A Technique for QoS-based System Partitioning

Johnson Kin†, Chunho Lee‡, William H. Mangione-Smith† and Miodrag Potkonjak‡

†Electrical Engineering Department
University of California at Los Angeles
Los Angeles, CA 90095-1596, USA
email:{johnsonk, billms}@icsl.ucla.edu

‡Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095-1596, USA
email:{leec, miodrag}@cs.ucla.edu

Abstract— Quality of service (QoS) has been an important topic of many research communities. Combined with an advanced and retargetable compiler, variability of applications-specific very large instruction word (VLIW) processors has been shown to provide excellent platform for optimizing performance under area constraints. Although media servers and others that are natural candidates for QoS management are embedded systems and implementing QoS management using the applications-specific VLIW platform can provide a new horizon for efficient and effective system design, until now no effort has been reported. In this paper, we introduce a QoS-based system partitioning scheme that addresses the need for maximizing net benefit of a system under resource constraints by exploiting applications-specific VLIW processors. The approach utilizes the modern advances in compiler technology and architectural enhancements that are well matched to the compiler technology.

Experimental results indicate that the approach presented in this paper is very effective in system partitioning for QoS given a set of heterogeneous processors.

I. Introduction

Quality of service (QoS) has been an important topic of many research communities. Service schemes that can offer varying degrees of service quality depending on available resources are well known in networking, multimedia systems, real-time systems and distributed systems. The idea is that by devoting more resources for an application, a user can experience increased benefit from the application due to its increased performance. For example, multimedia systems involving audio and video streams can provide higher resolution video or higher quality audio if more system resource is used. Lower frame rate or lower resolution video is acceptable, albeit not desirable, when system resource is tight.

In the last decade multimedia services gained popularity in many application domains such as education and training systems, office and business systems, information and point of sales systems [8]. In the recent years, the wide spread use of the World Wide Web produced a fertile ground for multimedia services [1]. The deployment of consumer broadband services (e.g. cable modem and xDSL services) seem to be accelerating the trend.

Consequently, system design approach with efficient and effective resource assignment for management of QoS on the part of service providers is desired. Systems that service providers use must be designed with QoS in mind in order to exploit the service characteristics and maximize the utility of systems. Such systems will typically run multiple applications for multiple users. Each application provides different level of utility and each user perceives raw utility generated by the system differently from each other based on his/her needs.

In addition, advances in compiler technology for instruction-level parallelism (ILP) have significantly increased the ability of a microprocessor to exploit the opportunities for parallel execution that exist in various programs written in high-level languages. State-of-the-art ILP compiler technologies are in the process of migrating from research labs to product groups [3, 10]. At the same time, a number of new microprocessor architectures having hardware structures that are well matched to most ILP compilers have been introduced. Architectural enhancements found in commercial products include predicated instruction execution, VLIW execution, multi-gauge arithmetic (or variable-width SIMD) and split register files.

It has been shown that the variability of applications-specific VLIW processors using an advanced and retargetable compiler is highly beneficial in terms of optimizing performance under area constraints [4]. Although media servers and others that are natural candidates for QoS management are embedded systems and implementing QoS management using the applications-specific VLIW platform can provide a new horizon for efficient and effective system design, until now no effort has been reported.

In this paper, we introduce a QoS-based system partitioning scheme that addresses the need for maximizing net benefit of a system under resource constraints by exploiting applications-specific VLIW processors. The approach utilizes the modern advances in compiler technology and architectural enhancements that are well matched to the compiler technology.

II. Related Work and Our Contributions

Extensive work on resource allocation and assignment such as work on resource assignment for media server domain [11, 15, 18] and ATM switch scheduling [16] has been reported. Rajkumar and others published an analytical model for QoS management [13, 14]. Resource allocation for QoS has been well understood topic in many research areas [5, 17].

We identify the need for an efficient and effective approach to QoS-based system partitioning. We develop tools to address the need. Our tools combine a retargetable ILP compiler, instruction level simulators, a set of complete media applications written in a high level language and a QoS-based system partitioning algorithm.

We prove that the task partitioning problem presented in this paper is NP-hard. Comparisons between solutions obtained by the developed heuristics and upper bound values indicate that they are very effective. We adapted a force-directed approach for data-flow graph scheduling developed by Paulin and Knight [12]. It has been adopted by many authors due to its clear intuitive foundations and strong performances. The overall procedure for selecting and optimizing benefit values is done by adapting a general combinatorial optimization technique known as simulated annealing [7].

We adopt a target architecture that includes heterogeneous processors that are general purpose multiple-issue machines. We use the notion of application-specific VLIW processor, a general purpose processor that is more suitable to run specific applications than others thanks to its architectural features. e.g. more issue units for typical media applications.

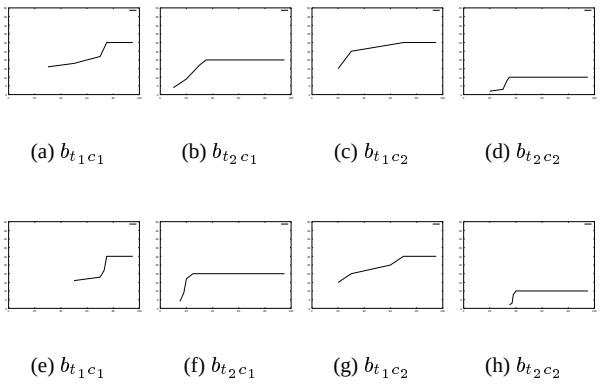
Experimental results indicate that the approach presented in this paper is very effective in system partitioning for QoS given a set of heterogeneous processors.

III. A Motivational Example

We illustrate the key ideas of our approach using a simple example. Consider that we have two processors (p_1 and p_2) for two applications[1] (t_1 and t_2) and two users (c_1 and c_2). Then there are eight benefit curves showing benefit values of applications to users corresponding processor utilization factors. We assume that the only requirement to have a feasible partition is to make sure that the total utilization is less than 100%. Figures 1(a) through 1(d) show benefit curves for processor p_1 whereas Figures 1(e) through 1(h) for processor p_2. The diagrams represent benefit values (vertical axis) corresponding to processor utilization (horizontal axis).

No processor given is able to run more than two applications and achieve maximum benefit. For example, if we assign applications t_1 and t_2 for user c_1 and t_2 for c_2 to processor p_1 (represented by Figures 1(a), 1(b), and 1(d)) we have to reduce at least one benefit value of the applications since the total utilization will exceed 100% when all application run to provide highest possible benefit values. When we assign two applications to a processor, say t_1 and t_2 for user c_1 to processor p_2 (represented by Figures 1(e) and 1(f)), there is no need to reduce the benefit values to fit the applications to the processor. On the other hand, the remaining two applications cannot fit into processor c_1 without reducing a benefit value or both since the total utilization exceeds 100%. However, as shown in Figure 1(c), the benefit value of t_1 on p_1 to c_1 is most insensitive to the decrease of utilization interval 30-70% among all benefit curves. It turns out that the choice of the application to reduce benefit value in order to obtain a feasible solution is optimal in terms of maximizing total benefit.

[1]We use the terms task, individual application and media application interchangeably.



(a) $b_{t_1c_1}$ (b) $b_{t_2c_1}$ (c) $b_{t_1c_2}$ (d) $b_{t_2c_2}$

(e) $b_{t_1c_1}$ (f) $b_{t_2c_1}$ (g) $b_{t_1c_2}$ (h) $b_{t_2c_2}$

Fig. 1. A set of benefit curves for two users (customers) and two task on two processors (Figures (a) through (d) show benefit curves for processor p_1 whereas Figures (e) through (h) for processor p_2)

IV. Target Architecture

The target architecture we use resembles a multiprocessor system with shared memory except that all the processors are assumed to be laid out on a single die. A media task is assigned to its dedicated processor. i.e. a task cannot be distributed to more than one processor. More than one application can be assigned to a processor if the given performance constraints are guaranteed to be met, i.e. total utilization by all tasks on the processor must not exceed 100%. Shared memory is used for data communication between tasks. Each processor maintains its own cache. When more than one application are assigned to a single processor, flushing and refilling of the cache is needed and the run time measurement platform takes it into account by flushing cache.

For synchronization and scheduling, we use the notion of *quanta*. One of the benefits of using the notion of quantum is that it simplifies synchronization of several applications running on multiple processors. As explained in [13], we assume the following:

Utilization: The resource allocation to an application will be in terms of the utilization of a resource. Once a certain utilization has been allocated, an application may either choose its own execution time and period to achieve that utilization or use an appropriate processor-sharing scheme such as weighted fair-sharing.

Schedulability: The constraint $\sum_{i=1}^{n} R_i \leq R_j$ implies that a resource can be "fully" consumed. The term R_i refers to a portion of resource type R_j. As is well known, this is not always true for fixed-priority scheduling algorithms but is true for the earliest deadline scheduling algorithm under

ideal conditions. A different maximal resource constraint beyond the scope of this paper must be used to support fixed-priority schemes.

V. Problem Formulation

We first define terms that will be used to formally define the task partitioning problem.

Service level: Let $s_{t,p}$ be service level of an application t on a processor p. It is a function of resources devoted to the application.

An application can be executed on any processor as long as its resource utilization requirement under selected benefit level can be satisfied. The utilization factor of an application on a processor may or may not be the same as the one on other processors for a given benefit level (i.e. in general, $u_{p_i} \neq u_{p_j}$, $where\ i \neq j$). The service level of an application is given by

$$s_{t,p} = f_{t,p}(u_p).$$

The term u_p is utilization factor of processor p.

Benefit: The benefit of an applications to a user c is perceived utility materialized with the expense of processor utilization u_p and given by

$$b_{t,c,p} = g_{t,c}(s_{t,p}).$$

Net Benefit: The net benefit generated by a system is the sum of all benefits perceived by all user of the system and given by

$$B = \sum_{\forall c} \sum_{\forall t} b_{t,c,p'}.$$

The term $b_{t,c,p'}$ is the benefit materialized on the processor p the application t is assigned to.

QoS Partitioning Problem (QPP)

Instance: Given a set of applications $A = \{a_i | i = 1, 2, ..., l\}$, processors $P = \{p_j | j = 1, 2, ..., m\}$, users $C = \{c_k | k = 1, 2, ..., n\}$, service functions $F = \{f_{t,p} | t = 1, 2, ..., l, p = 1, 2, ..., m\}$, benefit functions $G = \{g_{t,c} | t = 1, 2, ..., l, c = 1, 2, ..., n\}$ and a constant M,

Question: Is there a partition of the application set across the processor set such that the net benefit B is greater than or equal to M?

Theorem The QPP is NP-hard.

Proof We prove this by showing that the inexact 0-1 knapsack problem maps directly to a special case of the QPP. The inexact 0-1 knapsack problem is as follows:

Maximize: $\sum_{i=1}^{n} v_i$

```
generate initial solution
while( system not in frozen or equilibrium state )
  for a number of iteration
    generate a new solution with a new set of benefit values
    if( the new solution is better than old solution )
      accept the new solution
    else
      accept the new solution with a probability
  endfor
  reduce temperature
endwhile
```

Fig. 2. Top-level view of the algorithm

Subject to:

$$\sum_{i=1}^{n} R_i \leq R$$

where there are n objects each with size R_i and value v_i, and R is the size of the knapsack.

The above problem is identical to the following special case of the QPP. Suppose we have a single processor and each user has a benefit function in the following form:

$$g_{t,c}(s_{t,p}) = \begin{cases} 0\ if\ s_{t,p} < S_{t,p} \\ S_{t,p} if\ s_{t,p} \geq S_{t,p} \end{cases}$$

where $S_{t,p} = f_{t,p}(U_p)$.

Thus user c receives a benefit value of $S_{t,p}$ if and only if utilization U of processor p is given to task t for user c.

An identity transformation makes the inexact 0-1 knapsack problem a special case of the QPP.

VI. Approach and Algorithm

We measure utilization factors of applications on a variety of processors using IMPACT compiler suite [2]. The IMPACT C compiler is a retargetable compiler with code optimization components especially developed for multiple-instruction-issue processors. The target machine for the IMPACT C can be described using the high-level machine description language (HMDES) [6]. A high-level machine description supplied by a user is compiled by the IMPACT machine description language compiler.

We divide the problem into two sub-problems: selection of benefit values and partitioning of tasks having the selected benefit values. Since the QPP is NP-hard, we adapted the simulated annealing heuristic for optimizing net benefit. Coupled with a force-directed heuristic for partitioning, the simulated annealing based algorithm is used to select benefit levels for each application and user.

The initial state of the optimization system is defined by initial temperature and initial random solution. The system moves toward an equilibrium (or frozen) state based on the following functions: net benefit measurement function, neighbor solution

function, temperature reduction function, nondeterministic solution acceptance function, equilibrium criterion and frozen criterion. The power consumption measurement is made after the tasks are assigned to a processors. The tentative assignment is done by a force-directed heuristic and is explained in the later part of this section. Secondly, a neighbor solution is generated by increasing or decreasing one randomly chosen benefit value at a time. Thirdly, the temperature is updated by the standard geometric temperature reduction function. Fourthly, the equilibrium criterion is specified by the number of iterations of the inner loop. The number of iterations of the inner loop is set to 65.

Force-directed heuristic consists of four step computations: computation of distribution graphs, computation of forces, computation of self-forces and selection of one move (one assignment of an application to a processor) based on self-force values. Distribution graph is given by

$$DG_p = \sum_{\forall t} DG_{t,p},$$

where $DG_{t,p} = (\sum_{\forall p_i} (u_{t,p_i})^{-1})^{-1}$.
Force is given by

$$F_{t,p} = DG_p \times \delta_{t,p},$$

where $\delta_{t,p} = 1 - u_{t,p}$ if task p is assigned to processor p. Otherwise $\delta_{t,p} = -u_{t,p}$.
Self force is given by

$$SF_{t,p} = \sum_{\forall p_i} F_{t,p_i}.$$

VII. Experimental Results

We use eight applications from [9] to form our test cases. The benefit functions of each user on different applications are generated randomly. Each curve consists of four discrete benefit values based on the service level. All the service level of applications are normalized with respect to the lowest service level. For example, the lowest service level of the *mpeg* is equal to 15 frames per second on a 1.5Mbps bitstream. We assume that the benefit curve is non-decreasing. Our algorithm can easily accommodate more discrete levels in the benefit curves.

First, we use all eight applications as eight tasks with different benefit graphs and try to fit them in four different pools of processors. We use the notion of *processor pool* to mean a set of processors available for system design. All processor pools consist of heterogeneous processors with issue width ranging from 1 to 8, and one or two branch units. All the processors have 8KB of instruction cache and data cache. The first pool has 7 processors, the second pool 6, the third pool 5, and the last one 4 (see Table VII). Going from 7 to 4 processors, the worst performing processor (usually with less issue width) is taken out each time to form the next pool. Then, we use our simulated annealing based algorithm to optimize the net benefit and compare it with the upper bound, a random solution, and a greedy solution.

We then increase the problem size to stress our algorithm with a larger number of tasks, 15 and 20 tasks, with different benefit curves. Table VII shows an example of how the benefit levels look like and the actual solution of 15 tasks partitioned across 7 processors shown in Table VII.

In order to get an upper bound of net benefit, we use the notion of *superprocessor*. We compute amount of work an individual processor has to perform when an application is proportionally distributed across processors based on their performance. Since faster processor gets more amount of work than slower ones, the utilizations of prorated work are the same across the processors. The utilization of an application t at service level l on the superprocessor is given by

$$U_{t,l} = (\sum_{\forall p_i} (u_{t,l,p_i})^{-1})^{-1}$$

We refer this utilization as superprocessor's utilization (sU). We calculate sU for each task at each performance level. Note that, if the summation of sU at minimum performance level on all tasks is higher than 1, it simply means that there is no feasible solution across the entire solution space.

Once we get all the utilization factor $sU_{t,l}$ for task t at service level l, we calculate all $\Delta sU_{t,l-1} = sU_{t,l} - sU_{t,l-1}$ from one performance level to the immediate next level within a processor. We also calculate all $\Delta_{B_{t,l-1}}$ again from one performance level to the next level. Now we sort the $\Delta sU_{t,l}$'s in descending order while sorting $\Delta B_{t,l}$'s in ascending order. Then we start our solution using highest performance level on all tasks and their best benefit. Check the total sU to see if all is less than 1. If not, reduce the utilization from the sorted $\Delta sU_{t,l}$ list and the total benefit from sorted $\Delta B_{t,l}$ list one at a time until the total sU is under 1 (i.e. a feasible solution is obtained). The random solution is obtained by simply putting tasks in different performance level randomly until we get a feasible solution. The program terminates when it finds a feasible solution or if it cannot find a feasible solution in 1000 iterations.

In order to get a greedy solution, we start out with the minimum solution, i.e., all benefit levels are at the minimum. Then using the sU, we find the best dB/dsU of a task to upgrade its performance level until it exceeds the utilization of any processor.

Figure 3 show some results. In Figure 3(a), we present a test case with 8 different media applications with different benefit curves and 4 pools of processors. We observe that our solutions are 10% range of the upper bound.

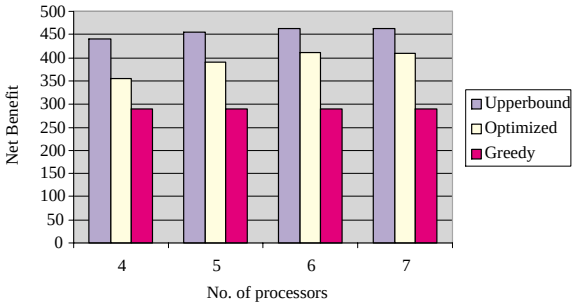
In Figure 3(a), we present a test case with 15 and 20 media applications and the same 4 pools. We observe a similar results to the previous ones. Our solutions are around 10% below the upper bound while it keeps its performance around 10% better than the greedy solution. The slightly less sharp results for more application cases seem to indicate that as we increase the number of applications, the solution space shrinks and the per-

TABLE I
PROCESSOR CONFIGURATIONS

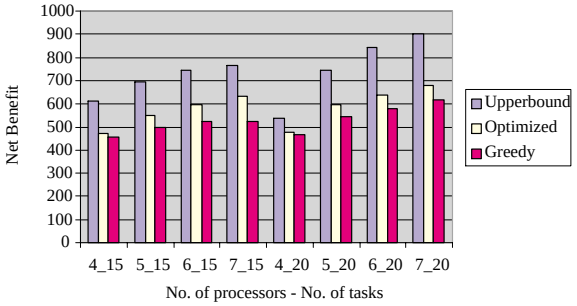
ID	I-cache	D-cache	issue width	branch units
1	8k	8k	1	1
2	8k	8k	2	1
3	8k	8k	4	1
4	8k	8k	8	1
5	8k	8k	4	2
6	8k	8k	8	2
7	8k	8k	8	4

TABLE II
BENEFIT GRAPH AND SOLUTION WITH 15 TASKS ON 7 PROCESSORS

App. Name	Level 1	Level 2	Level 3	Level 4	Selected Level	Proc. Assign.
Cjpeg	30	40	45	47	2	7
Cjpeg	5	10	15	50	4	7
Djpeg	2	5	7	30	4	4
Djpeg	30	45	67	70	4	3
Mpeg2dec	10	12	14	15	1	5
Mpeg2dec	40	43	56	60	1	6
Mpeg2enc	10	20	30	30	1	5
Pegwitdec	20	40	55	80	2	4
Pegwitdec	20	31	40	55	2	2
Pegwitenc	20	30	55	67	1	2
Pegwitenc	5	30	45	70	2	3
Rawcaudio	70	72	73	75	4	5
Rawcaudio	32	57	70	80	4	1
Rawdaudio	13	30	45	50	3	6
Rawdaudio	24	45	50	60	4	6



(a) 8 applications



(b) 15 and 20 applications

Fig. 3. System partitioning results

formance difference between our algorithm and greedy heuristic gets smaller.

The longest running time (20 tasks on 4 processors) is approximately 20 seconds on a Pentium II 450Mhz running Windows NT.

VIII. CONCLUSION

The experimental results indicate that there is good potential to design a system that is optimized for QoS management by utilizing advanced compiler and architecture. Algorithms developed for QoS-based system partitioning shows good performances in terms of speed and quality. The use of complete applications that are written in a high-level language demonstrates the practical nature of the approach.

REFERENCES

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world wide web. *Communications of ACM*, 37(8):76–82, 1994.

[2] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. m. W. Hwu. IMPACT: An architectural framework for multiple-instruction-issue processors. In *International Symposium on Computer Architecture*, 1991.

[3] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computing*, C-30:478–490, 1981.

[4] J. A. Fisher, P. Faraboschi, and G. Desoli. Custom-fit processors: Letting applications define architectures. In *International Symposium on Microarchitectures*, Paris, France, 1996.

[5] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its applications to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas in Communications*, September 1991.

[6] John C. Gyllenhaal, Wen mei W. Hwu, and B. Ramakrishna Rau. Optimization of machine descriptions for efficient use. In *Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

[7] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[8] F. Kretz and F. Colaitis. Standardizing hypermedia information objects. *IEEE Communications Magazine*, pages 60–70, May 1992.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitectures*, 1997.

[10] W. m. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery. The superblock: An effective technique for VLIW and superscalar compilation. *Journal of Supercomputing*, 1993.

[11] R. Nagarajan and C. Vogt. Guaranteed performance of multimedia traffic over the token ring. Technical Report 439210, IBM-ENC, Heidelberg, Germany, 1992.

[12] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of ASICS. *IEEE Transactions on CAD*, 8(6):661–679, June 1989.

[13] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1997.

[14] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. Practical solutions for qos-based resource allocation problems. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1998.

[15] K. Ramamritham and J.A. Stankovic. Scheduling algorithms and operating system support for real-time systems. *Proc. of the IEEE*, 82(1):55–67, January 1994.

[16] D. B. Schwartz. ATM scheduling with queuing delay predictions. *Computer Communication Review*, 23(4):205–211, October 1993.

[17] K. G. Shin, D. D. Kandlur, and D. Ferrari. Real-time communication in multi-hop networks. *IEEE Transactions on Parallel*, 1994.

[18] R. Steinmetz. Analyzing the multimedia operating systems. *IEEE Multimedia*, 2(1):68–84, 1995.