

# Architectural Adaptation in MORPH

Rajesh K. Gupta<sup>a</sup>

Andrew Chien<sup>b</sup>

<sup>a</sup>Information and Computer Science, University of California, Irvine, CA 92697.

<sup>b</sup>Computer Science and Engg., University of California, San Diego, CA 92093.

## Abstract

We present methods for architectural adaptation that use application-specific hardware assists and policies to provide substantial improvements in performance on a per application basis. We have used architectural customization to improve performance of the memory hierarchy and utilize network bisection for the multiprocessor architectures. We demonstrate the utility of architectural customization in efficient memory hierarchy management and memory bandwidth requirements using an application in sparse matrix manipulations. The experimental work is presented in the context of the MORPH machine that is currently being designed to provide high system performance by directly addressing memory system limitations in the current machines.

Based on our preliminary results, we propose that an application-driven machine customization provides a cost effective way to achieve high performance and combat performance fragility while maintaining application retargetability across architectures.

**Keywords:** architectural adaptation, co-design, memory hierarchy, prefetching

## I. INTRODUCTION

Embedded computing systems use a combination of hardware and software elements to physically divide the implementation task. Therefore, the most common architecture in these systems can be characterized as one of *co-processing*, i.e., a processor working in conjunction with dedicated hardware to deliver a specific application. Figure 1 shows schematic of a co-processing architecture. The specific implementations of the co-processing architecture vary in the degree of parallelism supported between hardware and software components. For instance, the co-processing hardware may be operated under direct control of the processor which stalls while the dedicated hardware is operational [3], or the co-processing may be done concurrently with software [5]. The instruction and data paths to the processing and co-processing hardware may or may not be shared depending upon application characteristics [8]. Similarly, a co-processing architecture may use more than one CPU to divide system functionality appropriately in software. The co-processing hardware is generally designed or synthesized using high-level synthesis tools. The architecture synthesis problem in high-level synthesis consists of determination of a schedule of operations in the target application, its implementation on specific hardware functional units (or resources) and generation of an appropriate controller (see for instance [4, 9]).

A system generated using this approach typically can not be retargeted to another application without repartitioning hardware and software functionality and reimplementing the co-processing hardware even if the macro-level organization of the system components remains unaffected. Of course, for each such division of hardware and software

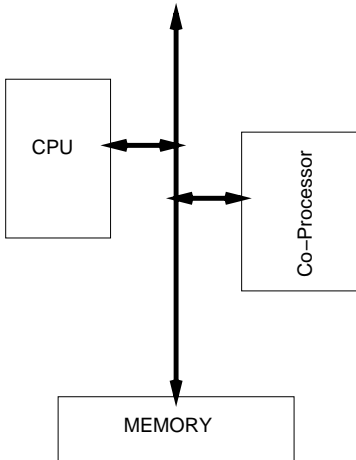


Fig. 1. A Co-processing Architecture

the design of the co-processing hardware may require an entirely different set of resources to deliver the needed performance gain. This presents a challenging problem for the system designer who must often find ways of retargeting a machine across different applications (or even application domains).

In this paper, we present a solution to this problem of machine retargetability by defining an architecture that places strong emphasis on customizable architectural mechanisms rather than synthesizing whole or part of an application. This presents an improvement over existing approaches to configurable computing by preserving machine usability through software; and over traditional computer architecture by providing application-specific hardware assists.

## II. ARCHITECTURAL ADAPTATION

Architectural adaptation refers customization of a machine organization to match an application. This is in contrast to architectural synthesis that attempts to build a machine organization for a given application. Architectural adaptation can be used in the bindings, mechanisms, and policies on the interaction of processing, memory and communication resources while keeping the macro-level organization same. Thus, while the machine architecture is changed to suit an application, the programming model, used in developing applications, is preserved. As mentioned earlier, this approach differs from traditional co-processing architectures where application-specific hardware assists are used to delegate parts of system functionality to a sequentially or concurrently running hardware. In this approach the entire application remains in software while the underlying hardware is adapted for system performance. Figure 2 shows a schematic comparison of the architectural adaptation versus co-processing using hardware assists.

The MORPH machine uses reconfigurable logic blocks integrated with the system core to control policies, interactions, and interconnections of memory to processing.<sup>1</sup> We use a basic machine architecture that is based on the premise that communication is already critical resource in most computing systems and increasingly getting so even for small-scale embedded systems [6]. Thus flexible interconnects can be used to replace static wires at competitive performance. The key elements of the baseline architecture include processing elements and memory elements embedded in a scalable interconnect. Architectural adaptation affects memory interleaving (physical memory element mapping), cache organization and policies [2]. Depending upon the hardware technology used and the support available from the runtime environment this adaptation can be done statically or at run-time. In the following section we describe a specific mechanism for latency management that is shown to provide significant performance boost for the class

<sup>1</sup>More information on MORPH is available at “<http://heze.ics.uci.edu/morph/>.”

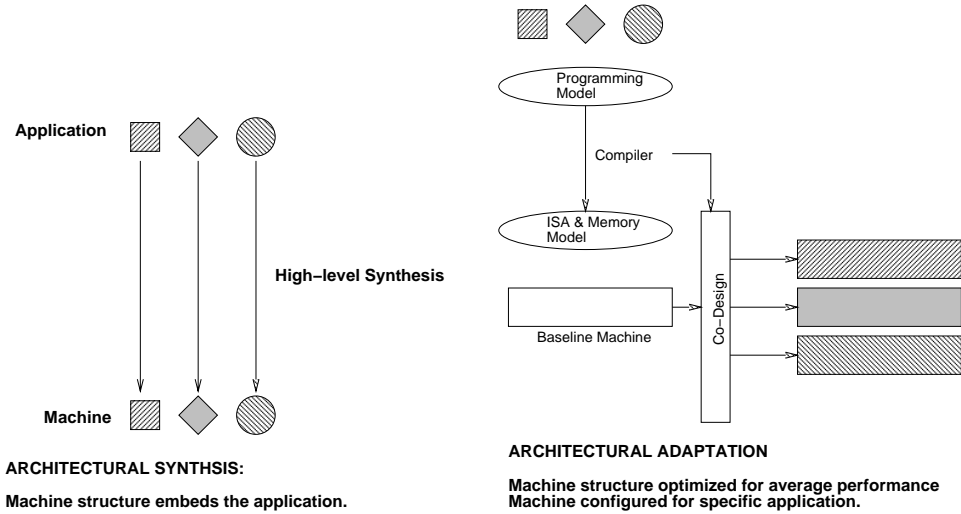


Fig. 2. Architectural Synthesis versus Architectural Adaptation

of applications characterized by frequent accesses to linked data structures scattered in the physical memory. This includes algorithms that operate on sparse matrices and linked trees.

### III. ADAPTATION FOR LATENCY MANAGEMENT

Latency management refers to techniques for hiding long latencies of memory accesses by useful computation in the computing element. Most common technique of latency hiding is by prefetching of data to the CPU. Prefetching is combined with smaller and faster (cache) memory elements that attempt to prefetch application context(s) rather than single data elements. The pointer-based accesses to data items in memory hierarchies typically yield poor results because the indirection introduces main memory and memory hierarchy latencies into the innermost computational loop. Techniques such as software prefetching (loop unrolling and hoisting of loads) do not adequately solve the problem, as prefetching at the processor leaves multi-level memory hierarchy latency in the critical path. Purely hardware prefetching [10] is also often ineffective because the address references generated by an application may contain no particular address structure. We use an application-specific prefetching scheme that resides in dedicated hardware at arbitrary levels of the memory hierarchy, in all of them, or to bypass them completely. This hardware performs application-specific prefetching, based on the address ranges of data structures used. When there is a reference to an address inside in this range, the prefetch hardware will prefetch the “next” element pointed to by the current element. The pointer field for the next element can be changed at runtime. The following code fragment in HardwareC [7] describes the prefetcher function:

```
process Prefetch (pen, addr, sizeofElement, rowPointer,
  colPointer, sizeBlock, startBlock, which, ploc, ready)
[
  ..port declarations..
  ..variable declarations..
  ready = 0; nshift = 0;
  while (sizeofElement > 1) {
    sizeofElement = sizeofElement >> 1;
    nshift = nshift + 1;
  }
  if ((startBlock <= addr) &&
    (addr < (startBlock + sizeBlock))){
    if (which == 0)
      whichPtr = rowPointer;
    else
      whichPtr = colPointer;
    ploc = (startBlock + whichPtr +
```

```

    ((addr-startBlock) >>nshift <<nshift;
  } else
    ploc = 0;
  < write ploc_out = ploc; write ready = 1;>
]

```

This prefetch hardware is combined, for some applications, with a address translation and compaction hardware in the memory controller that works well with data structures that do not quite fit into a single cache line. The address translation is done transparently from the application using hardware assist **translate** in the cache controller and a corresponding hardware assist **gather** in the memory controller as described by the pseudo-code below:

```

process translate(maddr_in, rst, maddr_out)
  ..port declarations..
[
  static m[8]; static new_offset[8]; static offset[8];
  static block_number[8]; static rv[8];

  write maddr_out = 0; load m = maddr_in;
  while(!rst);

  offset = (m - start_block) & element_size_mask;
  block_number = (m - start_block) >> element_size_shift;
  if (start_block <= m && m < end_block) {
    if (offset < offset_2)
      new_offset = offset - (offset_1 - packed_offset_1);
    else
      new_offset = offset - (offset_2 - packed_offset_2);

    rv = start_block+block_number <<packed_element_size_shift+
      new_offset;
  } else
    rv = m;
  write maddr_out = rv;
]

process gather(compacted_maddr, rst, b1o, e1o, b2o, e2o)
  ..port declarations..
[
  static start_block_addr[8];
  static b1[8]; static e1[8]; static b2[8];

  <write b1o=0; write e1o=0; write b2o=0; write e2o=0;>
  while(!rst);

  if(start_block <= compacted_maddr &&
    compacted_maddr < packed_end_block) {
    start_block_addr = ( compacted_maddr - start_block)
      >> packed_element_size_shift
      << element_size_shift + start_block;

    b1 = start_block_addr + offset_1;
    e1 = start_block_addr + end_offset_1;
    b2 = start_block_addr + offset_2;
    e2 = start_block_addr + end_offset_2;
  } else {
    b1 = compacted_maddr;
    e1 = compacted_maddr + 3;
    b2 = 0; e2 = 0;
  }
  < write b1o = b1; write e1o = e1;
  write b2o = b2; write e2o = e2;>
]

```

#### IV. RESULTS AND CONCLUSIONS

We use Sparse Matrix library version 1.3 available from **netlib** written by Kundert, *et. al.*, which uses an efficient pointer based storage scheme to represent non-zero elements. In 2D matrices, the elements in every row and column are connected by a linked list of fixed-size elements. Memory hierarchy simulations were done using a program-driven simulator MORPHSIM [1] that models the system architecture including hardware assists as a compiled simulator that executes in parallel with the application code.

Table IV shows the hardware costs associated with specific hardware mechanisms studied in a semi-custom technology using standard logic cells provided by LSI logic 10K family of gates. The performance gain due to these hardware assists (as shown in accompanying viewgraphs) is shown as a 10X reduction in the cache miss from 24.5% to 2.58% for reads and from 4.9% to 0.92% for writes when using the prefetcher with translate and gather hardware. The

Hardware Block	Cost	Delay (Cycles)
Prefetcher	4083	3
Gather	627	3
Translate	557	2

TABLE I  
HARDWARE COSTS

hardware assists also lead to a 100X reduction in data traffic from 552 MB to 6 MB. It is important to note that these performance advantage are only specific to the application, and indeed across applications these vanish pretty quickly [2]. Therefore, identification of such hardware/software assists for a given application is an important part of the system design. This process can generally be assisted by the application developer. Once identified, co-design of the architectural assists along with its exploitation in the application and/or compilation tools provides a challenging area for hardware/software co-design. Since the typical ratio between peak and average performance of most computing based machine across applications is large and getting worse with advances in device technology [6], architectural mechanisms and co-design techniques that exploit this gap to improve system performance are needed. System co-design using this approach presents a way to utilize application-specific hardware much more effectively than would be the case when part of an application is implemented in hardware as is the case in co-processing architectures.

#### ACKNOWLEDGMENTS

The work was sponsored by support from National Science Foundation CAREER award number MIP 95-01615, NSF/DARPA ASC-96-34947 and from NSF EEC 89-43166.

#### REFERENCES

- [1] CHIEN, A., DASDAN, A., GUPTA, R., AND ZHANG, B. Rapid Architectural Design and Validation Using Program-Driven Simulations. In *Symposium on High-level Design, Validation and Test (HLDVT)* (Nov. 1996).
- [2] CHIEN, A. A., AND GUPTA, R. K. MORPH: A System Architecture for Robust High Performance Using Customization. In *Proceedings of the The Sixth Symposium on The Frontiers of Massively Parallel Computation (Frontiers'96)* (Oct. 1996), pp. 336–345.
- [3] ERNST, R., HENKEL, J., AND BENNER, T. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Design & Test of Computers* (Dec. 1993), 64–75.
- [4] GAJSKI, D., DUTT, N., WU, C. H., AND LIN, Y. L. *High-level Synthesis: Introduction to chip and system design*. Kluwer, 1992.
- [5] GUPTA, R. K., AND MICHELI, G. D. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design & Test of Computers* (Sept. 1993), 29–41.
- [6] KOGGE, P. M. Summary of the architecture group findings. In *PetaFlops Architecture Workshop (PAWS)* (Apr. 1996).
- [7] KU, D., AND MICHELI, G. D. HardwareC - A Language for Hardware Design (version 2.0). CSL Technical Report CSL-TR-90-419, Stanford University, Apr. 1990.
- [8] MADSEN, J., AND BRAGE, J. P. Codesign Analysis of a Computer Graphics Application. *Design Automation for Embedded Systems 1*, 1-2 (Jan. 1996), 121–145.
- [9] MICHELI, G. D. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.
- [10] ZUCKER, D. F., FLYNN, M. J., AND LEE, R. B. A comparison of hardware prefetching techniques for multimedia benchmarks. Technical report CSL-TR-95-683, Computer Systems Laboratory, Stanford University, Stanford University, CA 94305, December 1995.