A Web/DVD-Based Multimedia Architecture Simulator

Erica Asai, Israel Koren and C. Mani Krishna

Abstract—

This paper describes a Web/DVD-based Multimedia Architecture Simulator and courseware which we have developed for a course module titled "Multimedia Architectures." The goal of this module is to study architectural enhancements to microprocessors to support multi-media applications. The course-ware provides lecture material including audio/video lecture recordings and slides, references to relevant websites, a search engine, and various Java applets and the Multimedia Architecture Simulator. The Simulator is Web/DVD-based and runs on-line.

Most of the existing simulators require the user to go through the steps of downloading, installing, and configuring the software. Our on-line simulator has the advantage of being able to run directly from the web browser without any configuration.

The Simulator has the complete functionality of a multimedia oriented computer system which effectively simulates an SIMD (Single Instruction Multiple Data) architecture. The simulator can compile, run, and debug image processing programs which can readily be presented on the screen. Using program examples and projects, students can experiment with the concepts introduced in the lecture.

The integration of the web-based simulators, lectures, references and search mechanism on-line has created an effective course environment.

Keywords— **Web-Based Simulator, Distant Learning.**

I. Introduction

Various online lectures series are available to date [1], but only a few provide a platform that allows the student to experiment with the concepts that the lectures introduce, such as a full-fledged computer architecture simulator. The learning process consists of forming abstract concepts from the information provided and then experimenting with the concepts in a real environment. Simulators provide the ideal environment for experimenting with the conceptual model.

Most existing simulators cannot be invoked from a web browser [2]. Moreover, the contents of the lectures are not always reflected in the actual implementation of the simulator.

To achieve the goal of integrating course material and a platform where students can experiment with the concepts introduced in the lectures, we have developed a Web/DVD-based Multimedia Architecture Simulator, as a part of a course module. The goal of this module is to study architectural enhancements to support multi-media applications. The main purpose of the simulator is to demonstrate the benefits of these architectural enhancements in practical multimedia programs. The simulator is an effective tool to identify such benefits.

The simulator supports all the significant features of a complete computer architecture. It has the capability to edit, compile, run and debug programs and then execute two algorithms in parallel for comparison. It is ready to run in the location where the information is, and can access any files available online. The simulator is machine-independent and thus open to the widest public.

Several image-processing programs are included with the simulator for demonstrating the algorithms introduced in the course. These programs are implemented with and without multimedia extension instructions to illustrate the performance impact of these instructions.

The rest of the paper is organized as follows: Section II briefly describes the complete course environment. Section III describes the functions of the simulator. Section IV concludes the paper.

II. The Course Website

The website contains the following course material:

Lecture Slides – The slides presented in the lecture, with audio/video narration. This also includes

Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, E-mail: {easai, krishna, koren}@ecs.umass.edu

animation algorithms and on-line simulators.

Supplementary Lectures on the web – Additional slides for supplementary information that augment the lecture contents. The materials included are: introductory computer architecture tutorials on pipelining, cache, and case studies for Intel, AMD and Sun multimedia architecture [3], [4].

Review Questions – Review questions to summarize the lectures.

Terms and Abbreviations – Brief explanation of terms and abbreviations.

The Multimedia Architecture Simulator – The simulator can run demonstrations and sample programs. Its manual includes tutorials and sample programs, and an its instruction set programming manual.

Links and Reference – Links and references to relevant sites including industrial and academic sites.

Search Engine – Search engine that can search the entire set of slides within this site for a keyword.

The lecture slides contain the slides presented in the lecture and the corresponding audio/video narration. The audio/video clips are encoded in RTSP (Real Time Streaming Protocol) to minimize the required bandwidth. The synchronization of the slide images and the narration is specified in SMIL (Synchronized Multimedia Instruction Language).

III. Simulators

Several simulators have been developed and are currently available on the course website for simulating the algorithms introduced in the lectures. The advantages of web-based simulators are:

- No installation is necessary. The simulator can run directly from the web browser.
- It can process images on the website on any computer platform.
- Manuals and references are available online.

From the point of view of the website administrator, web-based applications eliminate platform-dependent problems and lower distribution costs. The materials and resources can be distributed over the network. It is architecture-neutral and is suited to diverse educational environments.

The simulators are implemented in Java [5] on an Intel-based Linux platform. The OOP (Object Oriented Paradigm), the GUI (Graphical User Interface) library, and the network programming capability of Java allowed us speedy development and user friendly presentation. The GUI is based on the Java AWT library for network compatibility. Further details regarding the implementation and the class hierarchy of the simulator can be found in on-line manual [6].

A. Java Applets

The Java applets on the course website complement the lectures. Students can experiment with example images using the online simulators. The three applets are:

Color Saturation Arithmetic Simulator – This applet shows the effects of saturated and unsaturated arithmetic on color manipulation.

Conditional Selection Simulator – This applet simulates conditional selection of background color and the concept of masking an image.

Image Filtering Simulator – Four image filters can be simulated using this applet.

The latter two applets which perform image processing also show the hexadecimal numerical values of the image pixels and the result of the execution of the algorithm.

B. Multimedia Architecture Simulator

The Multimedia Architecture Simulator is a full-fledged, web-based simulator which simulates SIMD instructions. It has the capability to edit, execute, and debug assembly programs. The assembly instructions include multimedia extension SIMD instructions that can operate on multiple data at the same time. The data types supported by the SIMD instructions include byte, word, double word, quad word integer and single precision floating-point numbers.

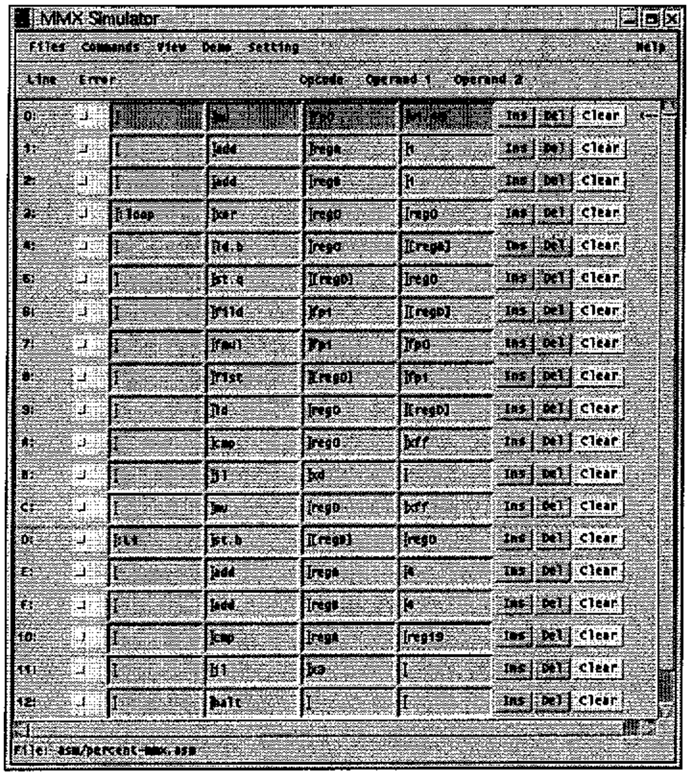
The simulator also includes a tutorial, an operation manual, and a reference manual for the instruction set available on the website. The demonstration and sample programs provide guidance on how to use the simulator and its components, and how to write a program in its assembly code.

The purpose of the simulator is to evaluate the effect of multimedia SIMD instructions such as *pack/unpack*, and *pcmpeq*. To clarify the impact of these instructions, two programs can be executed in parallel. The performance of the programs can be compared in terms of their total execution cycles.

The simulator can load program files from the server machine or from other machines on the network. The simulator can also edit a program online, and execute it over given input images. The simulator can load any images available online as well as images on the server.

The simulator has its own macro language which enables execution of commands in batch mode.

Figure 1 shows the main code input screen of the Multimedia Architecture Simulator. Using this screen the user prepares an image processing program, debugs it if necessary and then executes it. The code shown in Figure 1 increases the green pixel value by 46 percent.



```
0:          [illegible]  fp0     [illegible]
1:          add   rega    h
2:          add   reg8    h
3:  :loop   xor   reg0    reg0
4:          ld.b  reg0    [rega]
5:          st.q  [reg0]  reg0
6:          fild  fp1     [reg0]
7:          fmul  fp1     fp0
8:          fist  [reg0]  fp1
9:          ld    reg0    [reg0]
A:          cmp   reg0    0xff
B:          jl    [illegible]
C:          mov   reg0    0xff
D:  [illegible]  st.b  [rega]  reg0
E:          add   rega    4
F:          add   reg8    4
10:         cmp   rega    reg19
11:         jl    [illegible]
12:         halt
File: asm/percent-mmx.asm
```

Fig. 1. The simulator main code input screen.

The leftmost column shows the labels. In the sample code, the *:loop* label is used. The second column includes the opcodes, and the next two columns are operands. Labels are compiled in run-time.

The checkbox on the left of the label column sets breakpoints. The line with the breakpoint will be highlighted.

At the bottom of the screen, the status line indicates the file name of the code being executed and the current execution status.

The code can be directly edited on the screen. Each line is checked for valid syntax (type check) after its entry. Illegal entries are highlighted in red. The opcodes and its operand types are specified in an external file. The **Del** button will delete the whole line. **Ins** will insert an empty line. The **Clear** button will reset the entire line.

Table I lists the remaining six windows which the simulator uses, and indicates their function. The

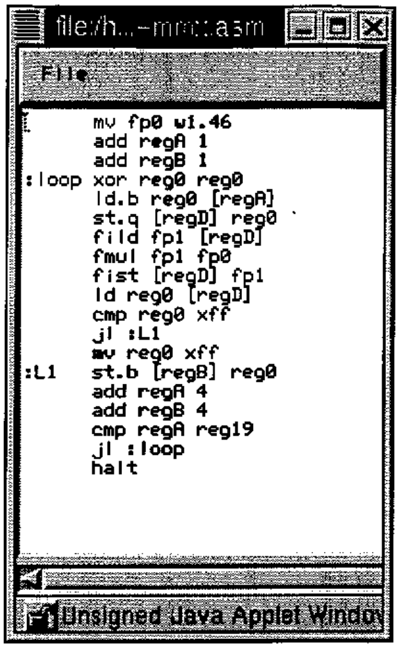
TABLE I
THE SIMULATOR'S WINDOWS.

Window	Function
Code	Displays the assembly code in plain text form for cut & paste
Debug	Displays the contents of the register for debugging
Pseudo-C	Displays the assembly code in pseudo-C format
EmuFrame	Displays the original and processed images
Memory Monitor	Displays the memory contents in hexadecimal figures
Wave	Displays the color component value of the image in a graph

Code window allows copying and editing of an assembly code. One of the difficulties we had with the web-based simulator is the ability to store programs locally for future use. The simulator which is located on a server machine does not have access to the local files unless the necessary permission is given on the local machine.

To circumvent this problem, we implemented the code window to allow for the cut and paste of a code entered online. Figure 2 shows the code which was copied from the main simulator screen. After copying the code to the code window it can be further edited there and then loaded back into the simulator screen for debugging and execution. A local copy of the simulator can be made and run locally. In such a case, the simulator has direct access to the local files and the processed images can also be stored locally.

To assist the users in debugging their programs we developed the debug window shown in Figure 3. It displays the contents of all the registers and status words. Furthermore, any altered value of registers are highlighted in red. The displayed registers in-



```
      mv fp0 w1.46
      add regA 1
      add regB 1
:loop xor reg0 reg0
      ld.b reg0 [regA]
      st.q [regD] reg0
      fild fp1 [regD]
      fmul fp1 fp0
      fist [regD] fp1
      ld reg0 [regD]
      cmp reg0 xff
      jl :L1
      mv reg0 xff
:L1   st.b [regB] reg0
      add regA 4
      add regB 4
      cmp regA reg19
      jl :loop
      halt
```

Fig. 2. The Code window.

clude 19 general-purpose registers, eight floating-point registers, and eight multimedia registers which are mapped onto the floating-point registers [7]. The program counter shows the current line of the code, the status word shows the result of a **cmp** operation.

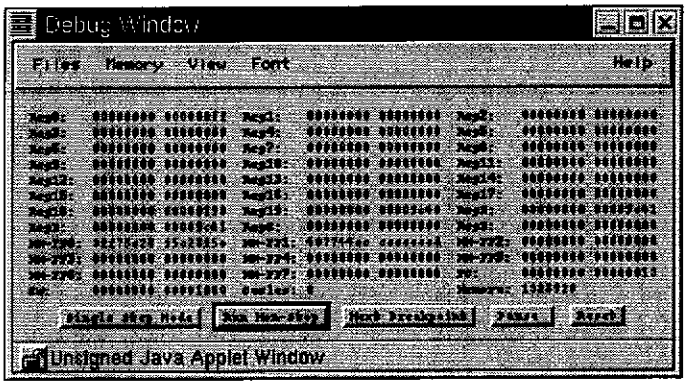


Fig. 3. The Debug window.

In addition, the Debug window controls the breakpoint mode execution as well as single step execution of the code. In the debug-mode execution of the program, the current line is highlighted and an arrow on the left side of the column is shown. The program can be executed in one out of four modes (see Table II). The **Single Step** executes one instruction at a time. The **Breakpoint** mode stops execution at user-specified breakpoints whereas **Non-Stop** executes the program regardless of the setting of breakpoints. The condition for breakpoints can be set from the simulator screen menu. The **Pause/Reset** button suspends/resumes the execution of the program. The **Reset** button clears the result of the previous execution.

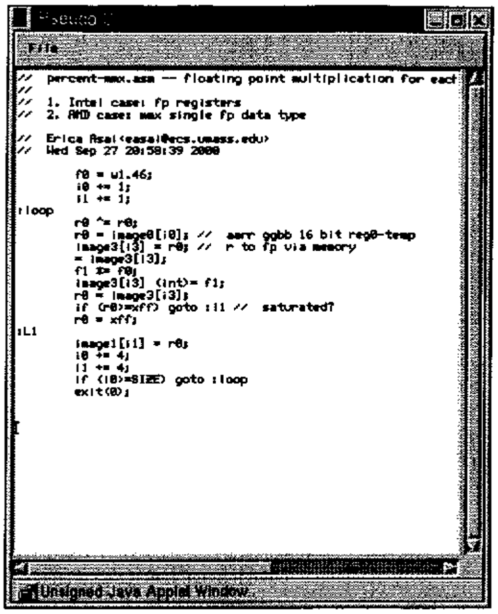
TABLE II
EXECUTION MODES.

Mode	**Description**
Single Step	Execute one instruction at a time. The current line to be executed is shown on the simulator screen indicated with an arrow on the right.
Run (Non Stop)	The code is executed without breaks.
Breakpoint	The execution is stopped at each breakpoints.
Pause	This button toggles pause and resume.
Reset	Resets the previous execution of the program.

To increase the readability of the assembly code, we have implemented the Pseudo-C window (see Figure 4) which shows the "pseudo C" equivalent of the code. In it the memory is represented as an array: image0[ ] – image3[ ] and the registers are represented as variables: r0 – r19 for general-purpose registers, m0 – m7 for multimedia registers, and f0 – f7 for floating-point registers.

Three separate windows are used to show the results of executing the program. EmuFrame shows the original/input image and the processed image (e.g., Figure 5). The Memory Monitor window shows the hexadecimal values of the image. The Wave window shows the pixel values in a graph form.

Figure 5 shows the results of the execution of the program for median filtering. The image on the left is the original. The result of the program execution is shown on the right. The gaussian noise has been removed from the image. The image and the window



```
//  percent-max.asm -- floating point multiplication for eact
//
//  1. Intel case: fp registers
//  2. AMD case: max single fp data type

//  Erica Asai<easai@ecs.umass.edu>
//  Wed Sep 27 20:58:39 2000

        f0 = w1.46;
        i0 += 1;
        i1 += 1;
:loop
        r0 ^= r0;
        r0 = image0[i0]; //  aarr ggbb 16 bit reg0-temp
        image3[i3] = r0; //  r to fp via memory
        = image3[i3];
        f1 *= f0;
        image3[i3] (int)= f1;
        r0 = image3[i3];
        if (r0>=xff) goto :l1 //  saturated?
        r0 = xff;
:L1
        image1[i1] = r0;
        i0 += 4;
        i1 += 4;
        if (i0>=SIZE) goto :loop
        exit(0);
```

Fig. 4. The Pseudo-C window.

are resizable.

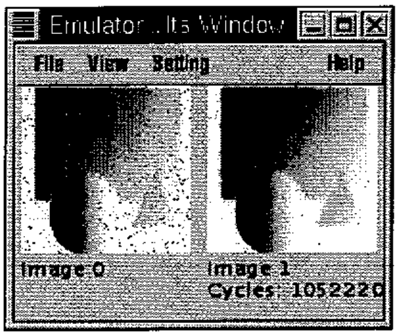


Fig. 5. EmuFrame Window: Median Filter Execution Results.

The Memory monitor window (Figure 6) shows the contents of the memory in hexadecimal. As the program is executed, the gradual alteration of the contents of the memory is displayed.

Figure 7 shows the image screen after execution of the assembly code shown in Figure 4. This example shows the comparison of two programs with and without the multimedia SIMD instructions, which have been executed in parallel [7].

The total execution cycles are 100003 without the extension and 62503 cycles with the pack/unpack instructions. This difference in the execution cycles quantifies the advantage of SIMD instructions.

The Wave window (Figure 8) shows the wave form

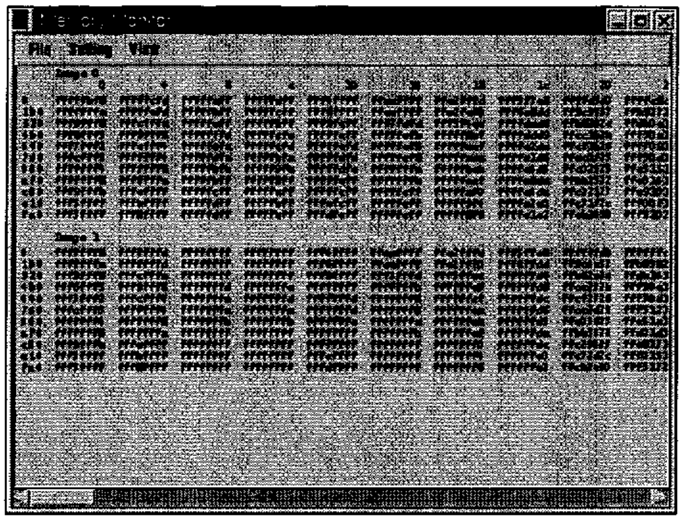


Fig. 6. Memory Monitor Window

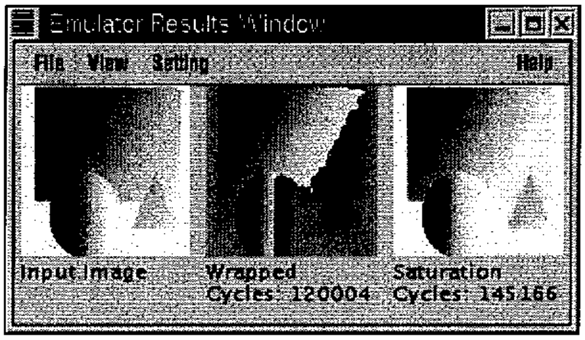


Fig. 7. Comparison of two image processing programs running in parallel: addition with and without saturation.

of the image memory. Shown in the figure are the original and the altered image pixels.

Finally, we present the results of several other image processing programs which have been prepared and executed using the simulator to further illustrate its capabilities. Figure 9 shows the results of an image smoothing program. Figure 10 depicts the output of a conditional selection program which superimposes one image on top of another image. Figure 11 shows the result of an edge-detection program.

IV. Conclusions

The Multimedia Architecture Simulator is full-fledged, web-based simulator which simulates multimedia SIMD instructions. It has the capability to edit, execute, and debug assembly programs. The assembly instructions include multimedia extension instructions that can operate on multiple data at the

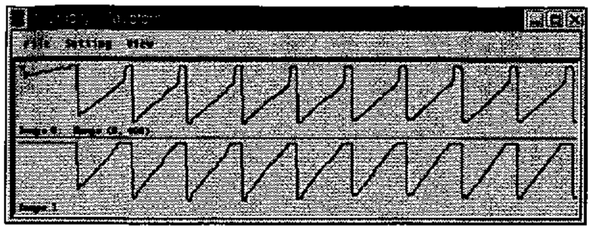


Fig. 8. The Wave window.

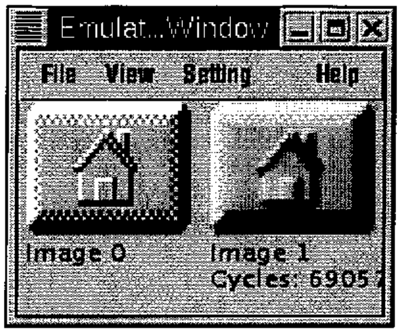


Fig. 9. Smoothing filtering execution results.

same time.

We have shown that the integration of the on-line multimedia course materials and the web-based Multimedia Architecture Simulator provides an effective course environment.

Acknowledgments

This project was funded in part by the National Science Foundation under grant EIA-9812589 and by the University of Massachusetts, Amherst. We would like to thank Andy Wolan for the design of the GUI, and Professor Wayne Burleson and other members of the VLSI Lab for help with the multimedia materials.

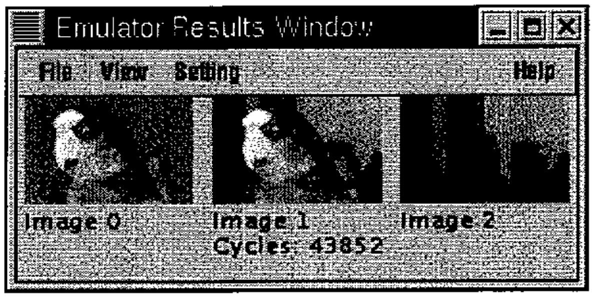


Fig. 10. Conditional selection execution results.

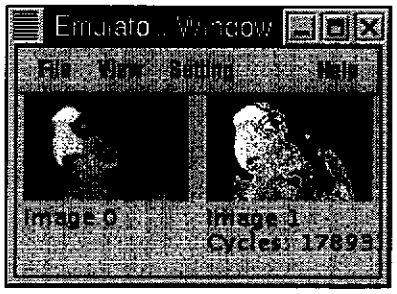


Fig. 11. Edge detection execution results.

References

[1] A. Schapira, K. De Vries, and C. Pedregal-Martin, "Manic: An open-source system to create and deliver courses over the internet," in *Symposium on Applications and the Internet*. 2001, IEEE Computer Society Press.

[2] D. A. Hennesy and J. L. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., 2nd edition, 1996.

[3] S. Oberman, G. Favor, and F. Weber, "Amd 3dnow! technology: Architecture and implementations," *IEEE Micro*, 1999.

[4] L. Kohn et al., "The visual instruction set(vis) in ultrasparc," *IEEE*, 1995.

[5] J. Gosling and H. McGilton, "The java language environment," http://java.sun.com/docs/white/langenv/, May 1996.

[6] E. Asai, "Multimedia architecture simulator manual," http://vsp2.ecs.umass.edu/dvd/doc/module4/Simulator/manual/simulator_tf.html, 2001.

[7] A. Peleg, S. Wilkie, and U. Weiser, "Intel mmx for multimedia pcs," *Communications of the ACM*, 1997.