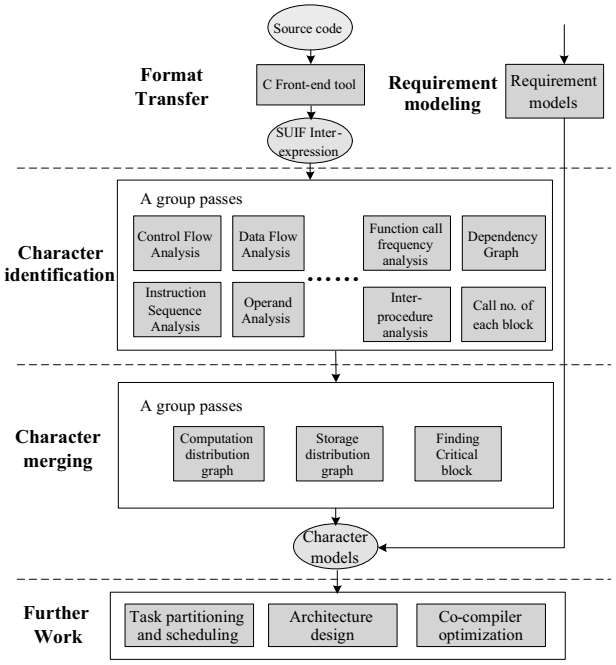

Figure 3. The overall structure of Media SoC hierarchical platform

In some typical applications of embedded media SoC system, the critical code blocks consume many system resources. So the foremost work of application profiler is to find these critical code blocks.

First, we partition the source code of application program into code block and build TFG (task flow graph) by taking code block as the node of TFG. Generally, there are three grains of TFG: instruction level, block level, and procedure level. We use block level in this paper, which further includes blocks, loops and functions, as shown in Figure 4. The loop or function can be compound, which means there may be some child blocks in them. The whole program is separated by loops or functions, and is built as TFG.

© 2009 ACADEMY PUBLISHER

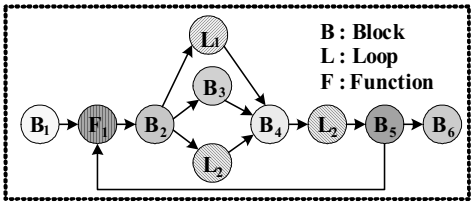

Figure 4. Three type of code block in TFG

Then, we analysis and statistic the character of each block node in TFG, such as:

1). the call numbers of each block node, and the iteration counts of each loop.

2). the numbers of instructions, execution time, and execution time percentage for each block node.

3). static code size.

4). storage capacity of each block node, including that of SUIF objective code and static storage area.

Finally, according to the statistics results, we recognize some blocks as critical blocks if their execution time percentage and storage capacity is higher than a thresh-old value. Task partition uses these critical blocks to decide which critical blocks should be speed up. These critical blocks are also useful for compiler-optimization. By code movement and code redistribution we can adjust their dynamic instruction numbers, which will have a strong impact not only on the size of critical block but also on its composition and the distribution of the frequently executed blocks.

C. Task Partitioning

When transferring source code of application program into TFG, we partition the critical blocks of TFG onto microprocessor or speed up components. The course is called task partition. Based on task partition, we analysis scheduling feasibility for the task system of TFG.

We must know about some attributes of the critical blocks in TFG for task partition. We can get the information by application profiling, such as computation information and storage information, as shown in Table I.

TABLE I. PROFILING REQUIREMENTS FOR TASK PARTITION

Critical block	Requirement description
Execution time	Percentage of execution time in program
Storage capacity	The storage capacity for SUIF objective code and static storage area

From the information in Table I, we use eACOGA (evolutionary ant colony optimization with genetic algorithm) algorithm for task partition [4]. These two parameters from Table I can provide the necessary data for the expected heuristic function of eACOGA:

$$\eta(k) = 1/((w_t * time(k)) + (w_m * \mathrm{mem}(k))) \quad (1)$$

In the above formulate, $\eta()$ is the local heuristic information, w_t and w_m represent the weight of computation and storage. The function *time()* and *mem()* represent the data of execution time and storage capacity from Table I. The variable k represents the kth ant for searching the optimal partitioning solutions.

After the task partition, the critical blocks, which are allocated onto speed-up components, will be noted by SUIF annotation. Then compiler can recognize this and generate corresponding codes.

ACO algorithm can find better solutions of partitioning more effectively [9]. But the strategies of random selection in constructing solutions lead to slow convergence speed [11]. Furthermore, the principle of positive feedback can not only strengthen the solutions with better performance, but also bring on stagnancy of search. The causation is that the main configurable parameters of the algorithm, such as α, β, ρ, Q, are set to fixed value when initializing, and it has no adaptability to various applications.

We present an eACOGA approach of task partition for media SoC. GA can evolve the configuration parameters of ACO algorithm by cross operation and variation operation. So eACOGA can evolve and optimize itself to search the optimal solutions.

We define the rules of eACOGA as follows:

Objective Function and Fitness Function: We define objective function as S_{best} = arg min C_p, fitness function as Fitness (p) = $1/C_p$. Where, C_p figures the cost of partition p.

Configure DAG: According to the need of specific applications, design experts configure the DAGs (Directed Acyclic Graph) by the application profiling to extract basic blocks as the nodes of DAG. The basic blocks perform at coarse-grained, including process, subprogram, loop and block. In this step, the parameters of the tasks should be decided.

Strategy of Render to DAG: For any nodes except t_n, ants try to render the color of t_j, the subsequence of t_i. Ants achieve the work according to the global heuristic information ($\tau_{ij}(k)$) of edge e_{ij} and the local heuristic information ($\eta_j(k)$) of node t_j. The ants on node t_i will render the color of node t_j as c_k at the probability of:

$$p_{ij}(k) = \frac{\tau_{ij}(k)^{\alpha}\eta_j(k)^{\beta}}{\sum_{l=1,2}\tau_{ij}(l)^{\alpha}\eta_j(l)^{\beta}} \quad (2)$$

Where, $\tau_{ij}(k)$ is the pheromone on edge e_{ij}, α and β is the factor of them and $\eta_j(k)$ is defined as above formula (1).

Use Genetic Algorithm to Evolve Parameters: We use genetic algorithm to evolve the parameters of ant colony optimization, such as α, β, ρ, Q. First, Configure the probability factor of cross and variation operation according to the size of population and the generation of evolution. The evolution rules of GA use proportional selection, single cross and even variation. Then by taking α, β, ρ, Q as the variable of fitness function, the optimal partition cost as fitness function and the course of ACO as the individual, we optimize α, β, ρ, Q repeatedly until finding the optimal solutions.

Pheromone Setting and Refreshing: We adopt MMAS (Max-Min Ant System) introduced by Thomas Stuzle ([10]), the refreshing equation of pheromone is:

$$\tau_{ij}(k) = \begin{cases} (1-\rho)*\tau_{ij}(k) + \Delta\tau_{ij}(k)_{best} \\ \tau_{ij}(k)_{\max}, \text{if } \tau_{ij}(k) > \tau_{ij}(k)_{\max} \\ \tau_{ij}(k)_{\min}, \text{if } \tau_{ij}(k) < \tau_{ij}(k)_{\min} \end{cases} \quad (3)$$

© 2009 ACADEMY PUBLISHER

Where, ρ is the evaporation ratio of pheromone, $\tau_{ij}(k)_{max}$ ($\tau_{ij}(k)_{min}$) is maximum (minimum) strength of c_k pheromone on edge e_{ij}, and $\Delta\tau_{ij}(k)_{best}$ is increment of c_k pheromone on edge e_{ij} done by the "best ant" in current ant system algorithm iteration. According to Ant-Cycle Model, $\Delta\tau_{ij}(k)_{best}$ is defined as:

$$\Delta\tau_{ij}(k)_{best} = \begin{cases} Q/C_{p_{best}}, & e_{ij} \text{ is rendered by } c_k \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

The pseudo-code for eACOGA algorithm is shown as follows. First, the configurations of DAGs are input, and after the execution of eACOGA algorithm the optimal solutions for partition are output. In eACOGA, ACO is built as a class, which has two main functions: GetAnt() and StartSearch(), as shown in Line 23-25. GA randomly encodes the variables of α, β, ρ, Q, as shown in Line 18-22. GA evolves the variables continuously by the operation of select, crossover and mutate, as shown in Line 6-14. In ACO, we put the ants randomly into DAGs and begin the search for the optimal solutions, as shown in the function on Line 28-36 and Line 37-47. After evaluating the fitness function values we output the optimal solutions, as shown on Line 12 and Line 15.

Pseudo-code for eACOGA algorithm

```
//Input: the configuration of DAGs
//Outputs: the optimal solutions for partition
//Q, α, β, ρ is the main parameters of ACO
1  Main(){
2     Generation = 0;
3     Initialize();
4     Evaluate();
5     Keep_the_Best();
6     foreach generation
7     {
8        select();
9        crossover();
10       mutate();
11       report();
12       Evaluate();
13       elitist();
14    }
15    Output the best fitness values;
16 }

17 Evaluate(){
18 For (mem = 0; mem < POPSIZE; mem++)
19 {
20    For (i = 0; i < NVARS; i++)
21       x[i+1] = population[mem].gene[i];
22    Q = x[1], α = x[2], β = x[4], ρ = x[4];
23    ACO partition = new ACO;
24    partition.GetAnt();
25    partition.StartSearch();
26    population[mem].fitness = CostFunctiontoFitness;
27 }

28 ACO::GetAnt(){
30    Randomly put ant into DAGs;
31    for (i = 0; i < nAntCount; i++)
32    {
33       task = rnd( nTaskCount);
34       ants[i].AddTaskIntoTabu(task);
35    }
36 }

37 ACO::StartSearch(){
38    Foreach ant
39    {
40       Select_NextNode_Accordingto_heuristic();
41       Moveto_NextNode();
42       Update_Tabu_Table();
43       find out the best solution of the step;
44    }
45    Update_Trail();
46    Find_theBest_Solutions_Of_Partition();
47 }
```

D. Automatic Cooperative Partitioning Flow

We have designed an automatic partitioning flow for mapping applications on media SoC, as shown in Figure 5.

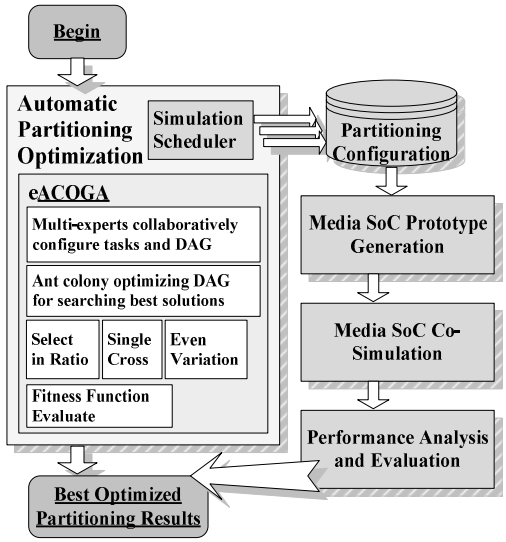

Figure 5. Automatic partitioning flow of media SoC

First, design experts configure the parameters of DAGs using application profiling. Then, application specific reconfigurable SoC prototype is generated according to existing media architecture templates [12]. Finally, we run media SoC transaction level co-simulation and output the best optimal partitioning solutions. Generally, we can get multiple optimal solutions after partitioning configuration using eACOGA. Then we use co-simulation to select the best optimal one from the multiple solutions.

The automatic partitioning flow has two main advantages:

(1).For each individual of genetic population in eACOGA, the flow of partition and reconfigurable SoC co-simulation can

© 2009 ACADEMY PUBLISHER

run automatically. When some constraints cannot be met, experts can request to stop the simulation.

(2).Transaction level simulation in SystemC can describe various behaviors of media SoC with faster speed and nicer accuracy. Architecture template enhances reuse of existing SoC design and achieves exploration speedup well.

V. Experimental Results

A. Target Architecture and Benchmarks

We use eACOGA algorithm for the task partition of a heterogeneous reconfigurable SoC system, which consists mainly of 32-bit RISC microprocessor called Estar and reconfigurable arrays called LEAP [13]. Both Estar and LEAP are developed by our research group. We use Estar for common computing. It has 8KB instruction cache and 8KB data cache, 266M Hz CPU core. Besides, we use LEAP for reconfigurable computing. It can accelerate applications through loop self-pipelining technique. LEAP steps the loop iteration automatically and has the ability to exploit parallelism at loop-level, instruction level, and task-level.

The SoC system integrates some typical algorithms in the field of SDR (Software Defined Radio), SAR (Synthetic Aperture Radar) image and high-precision digital image encode/decode. We have designed some typical algorithm blocks running on reconfigurable systems, such as FFT (Fast Fourier Transformation), Sobel Edge Detection, Median Filter, Matrix Multiply, FDCT (Forward Discrete Cosine Transform), IDCT (Inverse Discrete Cosine Transform), etc. We have tested the performance of these typical algorithms on Estar and LEAP and translate execution time into time cost and resources used into area cost, as shown in Table II.

TABLE II. The performance of typical algorithms. Execution time of them on Estar and LEAP are shown. cPE and mPE discribes the PE for computation and the PE for memory.

Typical Algorithms	Execution Time		Resources (cPE, mPE)
	Estar (Mcycle)	*LEAP (Kcycle)*	
512 point FFT	32.320	6.721	10c4m
1024 point FFT	72.353	12.802	10c4m
Edge Detection (320x240)	39.720	216.958	16c7m
Edge Detection (480x360)	87.898	474.205	16c7m
Median Filter (320x240)	1580.368	220.010	30c7m
Median Filter (480x360)	3590.500	478.792	30c7m
Matrix Multiply (64x64)	54.315	79.141	30c10m
Matrix Multiply (128x128)	2522.258	318.901	30c10m
FDCT	2433.389	2838.905	30c10m
IDCT	2437.417	2839.044	30c10m

We also tested some other basic blocks, such as process, subprogram, loop and block. We developed a tool for application program analysis and profiling, namely CBPT (Critical Block Profiling Toolset), which can recognize these algorithms and generate them as the nodes of DAG. We also generate the attributes of these nodes, such as configuration time, computing time, memory accessed, number of PEs used, and execution time on embedded microprocessor.

CBPT is developed based on SUIF, an infrastructure for compile analysis. The tool picks up some basic block at the granularity of procedure, control flow block, and instruction defined in SUIF. The procedure granularity represents a function. The control flow block granularity comprises if, for, loop and tree block. The instruction granularity provides the great details of instruction information. These basic blocks are imported as the task nodes of DAG for the later partition. We use SUIF to build many basic blocks in the application field of image process. All the basic blocks are imported as the task nodes randomly to build different scale DAGs for testing the proposed algorithm. Table III shows the statistics results of blocks in MediaBench. We define the threshold value of execution time (0.25%). We also make a statistics of some top critical blocks with execution percentage and storage percentage. Table IV shows top 5 execution percentage of critical blocks in MediaBench. We find that top 5 critical blocks contribute average 20.9% execution time of program. Besides, Table VI shows top 10 code block of computation code and storage code.

TABLE III. Statistic results for blocks in MediaBench

Image process application		Sum	Critical blocks	Critical blocks for speed-up
JPEGDec	Basic block	3925	---	---
	Loop	3567	282	9
	function	218	22	12
JPEGEnc	Basic block	3476	---	---
	Loop	2984	253	7
	Function	195	17	8
MPEGDec	Basic block	7139	---	---
	Loop	6527	479	11
	Function	371	39	14

TABLE IV. The cumulative execution percentage of top 5 critical blocks in MediaBench

Bench-Marks	Critical Block					Sum Top 5
	1	2	3	4	5	
JPEG Decode	19	33	37	42	46	20.4%
JPEG Encode	12	21	28	34	38	16.8%
MPEG Decode	34	37	43	46	52	25.6%

B. Result Analysis

In eACOGA algorithm, we set w_t = 1, w_a = 10, ACO iteration counts = 100, GA population size = 5, GA max

© 2009 ACADEMY PUBLISHER

generation = 50, GA cross probability factor = 0.8, and GA variation probability factor = 0.15, the parameters region of α, β, ρ, Q respectively as [5, 1], [5, 1], [0.8, 0.2], [100, 40].

To compare the quality of eACOGA with that of other researches, we select ACO algorithm in literature [6]. We set ACO parameters as: Q = 1000, α = 1, β = 1 and ρ = 0.8. Figure 6 show that eACOGA has better ability than ACO in searching for the global optimal solutions. The algorithm of ACO gets into local optimal solutions (1196). However, eACOGA can find the global optimal solutions (1012) effectively for the advantage of self-adaptive optimization. Besides, another algorithm we have researched, named initACO (ACO with init pheromone), has the performance between them (1068) [14].

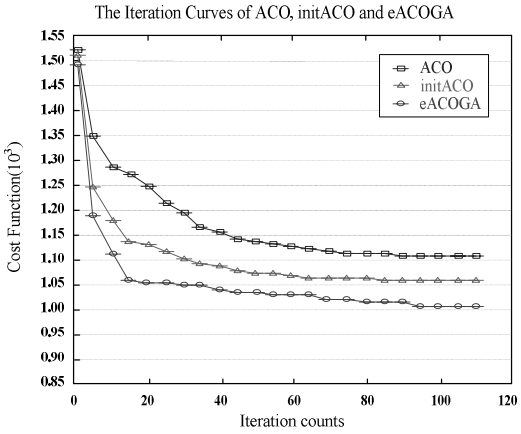

Figure 6. Compare the iteration curves of eACOGA with initACO and ACO

The possible explanation for the proposed eACOGA approach to outperform the basic ACO method with better optimal solutions is that the main control parameters of ACO affect its performance greatly. The pheromone ($\tau_{ij}(k)$) is the carriers of the past information, while the heuristic function ($\eta_j(k)$) is the carriers of the future information. Many experiments on basic ACO shows that the factor α which controls $\tau_{ij}(k)$ has important impact on ACO performance and the factor β which controls $\eta_j(k)$ has a substantial effect on global convergence. The factor ρ which reflects the change of pheromone affects the ability of global search and the speed of convergence. The factor Q which reflects the amount that the searching ants release, concerns the positive feedback ability of the searching ants and the rapid convergence of ACO.

The basic ACO algorithm initializes these factor parameters with fixed values, as shown in literature [11], which set the factor parameters in all the experiments with the same values: Q=1.000, α = β = 1, ρ = 0.8. While different DAGs needs a different combination of the factor parameters. So the information motivates us to a hybrid approach of ACO and GA together. That is we use GA evolve and select the suitable factor parameters of ACO for different DAGs. Our experiments show that eACOGA has statistically robust in finding close to optimal solutions.

Figure 7 shows the comparison of eACOGA with initACO and ACO by solution quality of partitions. The x-axis gives the solution quality compared to the overall solutions and the y-axis gives the total solutions worse than the solution quality in percentage. We find that 2% solutions found by ACOGA are within top 92.3% range. Besides, 2% solutions found by initACO and ACO are with top 86.4％ and 74.6% range. That is, most partition results by ACOGA are closer to the optimal solutions than the other two algorithms.

The algorithm of eACOGA has some configurable parameters. We can optimize the performance of eACOGA by configuring the parameters and running simulation. Table V shows that when cross factor = 0.8, eACOGA has better ability for random search and can reduce iteration counts. In the same way we make experiments on three group probability factor of variation (0.15, 0.20 and 0.25). The results show that 0.15 is the best value for our problem.

TABLE V. AVERAGE ITERATION COUNTS WITH DIFFERENT CROSS FACTOR.

Scale of Tasks	Single Cross Factor		
	0.6	0.7	0.8
8	9	11	8
16	15	11	13
32	28	26	25
64	53	51	46
128	89	83	85

TABLE VI. TOP 10 CODE BLOCK OF COMPUTATION CODE AND STORAGE CODE

Top 10 computation code block				Top 10 storage code block			
ID	Node Name	Computation Amount (cycle)	Computation Proportion (%)	ID	Node Name	Storage Amount (bit)	Storage Proportion (%)
1	j_fwd_dct_3	645	1.05%	1	j_fwd_dct_6	10208	0.81%
2	j_fwd_dct_6	641	1.04%	2	j_fwd_dct_3	10144	0.80%
3	j_rev_dct_5	551	0.90%	3	j_rev_dct_5	8800	0.70%
4	j_rev_dct_11	541	0.88%	4	j_rev_dct_11	8736	0.69%
5	pass2_dither_10	382	0.62%	5	pass2_dither_10	7360	0.58%
6	smooth_coefficients_7	279	0.45%	6	j_c_defaults_7	5312	0.42%
7	j_c_defaults_7	260	0.42%	7	h2v2_smooth_downsample_11	4960	0.39%
8	alloc_sampling_buffer_8	232	0.38%	8	reverse_DCT_12	4576	0.36%
9	get_rgb_ycc_rows_5	213	0.35%	9	h2v2_smooth_downsample_12	4480	0.35%
10	rgb_ycc_init_3	212	0.35%	10	color_quantize_dither_10	4480	0.35%

© 2009 ACADEMY PUBLISHER

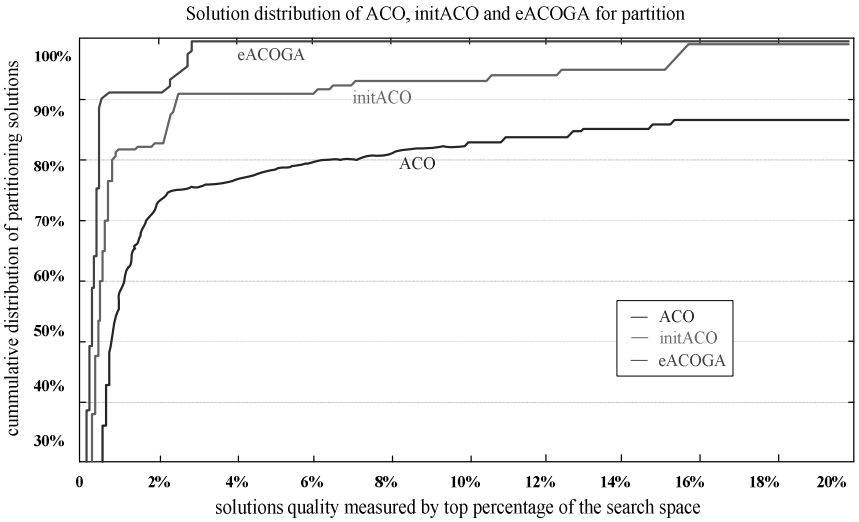

Figure 7. quality of partition solutions of eACOGA comparing with initACO and ACO

VI. CONCLUSIONS

Existing EDA systems support human-computer interaction only, not touch on human-human interaction. This paper proposes a collaborative system level design approach of media SoC design, which supports both human-computer and human-human interaction well. We present an application-concerned system level design and integrate it into Media SoC hierarchical platform, which provides convenient collaborative environment for multi-field experts to work.

The novel application-concerned system level design method, together with hierarchical platform for EsmpSoC, is a kind of advanced technology. It supports design reuse, concern-separation, and application-aware. So they can improve design quality and reduce design complexity.

Application profiling can find critical blocks in program automatically. The execution information gathered by application profiling can direct for task partition. From the experiments we see that critical blocks consume much system performance, and we gain system speed-up by finding them and partitioning them onto speedup components with heuristic evolutionary algorithm.

ACKNOWLEDGMENT

This work is sponsored by the National Science Foundation of China under the grant NO.90207019 and NO.90707003.

REFERENCES

[1] Reinhard W., Schweitzer and Volksen G, "CSCW tools: concepts and architectures," *IEEE Computer*, 27(5), 1994, pp. 28–36.

[2] Regli, W., "Internet-enabled Computer-Aided Design," *IEEE Internet Computing*, 1997, pp. 39–51.

[3] Pahng, F., Sein, N., Wallace, D.R., "Distributed Modeling and Evaluation of Product Design Problems," *Computer-Aided Design*, 1998, pp. 411–423.

[4] Sikun Li, Dawei Wang, etc, "Distributed Collaborative Partition Method of Reconfigurable SoC Using Ant Colony Optimization," *The 11th International Conference on CSCW in Design*, Melbourne, 2007, pp. 133–138.

[5] C.Lee, M.Potkonjak, W.H.Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," *International Snmposium on Microarchitecture*, *MICRO30*, 1997, pp. 292–303.

[6] Dinesh C. Suresh, Walid A. Najjar, Frank Vahid, Jason R. Villarreal and Greg Stitt, "Profiling tools for hardware/software partitioning of embedded applications," *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems,* San Diego, California, USA, 2003.

[7] A.Srivastava and A.Eustace, "ATOM: A system for building customized program analysis tools," *In ACM conference on Programming Language Design and Implementation*, 1994, pp. 196–205.

[8] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion and M. S. Lam, "Maximizing Multiprocessor Performance with the SUIF Compiler," *IEEE Computer*, December 1996.

[9] M Dorigo, V Maniezzo, and A Colorni, "Ant System: Optimization by a Colony of Cooperating Agents. IEEE Trans on Systems," *Man and Cybernetics*, Part-B. 1996, 26(1), pp. 29–41.

[10] T Stutzle, H H Hoos, et al, "MAX-MIN Ant System," *Future Generation Computer System*, 2000, 16(8), pp. 889–91.

[11] Gang Wang, Wenrui Gong and Ryan Kastner, "System Level Partitioning for Programmable Platforms Using the Ant Colony Optimization," *In 13th International Workshop on Logic and Synthesis (IWLS'04)*, 2004.

[12] Sikun Li, Dawei Wang and Peng Zhao, "An Architecture Template based SoC Transaction Level Modeling and Simulation Method," *The 7th International Conference on Computer-aided Industrial Design & Conceptual Design*, HangZhou, 2006, pp. 362–367.

[13] Yong Dou, Xicheng Lu, "LEAP: A Data Driven Loop Engine on Array Processor," *The 4th Int'l Conf on Parallel*

© 2009 ACADEMY PUBLISHER

and Distributed Computing, Applications and Technologies (PDCAT'03), 2003.

[14] Xiong Zhihui, Li Sikun, Chen Jihua, "Hardware/Software Partitioning Based on Ant Optimization with Initial Pheromone," *Journal of Computer Research and Development*, 2005, 42(12), pp. 2176–2183.

Dawei Wang was born in Donggang of P.R.China in 1980. He is now Ph.D.candidate in College of Computer Science from National University of Defense Technology at Changsha. The major field of his interest has been SoC system design method, and electronic design automation, etc.

Since 1998 he was 10 years with the College of computer Science from National University of Defense Technology. As a Ph.D.candidate, he worked at the university over 6 years in all. His research is recently oriented into application specific and coarse-grained reconfigurable architectures.

Peng Zhao was born in Hebi of P.R.China in 1979. He is now Ph.D.candidate in College of Computer Science from National University of Defense Technology at Changsha. The major field of his interest has been SoC system design method, and electronic design automation, etc.

Since 1998 he was 4 years with the Artillery College from National University of Defense Technology. Then he obtained Master's Degree in College of Computer Science from National University of Defense Technology. As a Ph.D.candidate, he worked at the university in the field of SoC high level design. His research is recently oriented into application profiling and memory analysis.

Sikun Li was born in Qingdao of P.R.China in 1941. He obtained Bachelor's degree in Military Engineering Institute of the PLA, Haerbin, in 1965. The major field of his interest has been SoC design methodology and advanced VLSI design method, and electronic design automation, etc.

Since 1984 he was 24 years with the the College of computer Science from National University of Defence Technology at Changsha. As a Full Professor he taught and made researches in the field of CAD, VLSI and SoC. His research is recently oriented into SoC system level design and embedded media design.

Prof. Sikun Li is the manager of CAD/CG committee of China Computer Federation, committees for international conference CAD/CG, CSCWD.

© 2009 ACADEMY PUBLISHER