Software for the n-core Generation

1060® Research white paper series:

Software for the n-core Generation

Version: 1.0.1

June 8, 2007

What's the problem?

For forty-plus years the IT industry has made major advances, dramatically improving much of the technology stack. Operating systems work above the hardware level to present to software developers a higher level abstraction of the physical resources (compute threads, memory, storage, connectivity, etc.), making it dramatically simpler and faster to utilize the computing platform. Current day efforts in virtualization takes this to a new level as whole machines can be seen as virtual resources.

Software has also made steady progress. Assembly language gave way to compiled languages. Modular and then object-oriented languages provided the ability to build abstractions within software. The pattern movement provided an abstract way of naming and describing the relationships between objects. The arrival of virtual machines, such as Java, that take care of memory management and translate bytecode into native machine instructions. But these improvements are not sufficient to meet the demands of enterprise information systems. Software projects are still late. In fact, a recent study shows that more than 50% of software projects are late and, worse, many are failing outright:

> "The outcomes of IT projects have been slipping, with 57% of respondents saying that fewer than half of the IT initiatives in their firms had a positive outcome."[1]

What's the problem?

When you step back and look at software you can see the problem – software is still too close to the metal. Even with software written on top of a great technology stack, it is too tied to low-level concerns, too constrained and ultimately – too brittle. Working at this low level is like building a suspension bridge with nuts, bolts, ladders and a jeweler's screwdriver instead of appropriate technology such as cranes and box girders.

As we move to new architectural models such as SOA and distributed systems, we see the impact of constrained brittle software everyday, even if we are too focused on the details to see it. For example, XML messages are passed between systems providing a very flexible medium for information exchange. But, when software developers reach into their toolkits for help processing these messages, all they find are ways to bind the messages to low level code and so they inadvertently destroy the very flexibility provided by XML. Same story with databases. Relational databases are built on a set-theoretic foundation in which one thing is clearly evident – a query operation on a table always results in a table. However, when it comes time to process this information using current language tools the binding issue raises its ugly head once again and the elegance and

1 BBC News: http://news.bbc.co.uk/1/hi/business/6720547.stm

power of the set approach is lost in a tangle of objects. Software projects are failing because business information's natural fluidity and constant change is not served by inflexible and brittle code.

And, it's going to get worse.

Why? We have another problem. Welcome to the n-core generation. We can no longer rely on pure CPU speed increases for performance gains. Because of physical limitations, chip designers are going wide – they are providing multi-core systems instead of faster single CPUs. But most of our software can't take advantage of these new chips.

What are the options?

What are our choices? It looks like there are three options.

First option - look for a magic bullet. A lot of very smart people are working to address the pain being felt by developers who are trying to make the transition from single-threaded to multi-threaded application development. Google's acquisition of PeakStream is an example of the value placed on such an approach. Granted, this may address part of the pain, but it doesn't address the problem of brittle code and it remains a small scale technology.

Second option – learn how to program with multiple threads effectively. This is not easy - witness Intel's multi-core education program devoted to developing and distributing educational materials to top universities with the goal of having graduates understand how to develop multi-threaded applications. Even with this help, multi-threaded programs are difficult to conceptualize and develop correctly and can only become more and more brittle.

Third option – think differently. Why not embrace the change and solve the issue of software flexibility properly? If operating systems and virtualization are proven for physical resources, can this work for software resources? In fact – it does.

Virtualize Software

When software is built within a logical model on top of an operating-system-like abstraction running on a micro-kernel, dramatic things happen. Developers no longer think about threads – threads are a physical level issue handled transparently by the micro-kernel. Software is no longer brittle, now software can be constructed using logically identified resources and services - objects, libraries, and other small scale details are hidden in the physical level by the micro-kernel. Software is larger scale – the amount of code required to build an application goes *down* by a factor of ten to one hundred. And performance goes *up* and stays up because a dynamic virtual-

ized system can self-tune to match changing workloads.

This sounds like the ideal solution for software development – move away from the small scale details and step up onto a new and more powerful abstraction, just as the IT industry has done throughout its history.

Will this work? Fortunately, the answer is yes. A growing number of companies are building their projects this way and repeatably find they reap the benefits of more flexible, faster systems written using significantly less code in dramatically shortened time frames. Systems scale linearly with compute cores, developers don't even think about threads. Finally, we can build our information bridges with cranes and box girders.

We call this new model *resource-oriented computing*. For more details, visit our website at http://1060research.com and read the "Introduction to Resource-Oriented Computing" white paper series. Or, get in touch with us. We would be pleased to talk about your challenges and how this new approach can save you time, money and schedule risk.

For additional information about Resource-Oriented Computing, including ROC architectural consulting services and training, please contact 1060 Research at www.1060research.com