

SOURCE CODE TRANSFORMATION USING ATOMIUM

At this moment we have assembled all the information to go into detailed design of the JPEG encoder:

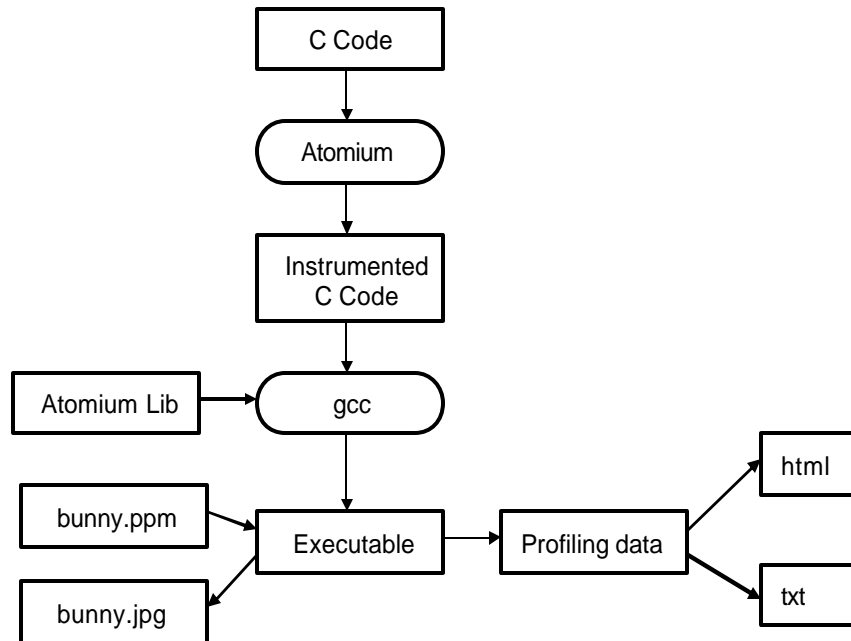
1. We have studied a sequential specification to identify the different parts (homework 2)
2. We have written a parallel system model to see how these parts work together (homework 3)
3. We have done data type refinement to fixed point or 16-bit C short (homework 4)

At this moment we can take the JPEG encoder and write a C program optimized to the implementation platform for it. Three of the four design environments that we use can read in such a C description: There is a C compiler for TI C54 and Analog Devices Blackfin. Also ART Designer supports a subset of C.

In a first step, we will transform the generic dataflow model to a program that is optimized in terms of memory usage. We will use the ATOMIUM tool to guide us in translation of the dataflow model into C code. ATOMIUM is a tool that allows you to find out the exact lifetimes, the number of writes and the number of reads of any variable of your C program. This knowledge allows you to rewrite the C program in such a way that memory use is optimized for the target architecture you are working with. The outcome of this homework is

- a) the memory architecture and allocation for the JPEG Encoder
- a) If you use TI, ADI or ART, the initial implementation spec for the encoder. If you target HandleC, you will be able to use the results of (a) but you will have to backannotate them into the HandleC description.

The design flow with ATOMIUM is as follows. Your C code is read in by ATOMIUM and *instrumented*. This instrumentation performs insertion of special function calls at places in your code where you access background variables (arrays). Next, you can compile and link your program with the ATOMIUM library. The resulting executable has the same functionality as before (that is, it can read in PPM files and compress them into JPG files), but in addition it also collects profiling data when you run the program. This data can be post-processed and converted by ATOMIUM utilities into `html` or `txt` files. An example output on a web browser is shown next.



This output shows the example of the dataflow model (jpeg_df) while processing the image bunny.ppm. For example, the variable row_res is read 16384 times and written 16384 times. In total, 32800 variable lifetimes are carried by this array. You can also see there are hyperlinks that, when clicked, take you straight to the source code that defines the variable accesses.

List of arrays sorted by number of accesses

Total accesses:

Accesses	Writes	Reads
377,646	158,988	218,658

Accesses per array:

Accesses	Writes	Reads	Elements	Lives	Name	Location	Type	Dimensions	Nature
182,914	91,457	91,457	4,096	35	data	[ffo.c:30]	int	[]	dynamic
49,152	16,384	32,768	8	32,800	row	[dct.c:103]	int	[8]	static
32,768	16,384	16,384	8	32,800	row_res	[dct.c:104]	int	[8]	static
16,384	8,192	8,192	64	16,400	tmp	[b4.c:34]	short	[64]	static
16,384	8,192	8,192	64	128	tmp	[dct.c:38]	short	[64]	static
8,384	64	8,320	64	1	zigzag_order	[b4.c:22]	int	[64]	static
8,209	16	8,193	8	1	linebufR	[b1.c:25]	int *	[8]	static
8,209	16	8,193	8	1	linebufG	[b1.c:28]	int *	[8]	static

THE ASSIGNMENT

Starting from the dataflow model, rework the source code into a C program that can be compiled without too much modifications by ART/ C54/ BlackFin. While writing C code, use Atomium to get feedback on the number of writes/ reads on arrays, and work to decrease that number. Specifically, you have to remove the FIFO blocks in the model and replace them with dedicated buffers. Overall you should try to optimize for performance (primary goal) while minimizing memory consumption (secondary goal). While making optimizations, take the memory architecture of your target platform *explicitly* into account. That is,

- For TI, take the bus architecture and on-board memory architecture into account. TI has a modified Harvard architecture (check the functional block diagram on <http://www-s.ti.com/sc/psheets/spru307a/spru307a.pdf>).
- For Blackfin, consider the memory architecture block diagram. Is there a fundamental difference with the memory architecture of the C54? (page 6-9 on <http://www.analog.com/technology/dsp/products/ads/blackfin/index.html>)
- For ART Designer, consider the architecture on slides 17 and 22 from the presentation of Doug Johnson (<http://www.ee.ucla.edu/~ingrid/Courses/ee201a/lectures/ArtUCLA2002ClassL6.pdf>). How many RAM blocks do you need? How many ROM blocks?
- For HandleC/FPGA, consider the modeling constructs of HandleC (ram, rom, wom, mpram, ...). Check also the Virtex-II datasheet on http://www.xilinx.com/xlnx/xil_prodcart_product.jsp?title=v2_resources. How many RAM blocks do you need? How many ROM blocks?

Most important, our design target is *to maximize performance*, even if this costs extra memory. The tables produced by ATOMIUM will be most useful in identifying bottlenecks in your code.

GETTING STARTED

On the project homepage, you will find a package that contains a version of `jpeg_df` that can be processed by ATOMIUM (Some minor adjustments are needed, such as substituting system include files with ATOMIUM-compatible ones). In order to run ATOMIUM, you will need to work on HPUX10 (the same machines as you have used for ART designer). Download the package and type `/usr/local/bin/make` while you are logged in into an HP machine (`hplab01.ee.ucla.edu` and `up`). You will see ATOMIUM processing the C files, running a compilation, running the executable and finally post-processing the profile data to return access statistics.

Now, you have to start reworking the C code in order to remove the `fifo`'s and make a dedicated memory architecture. The easiest way is probably to work incrementally, starting to make small modifications to the code and running ATOMIUM as you go along. In the end, you should have an optimized program that runs without `fifo`'s and for which you can also precisely estimate which variable goes into which memory. The target platform that you should consider is indicated on the project web page.

WHAT TO TURN IN

Due Date: 5/23 at 12:00 noon at Letty's desk (7440 Boelter Hall)

What: 1 sheet stating the following

- Your name.
- A list of the memory requirements for the JPEG encoder on your target platform based on the profile feedback that is given by ATOMIUM *and* your knowledge of the memory architecture of the target platform. Here is an example of how you could formulate your answer. You only need to give results for your own target platform.

Var	Type	Accesses	Storage
<code>row_res</code>	<code>Int[8]</code>	32768	<i>For Blackfin you could write:</i> Data memory <i>For Xilinx FPGA you could write:</i> BlockRAM: <i>For ART you could write:</i> RAM 1 (assuming you have 4 RAMs allocated).
..

- Include on this sheet a link where we can find your final C code for this project phase.
- IMPORTANT NOTES
 - Remember that you will have to carry the code that you create now to the rest of the project (into the implementation). Thus, a quick job now will result in more work later (while the reverse can be said for a careful job).
 - You can assume that images of only 64*64 are needed. You can allocate all array sizes at compile time.
 - To work with ATOMIUM, you need to log into one of `hplab01` to `hplab17` (a machine with HPUX10). You also need to set the following environment variable before running ATOMIUM:

```
setenv LM_LICENSE_FILE /w/ee.07/ingrid/schaum/atomium-1.2/atomium.lic
```