

MASTERING THE MULTICORE CHALLENGE

*An elegant and efficient approach to
exploiting the power of parallelism*

Robin Bloor, Ph D

Sponsored by:



*Texas Multicore
Technologies®*



*An Intel®
Software Partner*



The Bloor Group

© Copyright 2010, The Bloor Group

All rights reserved. Neither this publication nor any part of it may be reproduced or transmitted or stored in any form or by any means, without either the prior written permission of the copyright holder or the issue of a license by the copyright holder. The Bloor Group is the sole copyright holder of this publication.

□ 22214 Oban Drive □ Spicewood TX 78669 □ Tel: 512-524-3689 □

Email contact: info@bloorgroup.com

www.TheVirtualCircle.com

www.BloorGroup.com



Executive Summary

Texas Multicore Technology a relatively recent startup from Austin, Texas, has introduced a new software development language and environment, named SequenceL, which is specifically targeted at writing software that runs efficiently on parallel hardware. This white paper examines the technology, describing in outline how it works, why there is a need for it and how well it performs in parallel environments.

The contents of the white paper are summarized below:

- Since 2004 CPUs vendors have been increasing the power of each new chip generation by adding more cores (processors) to the chip. They are doing this because it is no longer possible to make chips run faster by increase the CPU clock speeds - too much heat is generated through voltage loss. Nevertheless, it is still possible to add more transistors to CPUs by further miniaturization. This makes it possible to add more cores with each new generation of CPUs.
- The strategy of adding more cores to each chip will ultimately fail unless the extra computer power this creates can be effectively utilized. This uncomfortable fact has created a challenge to the IT industry, because:
 - Almost none of the applications that run the businesses of the world have been written to work in multicore environments.
 - To compound the problem, writing parallel software using existing development environments is slow, arduous and requires highly skilled programmers.
 - There are very few development environments that directly address parallel operation, and most of those that do require programmers to learn new skills.
- The SequenceL environment, offered by TMT, solves the problem in an elegant and effective manner. The key points about this environment are:
 - The programmer does not need to know anything about writing parallel software.
 - The SequenceL programming language is easy to learn and easy to both read and write. It is a completely declarative language in the sense that it only tells the computer what to do without telling it how to go about it.
 - The SequenceL compiler and library add in all the parallel and computational methods that the final executable program uses.
 - SequenceL programs are independent of the target execution environment, so SequenceL will scale without any manual effort from 2 cores to 64 or more.
- SequenceL has been tested on multiple applications in multiple environments and has proven capable of delivering parallel efficiency up to 100 percent (as illustrated by the various test results on different processors described in this paper.) This is beyond remarkable, especially if compared to existing alternatives. In practice, TMT believes the low water mark for SequenceL to be around 50 percent efficiency of parallelization. The high water mark, as the benchmarks demonstrate, is 100 percent.
- In our view SequenceL is destined to become a widely used development environment in all areas of application from scientific to business use.



The End of Moore's-Law-As-It-Was

In 2004, the chip industry hit a brick wall. It was no longer possible to increase the speed of chips by increasing the clock speed. Clock speeds had risen to about 4 gigahertz and even at that speed, they gave out far too much heat to be practical for many uses, particularly for laptops. If clock speed were raised much further, the heat output would be unmanageable even in the data center environment.

This was not the end of Moore's Law, but it diverted Moore's Law in an inconvenient way. We can see this in the graph below which charts the progress of Moore's Law over many decades as applied to the Intel x86 family of chips. The reality it illustrates is the regular doubling of cpu power every 18 months.

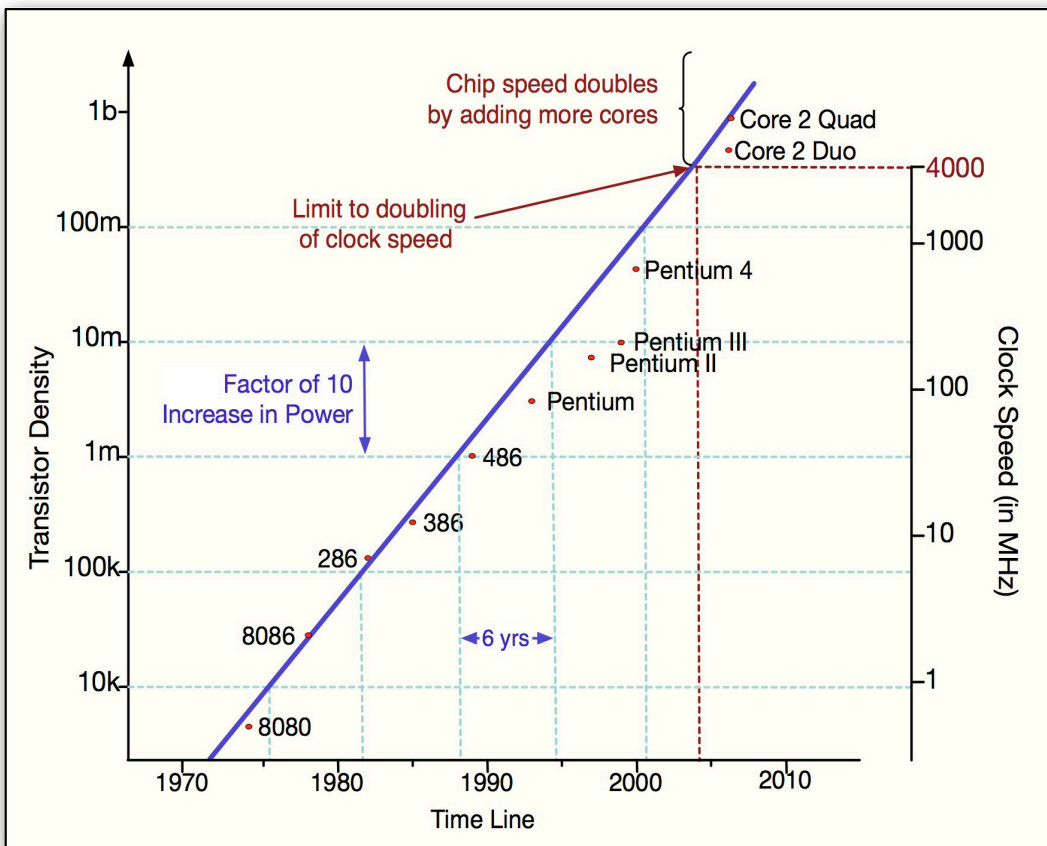


Figure 1. Moore's Law And Its Deflection

The two complementary mechanisms that the chip industry used to increase the power of chips was to miniaturize the circuits on the chip, so that more transistors could be added, and to increase the clock speed. These are shown in the graph as two vertical axes, to the left and the right. It is important to note that these two axes are logarithmic so that the "Moore's Law curve" appears to be a straight line rather than the exponential curve that it is.

When it was no longer possible to raise clock speeds, the chip manufacturers changed tactics and added more cores (or processors) to the chip, resulting in dual core, then quad core and most recently with Nehalem, 8 core chips. A two-core chip can execute twice as many machine instructions as a single-core chip, and four cores are twice as powerful as two. The



only difference now is that the chips are no longer running at a faster speed with each generation.

Why This Creates A Challenge

The chip business is driven by obsolescence. Vendors make new generations of chips that are better than the previous generation, and the previous generation of chips quickly becomes old hat. New applications appear that exploit the power of the shiny new chips and businesses and consumers eventually want to upgrade to take advantage.

The chip vendors' business model is predicated on delivering more power every 18 months or so, and for decades they've excelled at doing that. The problem the multicore chip poses is that it is difficult to write software for multicore chips. The dual core and quad core chips weren't so difficult in this respect. The operating system could divide up computer workloads reasonably efficiently between 2 cores and, although it is less effective, even between 4 cores.

But beyond 8 cores, and even to properly exploit 4 core chips, programmers need to write software that can run in a parallel manner - software that utilizes some or all of the separate cores at the same time. So the problem comes down to this:

- Very few of the applications we run operate in parallel. Pretty much everything is unsuited to the multicore environment.
- Very few programmers have any idea how to write parallel code. It's actually a rare skill and not so easy to acquire.

So the challenge is to find ways of producing software applications that run in parallel and hence can exploit the power of multicore chips. If this problem is not solved effectively, it won't just be the chip vendors who suffer. There will be a knock-on effect on the whole of the IT industry and some of the businesses that depend upon it. And that makes the SequenceL programming language far more important than it would otherwise be.

SequenceL: Its Genesis and How It Works

In the early 1990s, NASA approached Professor Daniel Cooke of Texas Tech University in Lubbock, Texas with a request that he design a language for specifying program design effectively and funding to support the activity. The language NASA wanted needed to be executable and it needed to be "Turing complete," which means being capable of specifying any process a computer can execute. NASA also wanted a language that was easy to read - no surprise given that the purpose of the language was to write specifications, rather than to write programs.

Professor Cooke produced the first version of SequenceL in 1996 and NASA was happy with it and put it to use. The irony was that there was no pressing need to write a compiler for SequenceL, because the goal was for a language that people rather than computers would read. And it's ironic because compiled SequenceL may be the solution to the multicore problem that the chip vendors have been praying for.

SequenceL might never have been compiled if Professor Cooke not had a Ph D student who needed a suitable Ph D project. It was years later, and after reviewing possible options, the professor and Ph D student that the project should be to write a compiler for SequenceL. As



both of them were interested in parallel processing, the Ph D student chose to write a compiler that made use of parallelism.

As the work proceeded, it became clear that SequenceL could be automatically parallelized and it could be done efficiently. Indeed the initial work that was done was so promising that it led to the founding of a software startup called Texas Multicore Technologies (TMT), with the goal of exploiting and further developing the parallelism that SequenceL was capable of.

After 20 years, \$10,000,000 of grants, and 50 man years of academic research, NASA still continues its commitment to SequenceL. Current efforts are directed at qualifying SequenceL in coordination with qualifying multicore chips for highly efficient parallelized on-board flight control.

SequenceL and Parallelism

SequenceL is a declarative language. A declarative language is one that tells the computer what to do, but not how to do it. There are few declarative languages; the only two that have seen much use at all are SQL (for accessing databases) and the AI language Prolog. SequenceL was deliberately designed to be declarative for the obvious reason that its purpose was to specify exactly what a program should do without the added chore of suggesting how it would be achieved.

The programming languages that most of us are familiar with are procedural language, which tell the computer both what to do and how to do it. Nearly all the commonly used programming languages are procedural; C, C++, Visual Basic, COBOL, Java, Python and so on. In those languages the programmer can easily order the computer to do something in an inefficient way, and given that programmers haven't cared much about computer performance for many years, that's often what happens.

In a declarative language, there is no specification of how to carry out a task, so it is possible to have the compiler choose efficient ways to do things. So, in a multicore environment, the compiler can choose how best to exploit all the cores.

The way that SequenceL is converted into an executable program is illustrated in the diagram. The compilation process consists of two passes. First, a compiler converts SequenceL into parallel C++ source code. (It's a hybrid compilation/interpretation process.) It carries out "large grain parallelization" producing parallel C++ code. The

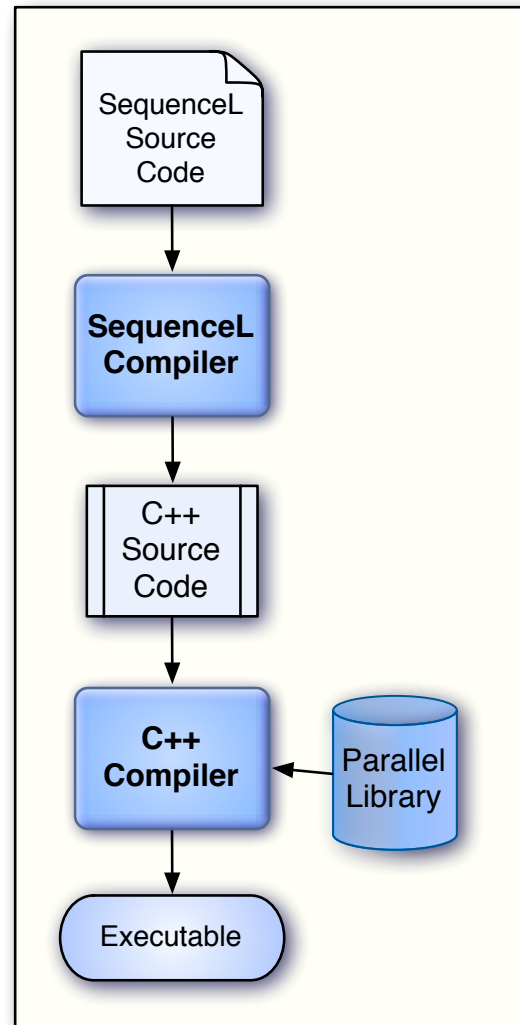


Figure 2. SequenceL in Operation



second pass is through a C++ compiler that has a special set of parallel libraries. You can think of that as a “fine grain parallelization,” augmenting the parallelization of the first pass.

The Efficiency of SequenceL

SequenceL is very new. TMT only began trading this year and it is still developing its technology. However, even at this early stage the company has little doubt that its technology delivers an unprecedented level of parallelism in the environments where it has been tested and deployed.

The Anecdotal Evidence

TMT chose to go to market directly through consultancy activity while simultaneously trying to stimulate interest in its technology amongst chip vendors and computer manufacturers. It’s initial consultancy projects involved companies that had a pressing need for efficiency in critical projects but had failed to achieve it.

One of TMT’s initial customers had a specific program within a process control system that needed to run far faster than it currently did. It normally took about 180 minutes but the program never tried to exploit parallelism at all. Parallelizing the program and running it on 16-way configuration would solve the problem if the parallelism was reasonably efficient.

So TMT rewrote the program in SequenceL. When the team compiled and tested the program it took just 2 minutes instead of 180 minutes, running 90 times faster! This was a shocking result that could not be explained just by the efficiency of SequenceL. The absolute best that the team had thought possible was to reduce the run time to 12 minutes (i.e. 15 times faster.)

The team analyzed the source code of the original program and concluded that the program was written in a naïve manner with little focus on performance. This experience exposes an interesting aspect of this technology. SequenceL doesn’t provide the programmer with as many opportunities to write code as inefficiently as a procedural language does.

In most circumstances, programmers don’t even think about application performance and haven’t for years. There are exceptions. Indeed there are even a few programmers who are trained and skilled in writing parallelized code, but not many.

With another customer, TMT was challenged to rewrite a program that had already been parallelized by such skilled parallel programmers. This constituted a much more important test of SequenceL’s capabilities. In this instance the customer was interested primarily in whether it could find a less expensive way to produce parallel applications than hand crafting them. So the customer’s technology test involved a program that was complex and needed to run in parallel. The TMT team was given a time limit of 3 weeks to reproduce the program in SequenceL.

The results of the test were that the SequenceL program that was written outperformed the hand-crafted program by 50 percent. Additionally the SequenceL program was more scalable. Hand-crafted parallel programs are sometimes hardware dependent, whereas SequenceL programs are not.

The customer chose to adopt SequenceL and is now considering how to employ it most effectively as a strategic parallel development environment.



The Benchmark Evidence

The parallelization delivered by SequenceL has been benchmarked on different processors using 5 distinct applications, all of which involve 2 dimensional or 3 dimensional contexts. The results of the tests are illustrated in Figure 3. below.

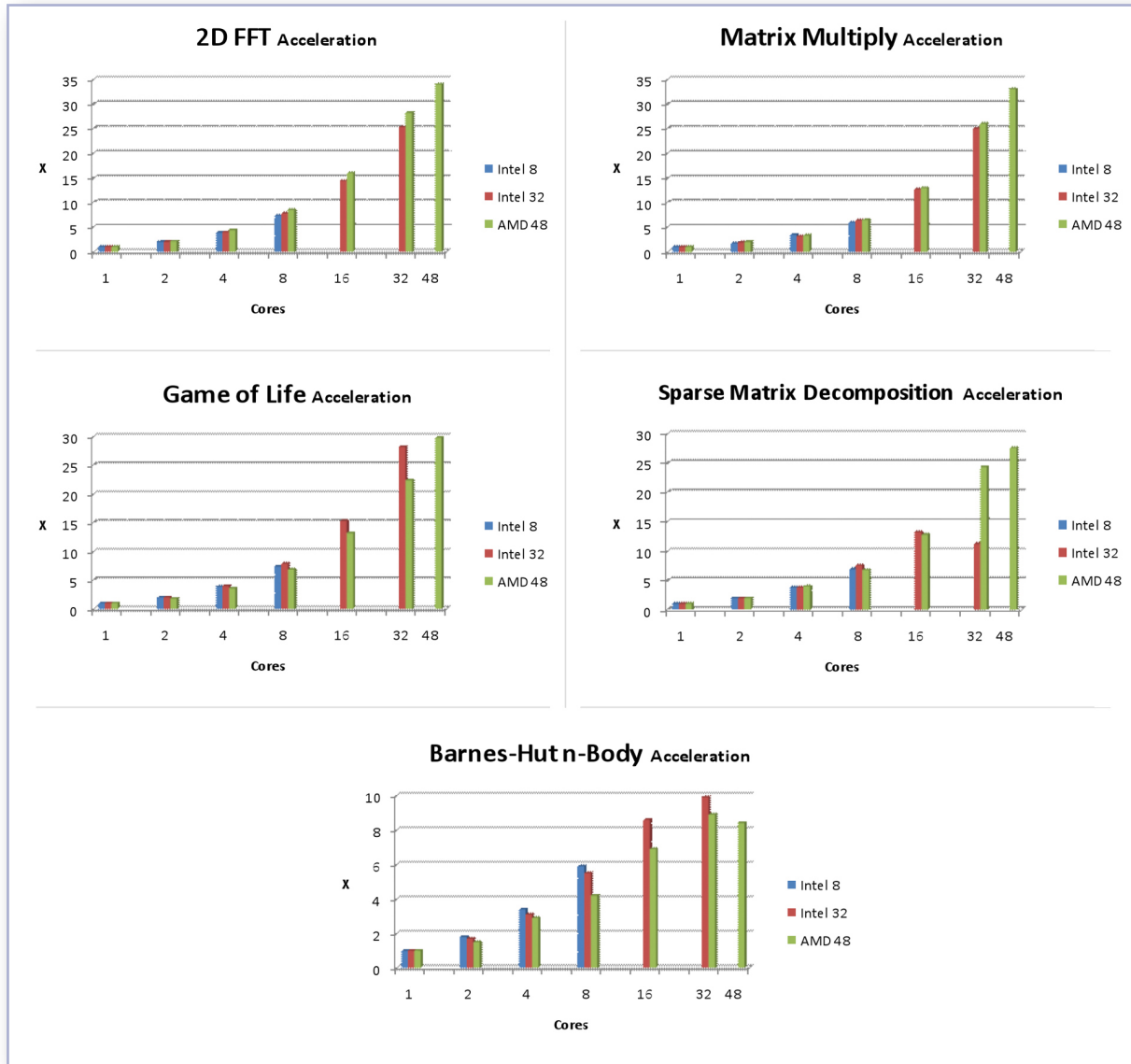


Figure 3. Parallel performance, with SequenceL

For each application, the tests show the efficiency of SequenceL parallelization running on 2, 4, 8, 16, 32 and 48 cores. The vertical axis indicates the equivalent number of fully utilized cores for each test. The three processor configurations used were: an Intel Nehalem EX 8 core on its own, Intel Nehalem EX 8 core in a 4-processor configuration for a total of 32 cores and an AMD Magny Cours 12 Core in a 4-processor configuration for a total of 48 cores

As can be seen, the results vary between the different processors and configurations. Direct configuration comparisons yield some surprises. It is odd, for example, that the Intel 32 core configuration is much more powerful running the Game of Life than the AMD configuration,



and yet so much less powerful than the AMD configuration when running Sparse Matrix Decomposition.

However, the most notable feature of these tests is SequenceL's consistent efficiency of parallelization. The results suggest that SequenceL will typically deliver 75% parallelization or better on 8 or fewer cores - which is a remarkable capability. The efficiency of parallelization appears to tail off a little as the number of cores rises, but it rarely falls below the 75% level. The worst performance shown was in the Barnes-Hut n-Body test where the AMD performance was less effective, however both the Intel and AMD architectures maintained nearly 50% efficiency.

Research into the Barnes-Hut n-Body tests identified a level-2 cache writing optimization anomaly caused by varying designs and cache sizes between the AMD and Intel chipsets. With a slight variation in the compiler's data sequencing, it will be possible to equal the performance gains in the other four tests. At the time of writing, this adjustment is in progress, and the tests will be rerun when it is completed and verified.

For readers who are interested in what these test applications are, here is a brief summary:

1. The **2 Dimensional Fast Fourier Transform (2D FFT)** test is a mathematical algorithm that is used for manipulating images, for example to apply a Gaussian blur to an image or part of an image. Naturally, this involves manipulating every pixel in the target area that needs to be blurred.
2. The **Matrix Multiply** test involves the multiplication of two matrices together. If you are unfamiliar with matrix mathematics think of this a very large set of multiplications of two numbers stored in two separate two-dimensional arrays putting the results into a third multi-dimensional array.
3. **The Game of Life** is the famous cellular based computer animation invented by Cambridge Mathematician John Conway, which appears to mimic the reproduction and dying off of cellular life within a 2 dimensional array.
4. **The Sparse Matrix Decomposition.** Sparse matrices are 2 dimensional arrays in which many of the cells of the array contain zero. Operations involving sparse matrices occur frequently in scientific and engineering contexts, for example in the solution of partial differential equations.
5. **Barnes-Hut n-Body Algorithm.** This is an algorithm for simulating the behavior of n-bodies moving in 3 dimensional space where each body exerts an influence on other bodies (for example a gravitational pull) according to proximity.

In Conclusion

A costly barrier to utilization of multicore technology is the continuous evolution to additional cores and the advancement of multicore architecture. Typically manual parallelization requires a major redevelopment of the application if the number of cores changes on the target CPU.

TMT has demonstrated in several projects that SequenceL is extremely transferable between environments. Software written for 4 core processors performs extremely well on 8 core, as well as 12 core processors and even on CPUs from different semiconductor manufacturers.



This is with no software rewrite or even recompilation. Transferability is critical to putting semiconductors back on the Moore's Law path.

In a series of benchmarks with three separate processor configurations, it demonstrated efficiency levels approaching 100 percent in many contexts, and consistently delivered above 75% percent efficiency. Additionally, SequenceL has been proved in mission critical applications in diverse environments.

These are remarkable results for an infant technology.

About The Bloor Group

The Bloor Group is a consulting, research and analyst firm that focuses on quality research and analysis of emerging information technologies across the whole spectrum of the IT industry. The firm's research focuses on understanding both the technical features and the business value of information technologies and how they are successfully implemented within modern computing environments. Additional information on The Bloor Group can be found at www.TheBloorGroup.com and www.TheVirtualCircle.com. The Bloor Group is the sole copyright holder of this publication.

□ 22214 Oban Drive □ Spicewood TX 78669 □ Tel: 512-524-3689 □

www.TheVirtualCircle.com

www.BloorGroup.com