The Linley Group

MICROPROCESSOR report

Insightful Analysis of Processor Technology

SILEXICA'S HARDWARE/SOFTWARE CO-DESIGN

System- and Software-Analysis Tools Exploit Multilevel Parallelism

By Tom R. Halfhill (October 2, 2017)

Had Rudyard Kipling lived in Silicon Valley instead of India, he might have written, "Hardware is hardware and software is software, and never the twain shall meet." Project teams segregated into hardware engineers and software engineers often reflect their schooling and the industry's traditional design methodology. But as the free ride of Moore's Law ends, a comprehensive approach to system design is mandatory to achieve performance targets.

That's why German startup Silexica is pursuing two difficult goals: optimizing sequential code for parallel execution and finding the optimal hardware to run the software. Either pursuit alone would be challenging enough for most companies, but Silexica views them as inextricably linked. Parallelism has limited value if either the hardware or the software can't fully exploit it. Consequently, the company's SLX technology enables high-level systemwide analysis of both domains.

SLX tools are most effective at the dawn of a design project, when both the hardware and software are malleable. Tweaking the hardware design for better parallelism can yield big gains in software performance, and vice versa. When the hardware design is already frozen or even deployed in the field, the software must adapt to it, but significant gains are still possible.

Silexica isn't trying to replace existing development tools for hardware and software. Nor do its tools replace today's System-C modeling or cycle-accurate hardware simulators. Instead, SLX is a suite of high-level analytical tools that work above traditional EDA tools, compilers, simulators, and debuggers. By raising the abstraction level, the company can improve design efficiency—especially if the architects have many alternatives to evaluate and each iteration is time consuming, as it usually is with low-level EDA tools.

SLX has been shipping since mid-2014 and has attracted some notable customers, such as Huawei Wireless, which has used it to evaluate the power efficiency of dynamic voltage and frequency scaling (DVFS) in a multicore SoC. Other disclosed customers are Denso, Fujitsu, and MBDA (a European missile manufacturer).

Crucial Element: The System Model

Silexica was founded in 2014 as a spinoff from the Institute for Communication Technologies and Embedded Systems at Aachen University in Germany. Now based in Cologne, the company has closed two funding rounds, has opened

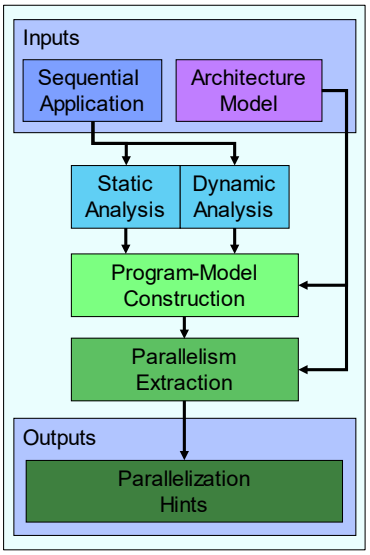

Figure 1. System analysis using Silexica SLX tools. Unlike typical code profilers, SLX requires a performance-accurate model of the target hardware to perform a thorough system analysis.

Price and Availability

Silexica's SLX tools are available now; the company withholds pricing. To find case studies and white papers online, go to *www.silexica.com/resources*. The first cloud version of SLX is available at *www.slx.cloud*.

offices in three countries, and employs about 50 people. Initially, it's focusing on four markets: automotive, aerospace/military, embedded vision, and wireless baseband. The ideal (but still rare) user is a system designer who balances both the hardware and software aspects of a project.

Figure 1 shows a conceptual view of SLX in action. As is often the case, this example assumes that some application software already exists but is written for sequential execution. Although other vendors offer profiling tools that can find hot spots, SLX tries to go further by statically analyzing the source code and dynamically analyzing the executable code. Also, it applies this analysis to a high-level but performance-accurate model of the system architecture. Conventional code profilers may have little or no knowledge of the target hardware.

The performance-accurate model is a crucial element that will largely determine the feedback's value. This model isn't as fully detailed as a gate-accurate RTL model or even a cycle-accurate simulator, both of which are more difficult to construct and run slowly on even the fastest engineering workstations. But neither is the model so abstract that the profile is generic.

In some respects, the SLX model resembles a cycle-accurate simulator—for example, it knows the clock-cycle latency of each CPU instruction. It also knows the CPU's clock speed and power consumption. In most respects, however, the model is more like a high-level abstraction. Silexica says the model runs quickly, usually producing results in minutes or a few hours, depending on the complexity of the hardware and software.

Customers can build the system-architecture model themselves using the SLX tools or employ Silexica's consulting services to build it (and even run it) for them. Constructing the model requires intimate knowledge of all the relevant system components and intellectual property (IP), including CPU IP, peripheral IP, and software IP. Some customers are uncomfortable sharing their IP or even licensed IP with an outside party, so they build the model themselves. Others trust Silexica to handle this step.

Building the System Model

Either way, the model must reflect the target hardware with reasonable accuracy. If the system is a single-chip SoC with multiple CPU and DSP cores and various accelerators, the model must reflect the performance of all the hardware that's in the software's critical path. If the system is an automobile with 100 processors and the customer needs a systemwide performance profile, the model must mirror the entire system. The SLX tools can extract some performance data from the IP and other sources automatically, but some data requires manual entry from technical documentation or from experience.

Silexica has built several system models to prove its technology and to serve customers. Among them are models of the ARM Cortex-A7, A9, A15, and A53 cores, with Cortex-A72 in progress; a PowerPC core; Intel's Atom processor; Texas Instruments' C65x and C66x DSPs; NXP's StarCore and Analog Devices' Blackfin DSPs; Infineon's TriCore heterogeneous automotive processors; and the Nios II and MicroBlaze soft CPUs for Intel and Xilinx FPGAs, respectively. Silexica claims it can develop the first useful version of a new model that's about 80% accurate in two or three weeks, depending on the target hardware's complexity.

One difficult project was the Intel Atom. The x86 architecture is complex, and its instruction reordering is a particular challenge. According to Silexica, this model took a month to build and runs standard benchmark tests that score within 20% of the actual hardware.

Also a challenge was the TI DSP architecture, another complex design. Building this model took about three months, and Silexica says it's 90–95% accurate. The company doesn't promise that all models will achieve the same results but considers 80% accuracy sufficient for making important design decisions at the beginning of a project.

This aspect of Silexica's technology is hard to evaluate and perhaps the most difficult for customers to achieve. The better the model, the better the analysis—but better models are harder to build and calibrate for accurate performance. Before committing to SLX, potential customers should look closely at what's required to build their system model and obtain sufficient accuracy.

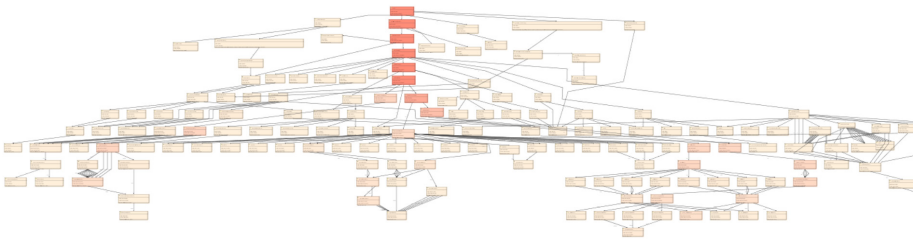

Figure 2. SLX call graph. This tree diagram shows which functions are calling other functions and which are potential bottlenecks. (Source: Silexica)

Finding Four Levels of Parallelism

As Figure 2 shows, SLX generates some feedback that resembles the output of other code profilers. This example is a call graph that shows which program functions are calling other functions and how often they're called. Darker coloring indicates "hot spots"—frequently called functions that may benefit the most from

optimization. A frequently called function isn't necessarily a bottleneck if it consumes only a few clock cycles, however. Optimizing it to save a cycle or two won't make much difference in overall performance. Only a model that knows the clock-cycle latency of each routine can measure the benefit of optimizing this code.

SLX analyzes the source code to find opportunities for parallelism at four levels: data, task, pipeline, and offload. Data-level parallelism typically employs single-instruction, multiple-data (SIMD) operations and is usually limited by data dependencies—that is, when a particular operation can't proceed until it receives the result of another operation. SLX can suggest alternative ways of refactoring the code to minimize such dependencies, perhaps by duplicating a common variable that both operations need.

To exploit task-level parallelism, SLX identifies tasks that programmers can assign to nondependent threads or processes. To exploit pipeline-level parallelism, it identifies tasks that are divisible into sequential pipeline stages (not to be confused with a CPU's low-level instruction pipelining). Streaming-data applications are prime candidates. The operations that a wireless base station performs during baseband processing are an example; they must execute in a particular sequence and meet the strict timing requirements dictated by international standards. SLX can suggest the most efficient division of these operations into steps that match the hardware's capabilities.

Offload-level parallelism uses various coprocessors and accelerators (such as GPUs and task-specific engines) to perform frequent operations that would burden a CPU or DSP core. To find these optimizations, SLX again requires intimate knowledge of the target hardware—not only which accelerators (if any) are available, but also their performance characteristics. Some offloads can run asynchronously with the host CPU or DSP core, whereas others require more supervision or share resources (such as registers or caches) with the host. The hardware model must expose these characteristics for the Silexica technology to do its job.

The SLX output is quite specific. Figure 3 shows an example of code highlighting that uses different colors to indicate nondependent code blocks eligible for parallel processing. Additional highlighting (not shown) identifies dependent code blocks that programmers can refactor to be nondependent. In these cases, SLX displays example code that programmers can cut and paste into their text editor and compile with their usual tools.

Although SLX can't inspect compiled binary code directly, it can analyze source code that calls closed-source libraries. When the program runs on the target system, it generates the platform-specific traces that SLX needs for analysis. When all of the source code is available, SLX provides instrumentation at the intermediate-representation (IR) level, static- and trace-based dynamic analysis, graph transformations, feature extraction, pattern matching, and performance estimates.

Optimizing for Different Goals

SLX can recommend alternative optimizations that depend on the developer's priorities. Is the main goal to maximize throughput, minimize power consumption, conserve memory, streamline I/O, or use less network bandwidth? Typically, developers must make tradeoffs to achieve the best efficiency in the dimension most important to them.

Figure 4 shows the results of one analysis that optimizes for DRAM utilization in a memory-bound system. In this example, the most memory-efficient alternative executes the task in 3.20 seconds. Optimizing for performance would cut the execution time to 2.23 seconds but would use more memory; optimizing for power consumption would increase the execution time to 6.06 seconds while also using more memory.

When targeting an existing system or hardware design that's already frozen, developers have little choice but to optimize the software. Ideally, they would employ SLX when starting a new project to optimize both the hardware and software in concert. The Silexica tools work best when customers take a holistic approach to system design instead of treating the hardware and software as separate domains.

In one case, SLX analyzed a critical task running on eight hypothetical SoCs that integrate 1 to 14 DSP cores. Execution times fell greatly when moving from a single-core chip to a quad-core chip, but then flattened. The 10-, 12-, and 14-core designs achieved almost identical results, indicating that a 10-core chip would be the most economical if the 2.51-millisecond performance were sufficient.

```
static int ParseFrame(VP8Decoder * const dec, VP8Io * io)
{
    int ret_val;

    for (dec->mb_y_ = 0; dec->mb_y_ < dec->br_mb_y_; ++dec->mb_y_)
    {
        // Parse bitstream for this row.
        VP8BitReader *const token_br = &dec->parts_[dec->mb_y_ & (dec->num_parts_ - 1)];
 4%     if (!VP8ParseIntraModeRow(&dec->br_, dec))
        {
            ret_val = VP8SetError(dec, VP8_STATUS_NOT_ENOUGH_DATA, "Premature end-of-partition0 encountered.");
        }
        for (; dec->mb_x_ < dec->mb_w_; ++dec->mb_x_)
        {
27%         if (!VP8DecodeMB(dec, token_br))
            {
                ret_val = VP8SetError(dec, VP8_STATUS_NOT_ENOUGH_DATA, "Premature end-of-file encountered.");
            }
        }
        VP8InitScanline(dec);   // Prepare for next scanline

        // Reconstruct, filter and emit the row.
 67%    if (!VP8ProcessRow(dec, io))
        {
            ret_val = VP8SetError(dec, VP8_STATUS_USER_ABORT, "Output aborted.");
        }
    }
    if (dec->mt_method_ > 0)
    {
        if (!WebPGetWorkerInterface()->Sync(&dec->worker_))
            ret_val = 0;
```

Figure 3. SLX code highlighting. Silexica's tools identify and highlight the source code that programmers can refactor to increase parallelism. (Source: Silexica)

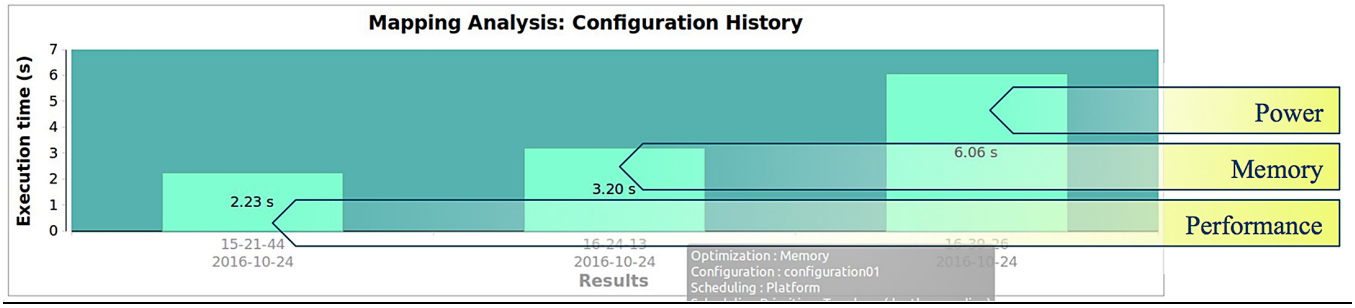

Figure 4. SLX software-design options. After these test runs, SLX displays the execution time (3.20 seconds in this example) when the task uses the least amount of memory. Other test runs optimize for power or performance. The analysis estimates DRAM utilization (948.08KB), energy consumption (34.65 kilojoules), peak power (17.63W), and average power (10.83W). (Source: Silexica)

Avoiding Wrong Turns

Developers can also use SLX to avoid making impractical optimizations—assuming the tools have enough knowledge of the system parameters. For instance, if optimizing a program for parallel processing requires a heavier data stream than the DRAM or I/O interfaces allow, perhaps it's better to leave the code alone until the hardware isn't memory or I/O bound. Likewise, if SLX determines that the program is compute bound, the hardware engineers needn't waste time designing a faster DRAM bus or PCI Express controller.

Silexica says its technology can theoretically work with any hardware, but only some hardware has been modeled. Currently, it can analyze system designs based on some CPUs, DSP cores, and custom accelerators; GPUs and FPGAs are on the roadmap. It also works with the existing development tools for those architectures, providing optimization hints to software compilers or RTL-synthesis compilers that accept those inputs as directives or as language constructs.

For example, the company used SLX to analyze and optimize the reference source code for the H.265 video codec. This video-compression program has more than 86,000 lines of code and defines 3,268 functions. Silexica's system model was a hypothetical SoC with six ARM Cortex-A7 cores. After four hours of analysis, SLX found that optimizing only six functions would boost overall performance by 38%.

The technology also found more than 90 obstacles to additional parallelism that, if removed, would allow the codec to run even faster. For each function or loop, SLX reports the potential local speedup (which is often major) and the global program speedup (which may be minor). These statistics enable developers to decide which opportunities merit further attention.

A Unique Offering

Silexica makes a reasonable sales pitch. Development tools have been growing more sophisticated, abstract, and automated since assembly language replaced Eniac's plugboard. Optimizing both the hardware and software of a system design using one tool suite is a logical step, because the mutual dependence of those domains keeps growing. Today, it's common for chip vendors and system OEMs to employ as many software engineers as hardware engineers.

Optimizing the software is probably easier than optimizing the hardware. SLX does the kind of code analysis and hinting that other supplemental tools offer. Wisely, the company isn't trying to replace existing development tools or platforms such as Gnu's GCC or Nvidia's Cuda.

On the hardware side, SLX does the kind of analysis that traditional EDA tools, simulators, and System-C models can handle, but Silexica isn't trying to replace them, either. Instead, it says SLX offers higher abstraction and faster design exploration. Where the technology departs from conventional tools is by integrating the two domains. Deeper insight enables SLX to optimize the system holistically. Therefore, its success depends largely on the thoroughness and accuracy of the underlying system model.

Constructing that model may be the biggest challenge for customers, particularly if their design is a complex one with heterogeneous processors and numerous other components, as is often the case. Those components may be an eclectic mix of in-house IP and licensed IP from various sources, and they may not expose all their performance parameters. Building the model may require some exploration, educated guesswork, improvisation, and manual labor. Customers that are looking for a turnkey solution and are comfortable sharing their IP with Silexica may be better off engaging the company's consultancy services for that phase of the project—or even for the whole analysis.

Silexica recently announced a cloud-based version of SLX that may address some of these concerns and enable customers to employ the tools without installing the software on their local workstations. A trial version allows customers to try before they buy, using some cloud-based models. Certainly, SLX is a product that requires careful evaluation before a commitment. Although the evaluation requires time and effort, a successful engagement may deliver big advantages in performance, development costs, and the final product's bill of materials. ♦