How Java Software Solutions Outperform Hardware Accelerators

MIPS Technologies, Inc.
April 2005

Java is a programming language which has one big advantage and one big disadvantage: the big advantage is 'write once, run anywhere'. The big disadvantage is performance. The mechanism that makes it portable also makes it difficult to execute.

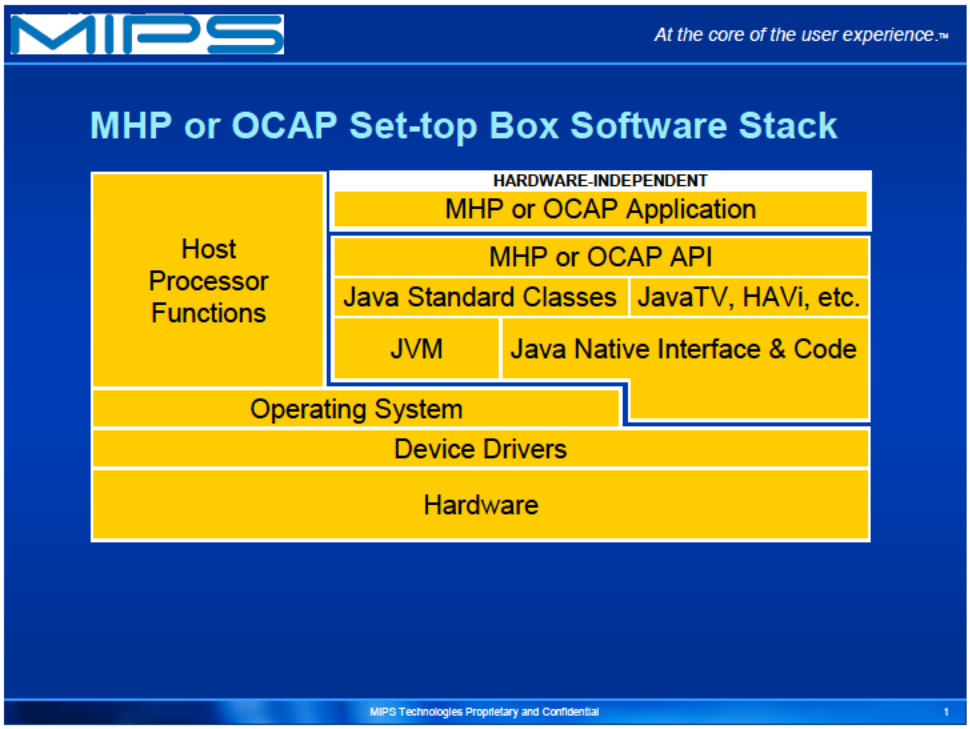
Java for a long time has been in use only in PCs, servers and so forth, but it is making its way into the embedded market because of interactive television and mobile phones.

Why Java?

Interactive television historically has been a very fragmented market. Different satellite and cable companies, different software vendors, and different manufacturers of set-top boxes had different ways of implementing it, which meant that if a given cable company wanted to write an application, they had to write it several different times and then deploy each to the box for which it was intended. It was a high investment, low return endeavor.

In order to fix that problem, two standards were introduced worldwide. One is called MHP—multimedia home platform. It is the European standard, though it has worldwide applicability. The other is OCAP—the Open Cable Applications Platform—that is the U.S. cable standard, created by Cable Laboratories, which was formed by the cable companies to establish open standards. Both of them are similar. In fact, OCAP is largely based on MHP. Consequently, once these standards are deployed, one would be able to write an interactive TV application and not have to worry about what box it is going to run on.

Now as far as deploying them, in the OCAP world it is going to be the cable companies, which means it is going to be a very sharp ramp, because one day they are going to deploy and mandate OCAP. Every box that they ship from that point on will have OCAP in it. MHP, on the other hand, is being deployed on a voluntary basis. The broadcasters in Europe—both satellite and terrestrial—provide their boxes largely through retail outlets. Their users therefore have a choice. They can buy a box with or without MHP, and MHP boxes are more expensive. In any case, MHP is growing, but it is a gradual ramp because of the cost difference and, for the time being, less perceived value on the customer's side. But both of them will reach multi-million units annually within the next five years.



Java Solutions

Java hardware accelerators first appeared about five years ago. They tried to fix the performance problem by adding a little processor next to the main processor to run Java. While such accelerators do improve performance and require very little memory, they are far from ideal.

For one thing, memory requirements today are no longer critical; what used to be a memory-limited system no longer is. For another, software techniques--such as dynamic compilation--that have come along far outstrip hardware accelerators in terms of performance and do not require the extra area that the hardware approach requires or the power that it therefore consumes.

To see how software solutions can outperform older hardware ones, let us look at the evolution of Java. The oldest implementation is the interpreter. Every Java command is interpreted to the equivalent microprocessor command and is run in the order in which it arrives. This is a really slow way of doing things.

Then came the JIT (just in time) compiler. Every time the Java execution runtime environment would run into a new class—classes are functional groups within the Java program—the JIT compiler would compile it right there. Once something is compiled, it runs with native commands, and it is fast. Spending a little bit of time up front can save a lot of execution time later. That did improve matters, but it still did not get the top performance, because some things that would only run once could take longer to compile than it would take to run them with the interpreter. This means you could wind up with a net loss.

With that observation came the dynamic compiler, which compiles only those things that matter and leaves the rest alone. The dynamic compiler decides whether to compile each class. It has two weapons in its arsenal: an interpreter and a JIT, and it makes an intelligent decision on a class-by-class basis whether to use the one weapon or the other. The dynamic compiler makes that decision by “profiling,” or letting the code run a few internal loops before deciding whether or not to compile that section of code. The decision may be wrong, but statistically the dynamic compiler is right much more often than not; in fact, the longer you run the code, the more likely it is to get it right.. The result is faster code execution than you can achieve by older methods—as much as 50% faster on a MHz-to-MHz basis.

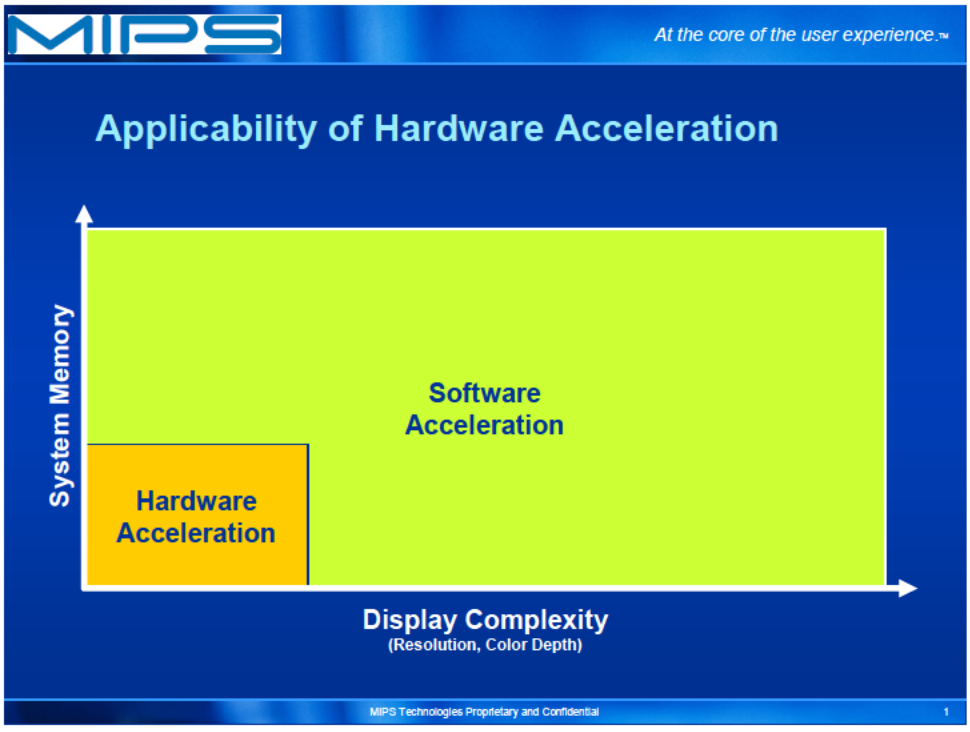
How can it be that software can run faster than hardware—in particular a specially designed "hardware accelerator"? There are a few ways to understand this:

A Java accelerator can be either small or fast. Building a Java processor is a complicated thing. It has to do with the portability that Sun built into Java. Sun made the command structure unlike any hardware architecture—in order to make it run equally badly on all architectures, if you will. You can't make hardware accelerators big, because that would overwhelm the size of the core and make it uncompetitive. So you need to make them small, and therefore slow. Such accelerators run Java faster than a simple interpreter, but in today's world that is not a meaningful comparison.

Another reason that software is faster is that the fastest way you can run code on a processor is in native form. That is, if you compile something for a given processor, that is as fast as it is going to run. A dynamic compiler gives you near native performance—it compiles all the things that matter. Hardware accelerators do not run things natively, they run them in Java.

Java puts up a lot of roadblocks to good performance. It disallows registers; everything happens on a stack. Push values onto the stack, pop them from the stack. It also does not allow pointers, which C allows. If you compute a value for one class and you want to make that value available to another class, in C you just pass a pointer to the memory location where that value is stored. Java disallows pointers for the sake of robustness, since misused pointers can cause memory leaks. But the tradeoff is that you have to move data around. To pass a value, you have to copy it from one memory address to another memory address, even though you are not changing it in any way. That is a lot slower than just passing a pointer to the original memory address where the data resides.

All those things hardware accelerators cannot address. Consequently, while hardware accelerators have marched in place for 4-5 years, software solutions have improved by leaps and bounds, now far surpassing them in performance.



The Memory Non-Issue

It used to be that cell phones were very tight on memory. They could not afford any compilation resources, which would just deplete the amount of available memory. At the time they were introduced (1999), hardware accelerators made sense.

No longer. Today, even low-end phones typically contain 2-3 Mbytes of memory. The Esmertec dynamic compiler running on the MIPS architecture requires just 235 Kbytes—a tiny fraction of the available memory. So that design limitation no longer exists. And that opens the door to all of these wonderful software solutions.

One final point: modern cell phones have larger, now almost entirely color screens, with highly graphical applications. The largest problem for the cell phone microcontroller is accelerating graphics, not Java. If you have a simple screen and very little memory, then you need a hardware accelerator. Under any other scenario, a software solution is far superior.

Summary

Java is quickly becoming a requirement in many consumer devices. While Java provides a universal platform, it also poses significant performance challenges. Hardware accelerators have only partially alleviated these performance problems. In contrast, software-only dynamic compilers not only provide upgradeability but far exceed the performance of hardware accelerators. To wit, the Esmertec Jbed CDC Dynamic Adaptive Compiler running on the MIPS architecture provides the industry's highest Java performance.

Peak Java performance is just one aspect of MIPS Technologies' leadership in the set-top box and digital television world. In fact, MIPS Technologies has become an industry standard thanks to the unrivaled software ecosystem available on the MIPS architecture. This ecosystem allows manufacturers to create products to address any market segment, geography, and device type. In addition, MIPS Technologies continues to provide the highest performance licensable CPU cores, ensuring its customers maximum flexibility in system design.