HYPERMEDIA PROCESSORS: DESIGN SPACE EXPLORATION

Johnson Kin†, Chunho Lee‡, William H. Mangione-Smith†
and Miodrag Potkonjak‡
†Electrical Engineering Department, University of California, Los Angeles, CA
‡Computer Science Department, University of California, Los Angeles, CA

Abstract - We present a framework for area optimal system design space exploration for hypermedia applications. We focus on a category of processors that are programmable yet optimized to a hypermedia application. The key components of the framework presented in this paper are a retargetable instruction-level parallelism compiler, instruction level simulators, a set of complete media applications written in a high level language and a media processor synthesis algorithm. The framework addresses the need for area optimal system design by exploiting the instruction-level parallelism found in media applications by compilers that target multiple-instruction-issue processors.

Using the framework we conduct an extensive exploration of area optimal system design space for a hypermedia application. We found that there is enough ILP in the typical media and communication applications to achieve highly concurrent execution when throughput requirements are high. On the other hand, when throughput requirements are low, there is no need to use multiple-instruction-issue processors.

INTRODUCTION

Hypermedia represents a combination of hypertext and multimedia technologies [4]. The concept of hypertext was proposed more than 50 years ago by V. Bush but T. Nelson is generally credited as the first to coin the term "hypertext" [5]. There are two very different models for hypermedia [7]. The first uses hypermedia to deliver constrained embedded applications, e.g., stand-alone CD-ROM based applications. The second is distributed hypermedia as a world-wide system for information discovery and management, e.g., World Wide Web [1] and Hyper-G [6].

The common model for distributed hypermedia information systems supports three distinct roles. They are *passive participant*, *information/service provider* and *active participation* [7]. The last role is emerging model for supporting collaboration. From the hypermedia processor designer's standpoint, the roles hypermedia systems must support present a unique challenge since most media tasks are computation-intensive and run in parallel. Yet they are required to provide real-time performance. For example, tasks such as video and audio encoding/decoding, text processing, image processing, authentication and encryption/decryption should run in parallel to support hypermedia systems for collaboration.

Unfortunately, we know of no work investigating the synthesis of distributed hypermedia processors that can support collaboration. As quality requirements change,

This work was supported by DARPA under contract F04701-97-C-0010.

it is necessary to take into account not only the timing and synchronization requirements, but the throughput requirements when designing a hypermedia processors. The benefits of advances in compiler technology and architecture also should be incorporated in designing hypermedia systems as low-level programming is practically not able to sustain current advances in VLSI technology.

We present an approach to explore the area optimal system design space exploration for hypermedia applications that support collaboration roles. We focus on a category of processors that are programmable yet optimized to run a hypermedia application. The approach utilizes the modern advances in compiler technology and architectural enhancements that are well matched to the compiler technology. 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. 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.

TARGET ARCHITECTURE

The target architecture we use for the experiment resembles a multiprocessor system with shared memory except that all the processors for a given hypermedia application scenario are assumed to be laid out on a single die. A media task is assigned to its dedicated processor. More than one media application can be assigned to a processor if the given performance constraints of all assigned tasks are guaranteed to be met. Shared memory is used for data communication between tasks. Each processor maintains its own cache. When more than one media 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.

Individual applications are divided into fixed run time units called *quanta*. In our experiment, we use as one quantum length the time taken to encode or decode 4 MPEG-2 frames. For a typical throughput of 15 frames per second, the quantum size is approximately 0.267 seconds. The performance constraints used to produce our experimental results are based on the number of processor cycles equivalent to at least one quantum. One of the benefits of using the notion of quanta is that it simplifies synchronization of several applications running on multiple processors. Aside from optimal use of allocated resources, running given tasks faster than required will provide no benefit to users. Hence, the use of quanta equivalent to the longest time frame with which tasks should be synchronized gives a convenient task assignment unit to allocate resources.

We develop a simple area model based on SA-110. The area of the chip is $49.92mm^2$ ($7.8mm \times 6.4mm$). Approximately 25% of the die area is devoted to the core ($12.48mm^2$). The issue unit and branch unit occupies approximately 20% of the die area ($2.50mm^2$). The integer ALU and load/store unit consume roughly 20%

of the die area ($2.50mm^2$). The DMMU and IMMU (data and instruction MMU) occupies about 40% ($5mm^2$) of the area. The rest of the area is used by other units such as the write buffers and bus interface controller. We assume that the area of miscellaneous units is relatively stable in the sense that it does not change as we increase the issue width or cache sizes. We assume a VLIW issue unit area model which is generally of complexity $O(n)$ or sub-linear. The area of an arbitrarily configured VLIW machine is given by

$$\begin{aligned} Area &= n_{issue}A_{issue} + n_{ALU}A_{ALU} + \\ & \quad n_{branch}A_{branch} + n_{mem}A_{mem} + A_{misc} + A_{cache}. \end{aligned} \tag{1}$$

The terms n_{issue}, A_{issue}, n_{ALU}, A_{ALU}, n_{branch}, A_{branch}, n_{mem}, A_{mem}, A_{misc} and A_{cache} are the issue width, the baseline issue unit area, the number of ALUs, the area of a single ALU, the number of branch units, the branch unit area, the number of memory units, the area of single memory unit, miscellaneous area and cache area, respectively. Cache area is calculated using the Cache Design Tools [3].

PROBLEM FORMULATION

The set of media applications used in this experiment is composed of complete applications which are publically available and coded in a high-level language. We use 8 applications culled from available image processing, cryptography and DSP applications. Detailed descriptions of the applications can be found in [8].

We use the IMPACT tool suit [2] to measure run times of media applications on various machine configurations. 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). A high-level machine description supplied by a user is compiled by the IMPACT machine description language compiler. IMPACT provides cycle-level simulation of both the processor architecture and implementation. The optimized code is consumed by the Lsim simulator. At simulation time, Lsim takes cache structure information provided by a user.

Informally, the problem can be stated as follows: for a given set of media applications and their performance constraints, synthesize an area optimal processor that guarantees the timing requirements of the applications. We define the problem using more formal Garey-Johnson format.

Selection Problem

Instance: Given a set T of n media applications, a_i, $i = 1, 2, ..., n$, a set of m processors, c_j, $j = 1, 2, ..., m$, the run times e_{ij} of the media applications a_i, $i = 1, 2, ..., n$ on the machines c_j, $j = 1, 2, ..., m$ and constants C and E,

Question: Is there a multisubset (subset in which more than one instance of a processor can be included) M of k processors, c_p, $p = 1, 2, ..., k$, $1 \leq k \leq n$, such that $\sum_{j \in M} A_{c_j} \leq C$ and $\max_{j \in M} \sum_{i \in t_j} e_{ij} \leq E$? A_{c_j} is the area of the machine c_j and t_j is the set of tasks that are assigned to the machine j.

Theorem. The Selection Problem is NP-complete.

Proof. The Bin Packing Problem can be mapped to a special case of the Selection Problem. For a given task set T, $|T| = n$ and an integer $k \in \{e | e \in Z, 1 \leq e \leq n\}$, we map $2^n - 1$ objects to $2^n - 1$ subsets (excluding the empty set from the power set of T) of T. The k bins in the BPP are mapped to the k processors.

Since for each k we have $\binom{m}{k}$ problem instances to which a BPP with k bins can be mapped, there are in total up to $\sum_{k=1}^{n} \binom{m}{k}$ NP-complete problem instances. Note, however, that the simulation time to measure run times of each task on each processor takes well over one to two weeks depending on the experiment setup, it is well worth to try to devise an efficient algorithm to obtain the optimum solutions. In the following section, we explain an efficient optimal algorithm for the problem.

SYSTEM SYNTHESIS

We develop a branch and bound based algorithm for the selection of area minimal hypermedia processor configurations. The algorithm examines all possible combinations of media task run times to find a best processor configurations.

Before entering the branch and bound loop, the algorithm finds the area optimal processor for each element of the power set of a given media applications. This step eliminates all processors sub-optimal in terms of running partial task sets the size of which ranges from 1 to n, n being the number of media tasks. If, for a subset of tasks, we cannot find a processor that can satisfy a given timing constraint, then there is no feasible solution that run the particular subset of tasks on a processor.

The run time of the algorithm is dependent on two factors: the size of a given task set and the timing constraint. It is trivial to see that the size of a given task set affects the run time as the combination of tasks is exponential with respect to the number of tasks. As we lower the timing constraint, the number of feasible processors increases. This results in limited pruning of possible combinations of tasks in early stages of the algorithm. In our experiment, which is run on a SPARC4 machines and reported later, the run times of the algorithm ranges from less than a second for the strictest timing constraint to approximately 50 seconds for the least strict timing constraint.

EXPERIMENTAL RESULTS

We evaluate the framework presented in this paper by conducting an experiment with a set of 8 media applications (mpeg2 encoder/decoder, jpeg encoder/decoder, ADPCM encoder/decoder and pegwit encryption/decrytion). The range of the performance constraints we examined is from 5.7×10^6 to 4.47×10^7 cycles, which is the maximum amount of time allotted to finish processing a quantum worth of computation. In our experiment, a quantum size is equal to 0.267 seconds, which implies that the speed constraint of the processors is ranging from 21.37MHz to 167.6 MHz.

Figure 1 shows changes in the number of selected processors and overall area as the performance constraint loosens. As we reduce the throughput requirement, we observe the reduction of the total number of processors as well as the total area of the selected processors needed to satisfy the requirement. The results are consistent

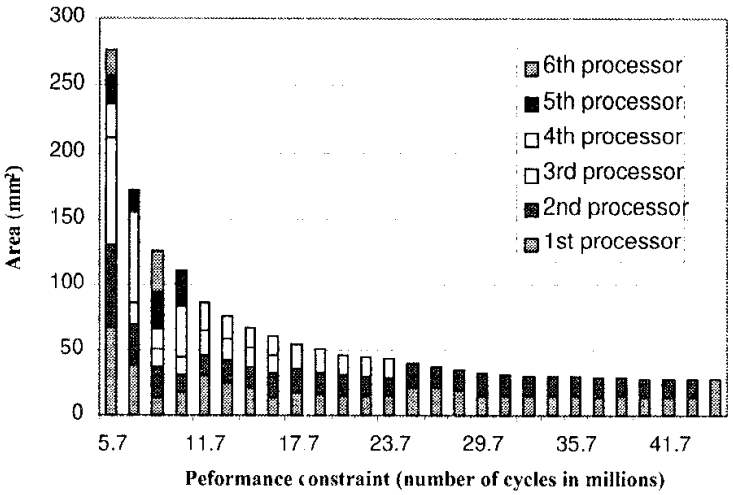

Figure 1: Minimum area configurations for a range of cycle time constrains

with the intuition that for a lower performance requirement more applications can be fit onto single small processor rather than separate multi-issues processors. We also observe that single issue processors are dominant in the performance constraint range from 1.47×10^7 to 4.47×10^7 number of cycles. The reason is that single issue machines give the most performance per area than wider issues machines. The available ILP in the applications does not compensate the increase of the area of wider issues. Therefore, wide issue machines only appear to be viable choices between 5.7×10^6 and 1.17×10^7 where speed is more critical.

The size of branch units also shows the similar characteristics. Only the extreme case where 5.70×10^6 cycles are available would require 2 branch units on a single processor. The main reason is that the higher number of branch units only appear in wide issue machines and for those machines, the available ILP of the applications do not give the equal amount of speed up.

The cache requirement goes down as we reduce the performance requirement. Examining the results further reveals that all the bumps where the total size goes from local minimum to local maximum occur exactly when the number of processors is reduced. The area of a processor saved by reducing the issue width is transfered to a larger cache area so that the applications can be run faster and fit onto smaller number of processors. The same phenomenon is seen in instruction cache also.

The results show that the demand on instruction cache is higher for encoding applications (mpeg2 encoder, jpeg encoder, ADPCM encoder and pegwit encryption) since they are more computationally intensive and have much bigger inner loops than decoding applications (mpeg2 decoder, jpeg decoder, ADPCM decoder and pegwit decryption). For example, *mpeg2enc* requires 50% more cycles to encode the same number of frames than *mpeg2dec* to decode on same processor configuration. Thus, higher I-cache and area usage is seen on encoding applications than decoding at the same performance constraint. However, the D-cache size is roughly the same for

encoding and decoding since the working set and the temporal locality is small.

CONCLUSION

The approach presented in this paper makes use of a production quality ILP compiler, simulators, the notion of multiple-instruction-issue processors and an efficient algorithm to combine processors optimized for an individual task or a set of tasks to minimize the area of resulting hypermedia processor. Although the selection problem is shown to be NP-complete, we found that the optimal solutions can be obtained in reasonable run time for practically sized problems.

We found that there is enough ILP in the typical media and communication applications to achieve highly concurrent execution when throughput requirements are high. On the other hand, when throughput requirements are low, there is no need to use multiple-instruction-issue processors as they provide no desirable benefits.

The framework introduced in this paper is very valuable in making early design decisions such as architectural configuration trade-offs including the cache and issue width trade-off under area constraint, and the number of branch units and issue width.

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] M. J. Flynn. *Computer Architecture: Pipelined and Parallel Processor Design*. Jones and Bartlett, 1996.

[4] K. Gronbaek and R. Trigg. Design issues for a Dexter-based hypermedia system. *Communications of ACM*, 37(2):41–49, February 1994.

[5] F. Halasz. Reflections on note-cards: Seven issues for the next generation of hypermedia systems. *Communications of ACM*, 31(7):836–852, July 1988.

[6] F. Kappe, H. Maurer, and N. Sherboken. Hyper-G: A universal hypermedia systems. *Journal of Educational Multimedia and Hypermedia*, 2(1):39–66, 1993.

[7] M. Kessler. Distributed hypermedia. In *Proceedings of Southcon '95*, pages 190–195, 1995.

[8] 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.