Evaluating Memory Architectures for Media Applications on Coarse-Grained Reconfigurable Architectures *

Jong-eun Lee[†]
jelee@poppy.snu.ac.kr

Kiyoung Choi[†]
kchoi@azalea.snu.ac.kr

Nikil D. Dutt[‡]
dutt@cecs.uci.edu

[†] EECS, Seoul National University, Seoul 151-742 KOREA
[‡] Center for Embedded Computer Systems, University of California, Irvine, CA 92697

Abstract

Reconfigurable ALU Array (RAA) architectures—representing a popular class of Coarse-grained Reconfigurable Architectures—are gaining in popularity especially for media applications due to their flexibility, regularity, and efficiency. In such architectures, memory is critical not only for configuration data but also for the heavy data traffic required by the application. Hence, system designers would like to evaluate the effects of different memory architectures and memory traffic early in the design process. In this paper, we offer a scheme for system designers to quickly estimate the performance of media applications on RAA architectures. The proposed scheme is based on the performance-oriented model of RAA architectures, which we develop to model different memory architectures in a uniform way so as to allow for easy mapping of application loops and early performance estimation. Our experimental results estimating the performance of multimedia applications on three memory architectures demonstrate the flexibility of our memory architecture evaluation scheme as well as the varying effects of the memory architectures on the application performance, which also signifies the need for memory architecture evaluation early in the design process.

1 Introduction

A trend in the architectural platforms for media applications is the adoption of reconfigurable computing elements for cost, performance, and flexibility issues [1]. Coarse-grained Reconfigurable Architectures (CRAs), mostly stressed by reconfigurable computing, are known to be flexible as well as efficient as demonstrated by recent architectures [2]–[4]. Reconfigurable ALU Array (RAA) architectures [4]–[8] form the most popular class of CRAs, which are built on a 2D array of Processing Elements (PEs) communicating via programmable interconnects. Though each PE can perform only a limited set of operations such as addition and multiplication, through dynamic reconfiguration during runtime the 2D array datapath can be programmed to perform different algorithms—typically, critical loops of the application—very efficiently. This regular datapath structure makes the RAA architectures very well suited to media applications, which are also characterized by structured and regular computations on large data sets [9].

* This research was conducted while the first two authors were visiting UC Irvine, and supported in part by grants from NSF (CCR-0203813 and CCR-0205712) and Hitachi Ltd. We also thank members of the UCI EXPRESS compiler team for their assistance.

The acceleration of RAA architectures mainly comes in two areas. First, the **arithmetic** operations can be parallelized on the PE array, with the maximum speedup equal to the number of PEs executing in parallel. Second, the **memory** operations can be implemented in a much more efficient way on RAA architectures by utilizing hardware addressing support provided by the architecture. For example, in MorphoSys [4] the local frame buffer can generate data streams that are sequentially used by the PE array, so that there is no explicit addressing needed for data transfers. Thus, RAA architectures can accelerate not only the memory access operations (through parallel memory access) but also the array index manipulation operations if the array is accessed with a scan pattern supported by the memory architecture.

On the other hand, there is some overhead with RAA execution as well. First, there is reconfiguration overhead whenever a new loop is loaded on the RAA. Also, if a loop uses more than one configuration (because, for example, the loop contains too many operations to be mapped onto the PE array using only one configuration), it may be needed to switch between multiple configurations throughout the loop execution, significantly adding to the reconfiguration overhead. Second, the input data for the PE array may need to be transferred from the main memory to the RAA local data buffer before or during the loop execution. Likewise, the output results may need to be transferred, too. Note, however, that some overhead may effectively be removed by optimizations. For example, the initial reconfiguration overhead of a loop may be hidden by starting the reconfiguration long before the loop is reached.

We target our research at the rapid and quantitative evaluation of RAA architectures for architecture design and exploration. This early evaluation can be very valuable, as today's application mapping is typically done by hand making it virtually impossible to compare and explore various architectural options. To derive first-order performance estimation with reasonable accuracy, we need to identify critical parameters of the architecture that have the biggest impact on the performance of applications. From this point of view, the memory architecture and memory operations are very important, not only because memory operations account for a large portion of the execution cycles for media applications but also because there can be potentially greater diversity in the memory architecture [10] than in the PE array (assuming a fixed granularity for the RAA). Hence, in this paper we focus on the memory subsystem of RAA architectures, although our performance estimation scheme covers the other parts as well. Our performance estimation is based on the *performance-oriented view*, which we present as an abstract model of RAA architectures for early performance estimation. The performance-oriented view has a set of *array-level operations* defined with it, allowing for a more natural representation of RAA architectures as well as an easier mapping of application loops for early performance estimation. We demonstrate the efficacy of our technique through a set of experiments estimating the performance of multimedia applications on three memory architectures. Our experimental results not only demonstrate the flexibility of our memory architecture evaluation scheme but also show that the memory architecture can have quite different effects on the application performance depending on the characteristics of the application, signifying the need for memory architecture evaluation early in the design process.

The rest of the paper is organized as follows. In Section 2 we review some of the related work. In Section 3 we briefly describe the target architecture model and present the performance-oriented view of RAA architectures. In Section 4 we describe our performance estimation flow for media applications, which is based on the performance-oriented view. In Section 5 we present our experimental results using multimedia application benchmarks, and conclude the paper in Section 6.

2 Related Work

Coarse-grained reconfigurable architecture has become an area of active research recently, with the increasing interest in reconfigurable computing in image and video applications [1], [11]. Lee et al. [12] have proposed a generic reconfigurable architecture template called DRAA representing a wide class of RAA architectures, for which a core mapping algorithm (placement and routing) has also been developed for loops based on loop pipelining [13]. Weinhardt et al. [14] have proposed loop pipelining technique to exploit the high degree of parallelism in reconfigurable hardware. In loop pipelining, loops with loop-carried dependency are difficult to get pipelined and achieve high throughput. To address this problem, Bondalapati [15] has proposed a technique called data context switching, which can improve the throughput by exploiting local memory elements to store the contexts. Maestre et al. [16] have provided another level of optimization for RAA architectures, which considers task scheduling and configuration memory management to reduce the configuration switch overhead.

Media applications have been extensively studied as they form a dominant workload in the computer industry. Ienne et al. [17] have shown a limit study on the performance improvement of media applications using the MediaBench application benchmark suite [18]. By incrementally relaxing the microarchitectural constraints of possible coprocessors (called ad-hoc functional units), they show significant speedup of up to 6 times is possible. In their study they find that the ability of ad-hoc functional units to access the data memory is a particularly critical architectural feature. Another study [9] on media benchmarks also confirms the importance of memory accesses in media applications. In an attempt to characterize the media applications on the memory activity, they have found the overall execution time of media applications has a good correlation with the amount of temporary memory accesses, although the memory access instructions are less than a third of the overall instruction mix. These studies point to the need as well as the feasibility of evaluating memory architectures for media applications.

The importance of the memory architecture for media applications have been noted in the context of CRAs as well. As CRAs increase the computation throughput by deep pipelining, it becomes more important for the memory interface to provide an increased data rate to keep up with the computation rate. For this, Herz [10] has presented efficient memory interface architectures that support address generation and data sequencing for CRAs. Although they are very effective compared to software solutions, there is no quantitative study or technique that allows for the rapid evaluation of different options in organizing the memory subsystem of CRAs. In this paper, we address this problem by providing a performance-oriented model of memory architectures and an evaluation scheme for media applications on RAA architectures.

3 Target Architecture Model

In this section we first describe the target architecture, the DRAA (Dynamically Reconfigurable ALU Array). The DRAA represents a broad range of RAA architectures, facilitating compilation as well as the design space exploration of RAA architectures. Next we present the performance-oriented view of the DRAA, developed for early performance estimation.

3.1 DRAA: Generic Architecture Template

Figure 1 (a) illustrates the DRAA, a generic architecture template for RAA architectures. The DRAA [12] serves as an architectural platform which defines, with a set of architectural parameters,

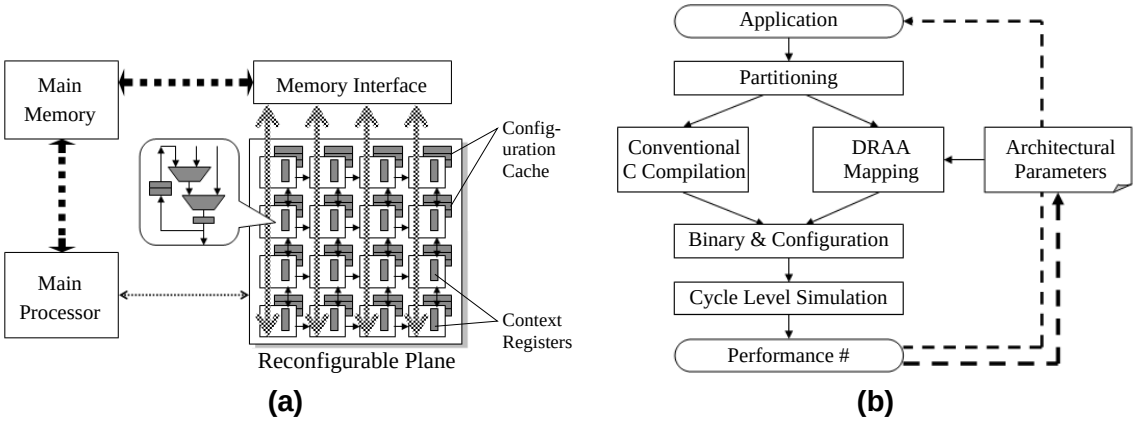

Figure 1. (a) DRAA: Generic architecture template for reconfigurable ALU array (RAA) architectures, (b) simplified compilation and design space exploration flow.

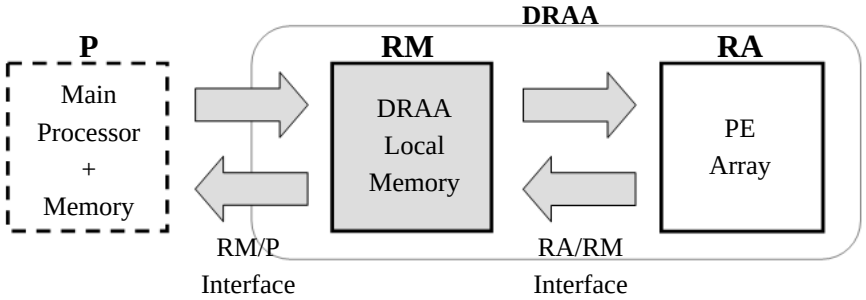

Figure 2. Performance-oriented view of the DRAA architecture.

a class of RAA architectures that are typified by a regular ALU array datapath with a fast memory interface. The architectural parameters include the following.

- the functionality of a PE, as a set of operation patterns supported with one configuration
- the size of the array (#rows and #columns)
- interconnect architecture, assuming all rows (columns) have the same architecture
- the memory interface for each column, assuming similar interface for all columns

The DRAA also provides a view for mapping upper abstraction layers (e.g., behavioral description of the application) to it, enabling development of mapping algorithms. Figure 1 (b) shows a compilation flow highlighting the key steps. The partitioning step identifies kernels (critical loops of the application) that may be mapped to the DRAA. Then the kernels are mapped onto the DRAA using a DRAA mapping tool while the rest of the application is mapped using a traditional compiler. For the DRAA part, the mapping should take into account the architectural parameters, which can be used for design space exploration. Once the binary code (for the main processor) and configuration data (for the DRAA) are generated, one can verify the functionality and estimate the performance through cycle-accurate simulation. For design space exploration, the performance number can be used as a feedback to optimize the architecture and/or the application.

3.2 Performance-Oriented View

While the compilation/simulation flow in Figure 1 (b) can be used to accurately evaluate RAA architectures for a given set of applications, there are problems with that approach. First, the compilation for DRAAs is far from automated as in the compilation for general purpose processors; rather, a typical approach to application mapping is based on manual effort and designers' intuition. The lack of an automated mapping flow with full optimization is an important difference distinguishing RAAs from general purpose processors. Second, if the evaluation relies on repeated compilation and simulation for every change in the architecture or the application, it would be nearly impossible to explore the huge design space created by the numerous architectural options. Yet, to estimate the performance we need to map the application to the architecture somehow, without the detailed compilation.

To obtain early performance estimation without the detailed compilation, we identify performance-critical parameters of the architecture in a model that can be easily related to applications. This is well summarized in our performance-oriented view, an abstract model of the DRAA architecture developed for early performance estimation. The performance-oriented view emphasizes on the memory subsystem architecture, as RAA architectures often have specialized memory interfaces to accelerate the large amount of memory operations for media applications. Also, it defines a set of array-level operations that can be used to map application loops onto it, to generate performance estimation results. This performance-oriented view can be used to represent a range of detailed architectures including MorphoSys [4] and KressArray [5] for the purpose of performance estimation.

3.2.1 Performance-Oriented View

The performance-oriented view (Figure 2) essentially provides a simplified view of detailed memory architectures, identifying the performance-critical information (compare with detailed memory architectures in Figure 4). This model has two components: the PE array (RA) and the memory subsystem. The memory subsystem further consists of three components: the DRAA local memory (RM), the RA/RM interface, and the RM/P interface (P stands for the processor-memory subsystem).

- The RM represents the local data memory often present in the DRAA, which is crucial to reduce the memory access cycles (through data caching) for media applications. This memory has a finite capacity, which will determine the amount of data caching.
- The RA/RM interface represents the memory interface of the PE array, typically with large bandwidth. The RA/RM and the RM/P interfaces have generic array-level operations associated with them.
- The RM/P interface, which represents the main memory interface of the RM, often employs the DMA (Direct Memory Access) capability.

Besides the parallel memory access and data caching (to exploit data locality), the memory subsystem may offer hardware addressing support to the PE array. For deterministic memory accesses (especially when dealing with stream data), the memory subsystem may be instructed to provide data according to the scan pattern of the application, thus eliminating the need for the PE array to generate addresses and requests to the memory subsystem. This type of addressing support can be regarded as having AGUs (Address Generation Units) in RA/RM or RM/P interfaces, that support a set of pre-defined scan patterns. Then the main processor can select a scan pattern in a similar way to configuring the PE array.

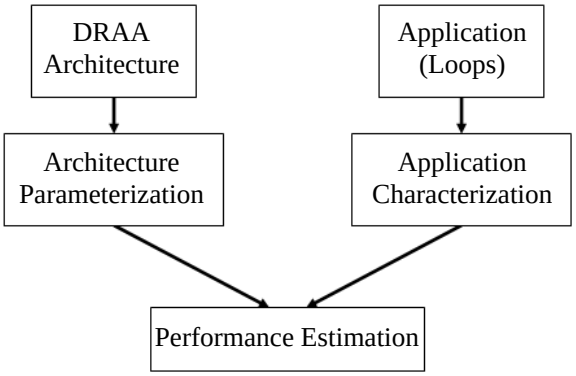

Figure 3. DRAA performance estimation flow.

3.2.2 Array-Level Operation

Since typically only loops are mapped to the DRAA, the arithmetic operations performed by the PE array are on arrays (or streams) of data. Similarly, the memory operations requested by the PE array can also be seen at the array level. The RM/P and RA/RM interfaces define generic operations at the array level to facilitate the mapping of application loops onto the architecture.

Each of the RM/P and RA/RM interfaces has two generic, array-level operations associated with it. For instance, the RA/RM interface has two data transfer operations, i.e., from RM to RA and from RA to RM. Also, each data transfer operation has parameters associated with it such as the bitwidth and the number of data items to transfer, so that specific architectures can define a performance estimation model (cycle count estimation rule) for each generic operation. These generic operations can be used to map the memory access operations in the application, eventually to generate the early performance estimation results.

4 Performance Estimation

4.1 Estimation Flow

We now describe a performance estimation plan based on the performance-oriented view presented in the previous section. The estimation flow (Figure 3) has the three main steps listed below.

- Architecture Parameterization: For the given architecture, performance-critical parameters identified in the performance-oriented view are captured. This includes annotating the performance-oriented view with (1) its hardware capability such as the set of supported scan patterns and (2) a cycle count estimation rule for each generic, array-level operation defined in the view.
- Application Characterization: Here, the application is a set of candidate loops, which may be mapped to the DRAA. The application loops are analyzed into a series of array-level operations that will be later mapped to operations in the performance-oriented view of the DRAA.
- Performance Estimation: In this step, it is decided whether each loop can be executed (mapped) on the DRAA by comparing the architecture-provided features with the application-required features. If they are compatible, DRAA execution cycles are estimated, simply by matching the operations of the loop with the operations of the architecture and applying the cycle count estimation rules of the operations.

4.2 Architecture Parameterization

The given DRAA architecture is parameterized in two areas: the memory subsystem and the PE array. For the configuration part, we assume that the initial configuration overhead (of a loop) does not affect the final performance significantly, as it can be easily hidden by initiating the reconfiguration early. However, the reconfiguration overhead due to the configuration switch during execution of each loop can be significant and is considered in the PE array cycle count estimation. For the cycle count estimation of the PE array, we use a simple statistical model based on the number of operations, the number of iterations, and loop-carried dependency.

4.3 Application Characterization

From the application description, we extract the following information.[1] We assume that conditional operations (if-else constructs) are converted into data-dependent operations if the DRAA architecture supports predication in the PE array.

- basic loop information: the repetition count and loop-carried dependency
- computation: the number of operations in the loop body (for the statistical model), whether there is any special operators, etc.
- memory access: the number of data items (both loop-invariant and per-iteration), the scan pattern, etc.

Assuming that the mapping is based on loop pipelining for high throughput, each loop can be uniformly represented as having three parts: prolog, steady state, and epilog. The prolog includes data transfers from the P to the RM and/or from the RM to the RA. This is where loop-invariant data or look-up tables are also arranged before iterations start. The steady state can include all types of data-transfers and computation operations for each iteration. Finally, the epilog includes the data transfers back to the main memory.

4.4 Performance Estimation

The last step estimates the number of execution cycles of each loop on the DRAA and finds the speedup over the software execution (on the main processor). It first decides whether a loop can be mapped to the DRAA, by matching the features required by the loop with those supported by the architecture. Next, if it is possible to map to the DRAA, it estimates the number of execution cycles for the DRAA execution, by applying the cycle count estimation rules of each operation. The number of execution cycles for software execution can be estimated using conventional performance estimation techniques or obtained directly through compilation and cycle-accurate simulation.

5 Experiments

In this section we demonstrate the efficacy of our early performance estimation scheme through a set of experiments estimating the performance of multimedia applications on different memory architectures.

[1] Currently, this step is done manually for each application; however, it is also possible to generate this information automatically from the program description using compiler technologies.

5.1 Experimental Setup

To evaluate the effects of different memory architectures on the performance of media applications, we use three benchmarks (each with two modes) from the MediaBench benchmark suite [18] as listed in Table 1. We compare three memory architectures which are illustrated in Figure 4. The three architectures, differing only in the memory subsystem, are called A, B1, and B2. The memory architecture of A is similar to that of KressArray [5]. It features hardware address generation support through Generic Address Generator (GAG) and Scan Window, which basically can provide up to seven address sequences supporting a set of scan patterns. The memory architectures of B1 and B2 are similar to that of MorphoSys [4]. The difference of these from the architecture A is that these architectures have much more efficient memory interface[2] between the RA and the RM; however, this interface requires that the data arrays are put in the frame buffer in the order that they are used (scan pattern). Therefore, in the architectures B1 and B2, it may take more cycles to transfer data from the main memory (M) to the RM because the arrays have to be put in order and some data may be duplicated, resulting in an inefficient use of the memory.[3]

In B1 and B2, to arrange the data according to the application's scan patterns may take a significant amount of cycles if it is done by the main processor. In the architecture B1, either this is done by the main processor or the loops exhibiting non-trivial scan patterns are not mapped to the DRAA. The architecture B2 is an improved version of B1 by adding AGU capability to the DMA unit that is already present in B1. We assume the set of scan patterns supported by the architecture B2 is the same as that of the architecture A.

For our experiments, we assume that the three architectures have the same PE array, which means they are equally capable in terms of performing arithmetic operations. For each architecture, however, we varied the capability of the PE array as in the following.

- **Base**: PE array supports just the arithmetic operations.
- **Predicate**: Predication support is added to the PE array to be able to map conditionals (e.g., data-dependent **if**) within loops. This could be easily implemented by adding muxes in the PE. The PE array size is still limited.
- **Unlimited**: There are enough number of PEs, so that the latency of the arithmetic operations solely depends on the length of the longest chain of dependent operations. Special operators (e.g., division) are also supported in one cycle.

Thus, in total nine architecture combinations are evaluated for each application. These experiments not only enable us to see the variance due to the PE array but also demonstrate the capability of the performance estimation framework to explore different architectural options in the DRAA.

[2] There are 16 parallel buses (each 8 bits) between the RA and the RM.

[3] In these experiments, however, the RM is assumed to have enough capacity.

Table 1. Benchmark applications

	EPIC		GSM		MPEG2	
Properties	Enc	Dec	Enc	Dec	Enc	Dec
#cycles on SH-3	78M	16M	240M	89M	2,205M	144M
#kernels	7	3	7	4	6	2
Exec time in kernels	87.7%	74.6%	78.9%	89.7%	81.9%	67.7%
Perf. imprv potential	8.1	3.9	4.7	9.7	5.5	3.1

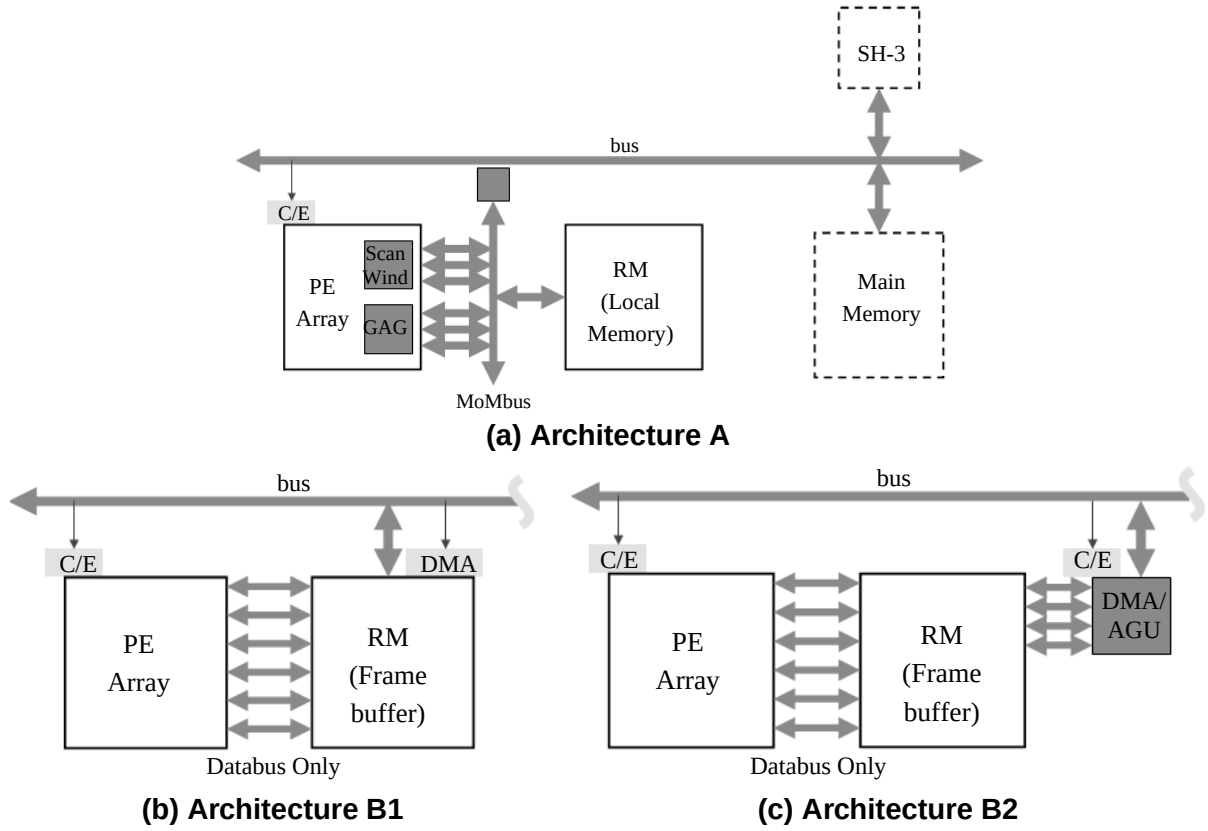

Figure 4. Three memory architectures, with the common main processor-memory subsystem shown only in (a). 'C/E' and 'DMA' represent controllers for configuration and execution, and direct memory access, for the attached hardware, respectively.

The performance estimation procedure is as follows. First, we parameterized the three DRAA architectures. For a fair comparison, we assumed the main memory access parameters are the same for all the three cases. Second, we profiled the applications and partitioned the applications based on the profiling results. Third, we ran the cycle-level simulation to get the number of execution cycles of each kernel (and those mapped to the main processor) on the SH-3 processor [19]. Table 1 lists statistics after partitioning. Finally, we characterized the candidate loops and estimated the performance according to the proposed estimation flow.

5.2 Results

Figure 5 summarizes the performance improvement results estimated for the six applications. For each application, nine architecture combinations were evaluated: the three memory architectures with the three levels of the PE array capability. Among the three levels of PE array capability, the **unlimited** (the top of a bar) represents the case when the PE array capability is pushed to the limit; therefore, this level of PE array capability may not be realistic in some cases. The performance improvement estimates in the figure do not necessarily mean that they are the maximum improvements possible with the DRAA for the application, since there are other factors affecting

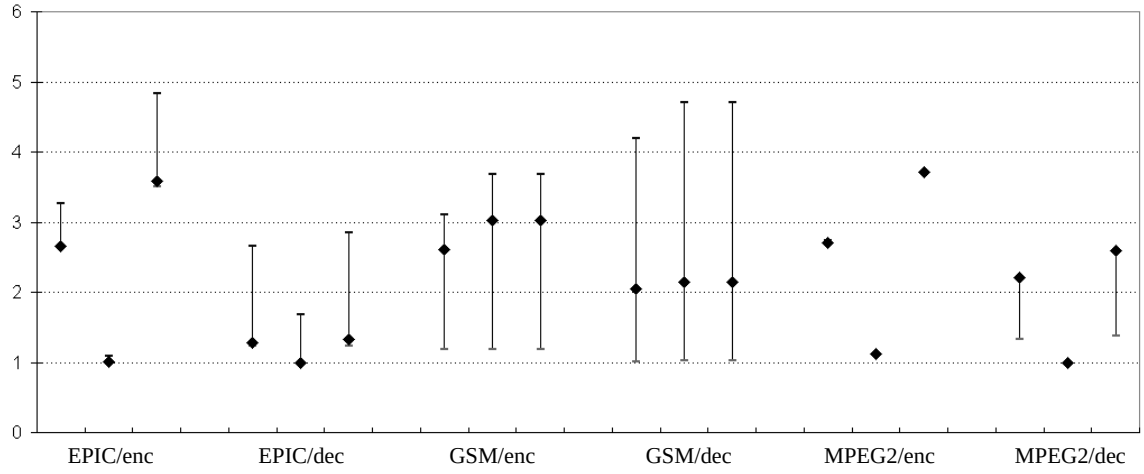

Figure 5. Comparing performance improvement with the three architectures. Three vertical bars plus dots per each application correspond to the three architectures, A, B1, and B2, from left to right. Each vertical bar and dot represents the performance improvement over all-software mapping: the bottom is for the base PE array, dot for the predicate PE array, and the top for the unlimited PE array.

the performance as well. Partitioning, for example, can make a significant difference in the overall performance improvements. In our experiments, we focus on the study of the effects of different architectural options using a fixed partitioning, which generates relatively small number of kernels for the sake of tractability in conducting the experiments.

From the results, we see that the capability of the PE array may play a big role in the overall performance, although the cycle count estimates for the PE array may not be precise (being based on a statistical model). In the figure, the distance between a dot and the bottom of a bar represents the additional performance improvement due to the predication support in the PE array. In applications such as GSM/enc, GSM/dec, and MPEG2/dec, we observe that the predication support is important in boosting the performance with DRAAs. The distance between a dot and the top of a bar, on the other hand, represents the additional performance improvement due to either the special operator support or the unlimited size of the PE array. In EPIC/enc and EPIC/dec, it is due to the special operator support (integer division) whereas in GSM/enc and GSM/dec, it is due to the unlimited array size.

From the figure, we also note that there are significant variations in performance due to the different memory architectures in many applications. For instance, the performance improvement of EPIC/enc application can be 1.01 times (almost no improvement with B1) to 2.65 times (with A) to 3.59 times (with B2), for the **predicate** PE arrays. The performance improvement difference between B1 and B2 is due to the AGU support added in the memory subsystem. In most applications this difference is considerable, which points to the significant role of memory architecture capability. Furthermore, even with the same AGU support in A and B2, there are significant variations in performance for such applications as EPIC/enc, GSM/enc, MPEG2/enc, and MPEG2/dec. These results show the importance of considering the memory architecture early in the design process.

Sometimes memory operations can be the most significant factor determining the performance. This is most exemplified in such applications as MPEG2/enc and MPEG2/dec, where very little improvement is observed even with the unlimited number of PEs, showing the dominant importance of the memory subsystem architecture in those applications.

6 Conclusion

In this paper, we presented an early performance estimation scheme that is based on the performance-oriented view of the DRAA architecture. The performance-oriented view, which can represent various memory architectures for performance estimation, provides a simplified view of the memory subsystem of DRAA architectures. Since typically loops are mapped to the DRAA architecture, the array-level operations, defined with the performance-oriented view, allow for an easy mapping and performance estimation of application loops on the DRAA architecture. Although the performance model needs to be calibrated against reference architectures for more accurate estimation, our initial experiments using multimedia benchmarks comparing three memory architectures show many interesting results and also demonstrate the flexibility of our memory architecture evaluation scheme. An interesting result from our experiments is that the memory architecture can have quite different effects on the application performance depending on the characteristics of the application, which also highlights the need for memory architecture evaluation early in the design process.

References

[1] A. Bindra. Reconfigurable architectures chart a new course for DSPs. *Electronic Design*, August 5 2002.

[2] R. Hartenstein. A decade of reconfigurable computing: A visionary retrospective. In *Proc. DATE*, 2001.

[3] S. Goldstein et al. PipeRench: A coprocessor for streaming multimedia acceleration. In *Proc. ISCA '99, Atlanta*, pages 28–39, 1999.

[4] H. Singh et al. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Computers*, 49(5):465–481, May 2000.

[5] R. Hartenstein et al. Using the KressArray for configurable computing. In *Proc. SPIE Vol. 3526*, Configurable Computing: Technology and Applications, pages 150–161, 1998.

[6] T. Miyamori and K. Olukotun. REMARC: Reconfigurable multimedia array coprocessor. In *Proc. ACM/SIGDA FPGA*, page 261, 1998.

[7] V. Baumgarte et al. PACT XPP — A self-reconfigurable data processing architecture. In *Proc. Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2001.

[8] P. Mannion and R. Wilson. Processor array alters approach to 3G basestations. *EETimes*, December 2 2002. http://www.eetimes.com/story/OEG20021202S0059, last accessed Jan/30/2003.

[9] B. Lee and L. John. Implications of programmable general purpose processors for compression/encryption applications. In *Proc. Application-Specific Systems, Architectures, and Processors (ASAP)*, 2002.

[10] M. Herz. *High performance memory communication architectures for coarse-grained reconfigurable computing systems*. PhD thesis, Kaiserslautern University, Germany, 2001.

[11] Elixent gets cash to bring ALU array to market. *Electronicstalk.com news*, July 31 2001. http://www.electronicstalk.com/news/exi/exi100.html, last accessed Jan/30/2003.

[12] J. Lee, K. Choi, and N. Dutt. Compilation approach for coarse-grained reconfigurable architectures. *IEEE D&T*, 20:26–33, January/February 2003.

[13] J. Lee, K. Choi, and N. Dutt. An algorithm for mapping loops onto coarse-grained reconfigurable architectures. *Proc. ACM SIGPLAN Languages, Compilers, and Tools for Embedded Systems (LCTES)*, June 2003.

[14] M. Weinhardt and W. Luk. Pipeline vectorization. *IEEE Trans. CAD*, 20:234–248, February 2001.

[15] K. Bondalapati. Parallelizing DSP nested loops on reconfigurable architectures using data context switching. In *Proc. DAC*, pages 273–276, 2001.

[16] R. Maestre et al. A framework for reconfigurable computing: Task scheduling and context management. *IEEE Trans. VLSI Systems*, 9(6):858–873, 2001.

[17] P. Ienne et al. On the limits of processor specialisation by mapping dataflow sections on ad-hoc functional units. Technical Report 01/376, Swiss Federal Institute of Technology Lausanne (EPFL), 2001.

[18] C. Lee et al. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proc. MICRO-30*, pages 330–335, 1997.

[19] Hitachi, Ltd., http://www.hitachi-eu.com/hel/ecg/products/micro/pdf/sh7700p.pdf. *SH-3/SH-3E/SH3-DSP Programming Manual*, 2000.