The Bloor Group

THE PARALLEL COMPUTING IMPERATIVE

Why we need it, how it works and the benefits it delivers

Robin Bloor, Ph D

WHITE PAPER

Management Summary

By virtue of Moore’s Law, disruptive changes occur in computer power roughly every six years, because every six years chips run about ten times faster. The disruptive IT developments that Moore’s Law enabled are easy to identify: the PC, the GUI, the Internet, Global Search capability, Social networks and so on. But, in 2004, the chip market itself was disrupted, as it encountered a limit to CPU clock speeds.

This didn’t stop Moore’s Law in its tracks. Instead, it forced chip manufacturers to start adding multiple cores to CPU chips so that they doubled their processing power every 18 months by a different route. Unfortunately, this change of direction has become highly disruptive to software technologies, since most software is not designed to utilize multiple cores. Now, software now needs to run in parallel to exploit the extra cores.

In this paper we examine the alternatives available for achieving this, taking care to explain and describe different approaches to parallelism. In particular we review and compare two software platforms that are designed for parallelism: Hadoop and Pervasive DataRush.

Our conclusions are:

- There are several approaches to exploiting multicore CPUs, as follows:
  - **Instruction level parallelism:** This is not efficient for most business applications, but good for mathematically intensive tasks.
  - **Virtualization:** Good for consolidating servers running Linux or Windows but not particularly efficient in its use of CPU resource. Virtualization itself consumes considerable CPU power and so does each guest OS that it runs.
  - **Hadoop:** Essentially this is parallelization by data partitioning. It is highly scalable for very large data volumes of homogenous data, and designed both to be completely fault tolerant and to run across large numbers of servers. Currently it doesn’t fully exploit multicore.
  - **Pervasive DataRush:** This product offers the most flexible and efficient approach for single servers, especially SMP nodes with large numbers of chips and cores. It scales to data volumes in the tens of terabytes and programmers find DataRush’s approach to parallelism easy to adapt to, with little training.
- In comparing Hadoop to DataRush we conclude that the sweet spots of the two products are distinctly different. The DataRush sweet spot is on SMP servers, either on a single server of this ilk or small clusters of such servers. Hadoop is best on arrays of hundreds or thousands of servers. In terms of efficiency and speed, Pervasive Data Rush has benchmarked as 26 times faster than Hadoop - in a benchmark that favors Hadoop’s ability to partition data.
- Finally, in examining the possible business benefits of parallelism, we note that with products like Pervasive DataRush, the potential exists to dramatically accelerate many business activities (by a factor of 10 or more!), especially in the area of Business Intelligence (BI) which suits DataRush’s data flow approach. There are clearly opportunities here for disruptive new approaches to BI, exploiting “new generation” capabilities such as deep analytics and predictive analytics. With DataRush, it is possible to envisage wholly new BI application architectures based on parallelism.

The Nature of Parallel Processing

The idea of parallel processing is not particularly new and the nature of parallel systems isn't hard to understand. Nevertheless, for decades, the IT industry has not tried to exploit parallel processing, because it didn't need to. This has changed because of the advent of multicore chips, which need parallel processing to deliver the power they embody. The compute density on CPUs now increases every 18 months entirely through the doubling of processor cores. This trend is now likely to continue until it becomes impossible to add more cores to a CPU or impossible to exploit the power of additional cores.

For that reason, parallel processing will soon become mainstream. In this paper, we discuss the different technical strategies that are available to take advantage of multicore chips, including explanations of the technical strategies used by Pervasive DataRush and the Open Source Hadoop. For those with little knowledge of parallel software, we begin with a simple discussion of processes and threads.

Processes and Multi-threading

We tend to think of a program as a single executing block of code and it can be exactly that. If you are running a UNIX web server, for example, and you look to see what processes are running, you will discover that there are maybe 20 or so. If you have just one CPU with one core, then those processes will all be sharing that resource. If you have a CPU with two cores then the processes will be allocated fairly evenly between the two cores. If you have two CPUs with two cores then the processes will be shared out fairly evenly between the four cores. This is *process-level parallelism*. UNIX (and other operating systems) naturally implement this, although few operating systems schedule those CPU resources in an efficient manner. Consequently, it is not unusual to find that one core is very busy while another has very little work to do.

When some programs run, they run more than one "thread." Technically, threads are separate processes being run by the same program. When a program is

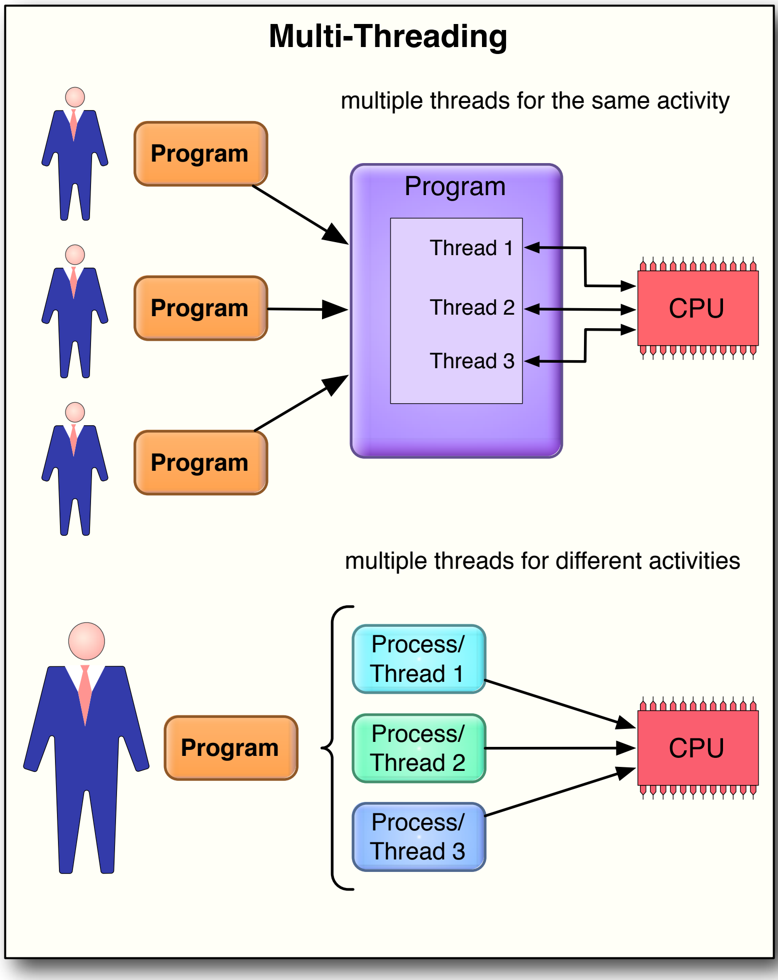

Figure 1. The threading of processes

being used concurrently by many users it can create a thread for each user. A common example is the client/server situation where the same application runs on many client PCs and a single server process (usually a database) manages the access to data.

This is illustrated in the top half of Figure 1. on the previous page. The database program starts a thread for each user and these threads run in a parallel manner. This provides a double benefit allowing for a high level of user concurrency as well as being more efficient. While one thread is waiting for data to arrive from disk, another can be sorting data in memory. All the threads can share the same core, or if that saturates, the work can be spread across more cores or even more CPUs. This is called ***thread-level parallelism***.

It is also possible to write programs that work in the way illustrated in the bottom half of Figure 1, where the program creates separate threads for separate activities. This is less common. The programmer has to work out which activities within the program could run in parallel and then start up separate threads for those activities, closing the threads down when the activities are finished. Few programmers have the skills and the tools for this.

Instruction Level Parallelism

It would be ideal if we could leave the management of parallelism to the compiler and the chip designer. Compilers with the help of on-chip logic could organize a program so that it automatically took advantage of parallelism. Programmers would not need to think about parallel operation and the industry everything could continue exactly as it is now.

So what are the possibilities in this area?

The important point to understand is that the typical business program will have some routines that must be executed in a specific order and other routines where the order of execution does not matter. So it is possible for the compiler to be clever and identify instructions that can be run in parallel across multiple cores. This is illustrated in Figure 2.

In this example, we show a single application running on a CPU with eight cores. For simplicity, in the illustration we show all program modules as being either one or two "units" in size or in the case of the units that can be parallelized, eight and sixteen units respectively. The effect of running this application in parallel is to almost double the speed of execution.

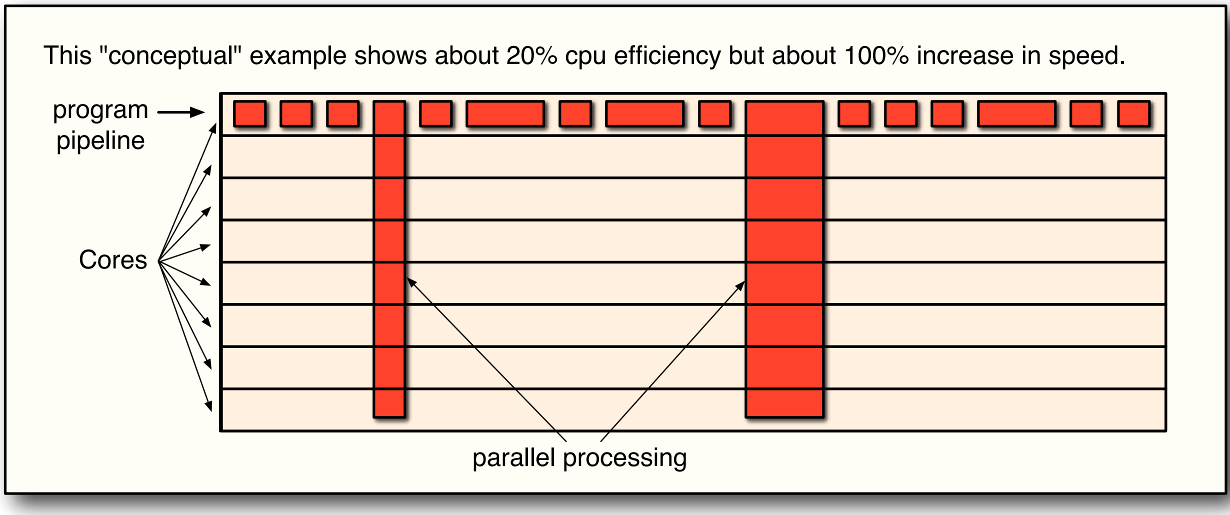
This "conceptual" example shows about 20% cpu efficiency but about 100% increase in speed.
program pipeline
Cores
parallel processing

Figure 2. Instruction Level Parallelism

However, when we do that, the average utilization of the whole CPU is less than 25 percent. It's faster, but not efficient. Now consider what would happen if there were 16 cores. That would speed the application up again, but only by about a further 5 percent, and the average utilization the CPU would go down to about 12 percent. In general instruction level parallelism is horribly inefficient.

Scientific applications with considerable amounts of mathematical calculations can sometimes be parallelized effectively in this way, but scientific and academic computer users have been employing such parallelism for many years. There's not a great deal of opportunity for new exploitation of parallelism here. ***Instruction-level parallelism*** is, at best, a niche solution.

The Parallelization Alternatives

We can summarize the possibilities for parallelization simply enough.

- **Process-Level Parallelism:** It can be done at the process level. It will inevitably be inefficient when processes are swapped one for another. The problem is that this swapping not only disrupts the processing activity, but will most likely disrupt the data cache, causing much data swapping between cache and main memory. If the processes are intelligently scheduled it can deliver some efficiency.
- **Thread-Level Parallelism:** When scheduled effectively, this is the best of the alternatives. It stands the best chance of keeping all CPUs and all cores busy doing useful work, but most programmers have little idea how to do this well.
- **Instruction-Level Parallelism:** This is the least attractive alternative, with current technology, it is likely to be very inefficient except in scientific and academic applications.

Parallelization And Virtualization

For years CPUs delivered far more computer power than many applications needed, and yet it was not advisable to run more than one application per server. Neither of the popular and inexpensive operating systems, Windows and Linux, had the robustness for running multiple applications. This has resulted in an enthusiasm for deploying virtual machines.

A virtual machine is a complete emulation of a computer run from within another computer. A "super operating system," called a hypervisor, manages one or more "guest" operating systems providing them with all the resources needed to run applications. As far as the guest operating system is concerned, it has a whole computer to itself, whereas it really has only the CPU, memory and I/O resources that the hypervisor has allocated to it. The arrangement is illustrated in Figure 3, which shows a hypervisor managing four guest operating systems. There is no

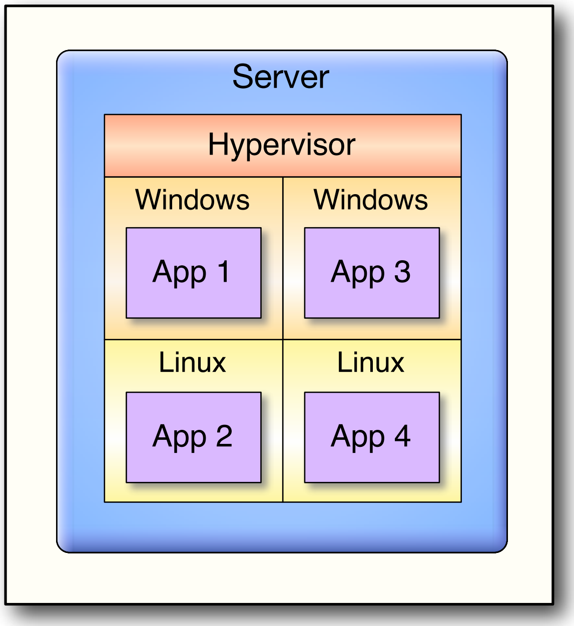

Figure 3. Virtual Partitions

necessity here for the CPU to be multicore. This works fine on single core CPUs. But, by 2002, computers had become so powerful that the majority of commodity Intel servers running a single application were using less than 10 percent of their resources. There was already a great deal of surplus power that virtualization could utilize before multicore CPUs made their entrance.

Virtualization implements process parallelism enabling a single computer to replace several computers. As more cores are added to CPU chips, servers will no doubt be capable of running larger numbers of virtual machines. It is a welcome capability, but it is efficient only in the sense that it is more efficient than what went before. Note, for example, that when we run four guest operating systems, we are running five different operating systems rather than one. There's a huge amount of redundancy in that. Anecdotal evidence suggest that this kind of virtual machine arrangement rarely achieves much more than 40 percent CPU utilization - and that's counting the resources consumed by the many operating systems in the 40 percent.

Large Scale Parallelization: Hadoop and MapReduce

Google deploys hundreds of thousands of servers; more than any other company. It is not surprising, given that Google is the premier search engine and the web has over 50 billion web pages. So naturally, Google uses parallelism in almost every area of its operation to process that data. Indeed, Google invented a parallel software framework specifically for processing very large amounts of data held on disk.

The framework is called MapReduce and it has been extended for general use through an open source product (and project) called Hadoop. Hadoop works in tandem with MapReduce enabling software components written in Java to be added to the MapReduce framework. It also contributes the Hadoop Distributed File System (HDFS), which allows very large data files to be distributed across all the nodes in the configuration. Hadoop, along with MapReduce provides one of the few software platforms for programmers to build parallel applications directly.

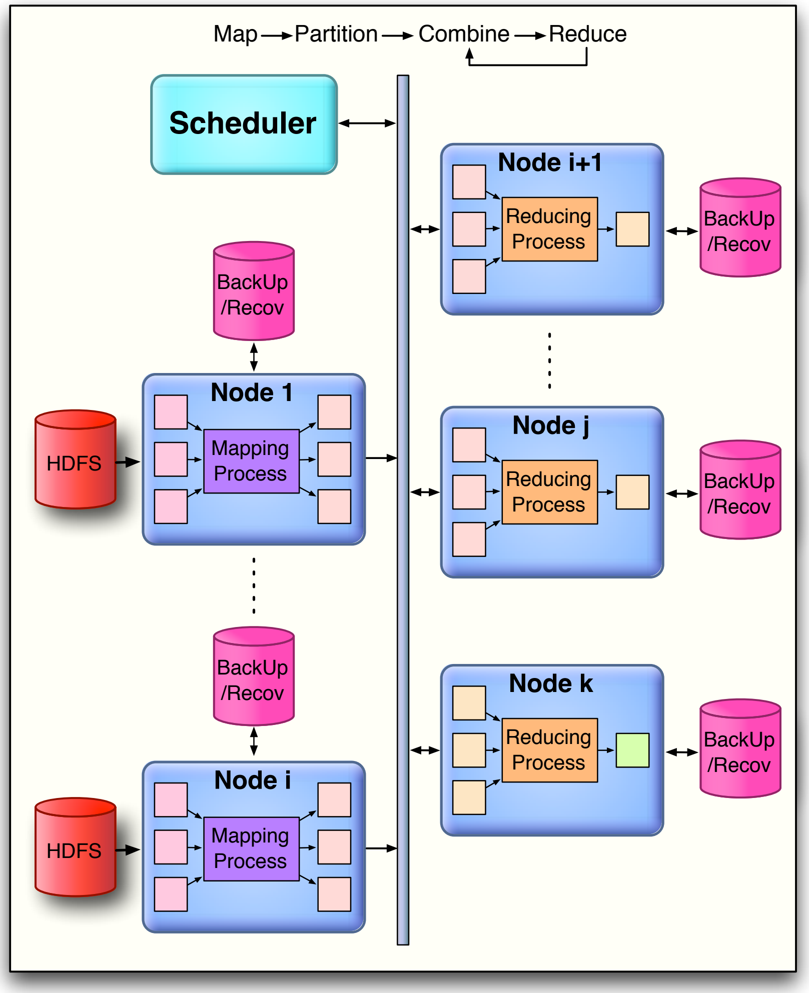

Figure 4. Hadoop and MapReduce

Figure 4 illustrates how it works. The whole parallel environment is based around having a mapping function which partitions data and passes the data to a reducing function (hence the name: MapReduce). All the data records consist of a simple "key and value" pair. An example could be a log file, consisting of message codes (the key) and the details of the condition being reported. The mapping logic processes these records, perhaps just removing some details and rejecting some records. This produces an intermediate set of records (consisting of a key and data) which are partitioned according to key. So the mapping process is a partitioning process.

Once the mapping stage is complete each node passes the resulting records to other nodes so that the records in each partition can be combined and the "reduce" logic applied to them. This process can iterate, if necessary until the final answer set is produced. A simple example will make this clearer.

Imagine we have a very very large log file containing messages and message codes and we simply want to count each type of message record. It could work in the following way:

The log file is loaded into the file system and each mapping node will read some of the log records. The mappers will look at each record they read and output a key value pair containing the message code as the key and "1" as the value (the count of occurrences). The reducer(s) will sort by the key and aggregate the counts. With further reductions eventually we arrive at the result; a map of distinct keys with their overall counts from all inputs.

While this example may seem facile, if we had a very large fact table of the type that might reside in a data warehouse, we could execute SQL queries in the same way. The ***map*** process would be the SQL SELECT and the ***reduce*** process could simply be the sorting and merging of results. You can add any kind of logic to either the ***map*** or the ***reduce*** step and you can also have multiple map and reduce cycles for a single task.

An important point feature of the Hadoop/MapReduce platform is that it is built to recover from the failure of any node. The setup is fully redundant in that every server logs what it is doing and can be recovered from the backup/recovery file, including the scheduler which manages the activity throughout the grid.

Hadoop and MapReduce: Strengths and Limitations

The most obvious limitation of this approach to parallelization is that it is designed primarily for enabling large population of servers to process very large volumes of similar data records quickly. Most organizations do not have many applications that fit the picture. Even if they did, they rarely have pools of available servers to spread the workload over. They might be able to rent resources in the cloud, but then they would face the problem of loading data.

Nevertheless, if you are able to use it, Hadoop with MapReduce can be very fast. For example, in one benchmark, it sorted a terabyte of data (in 100 byte records) in 62 seconds using a 1460 node configuration. Similarly, it sorted a petabyte of data (1000 times as much data) in 975 minutes (i.e. taking about 1000 times longer) using a 3658 node configuration. At the time of writing the largest implementation of Hadoop runs on roughly 5000 nodes, but that does not constitute a limit.

The Hadoop framework scales very effectively over very large numbers of nodes and in some situations it will outperforms traditional massively parallel databases by a wide margin.

Some high-end database products, such as Green Plum and Aster, include MapReduce as an option for processing large amounts of data.

The HDFS automatically keeps three copies of all data, by default, and this along with the recovery files, makes it possible for Hadoop to recover from the failure of any node. This is important for a system that can be using hundreds of nodes at once, because the probability of any node failing is multiplied up by the number of nodes.

However, Hadoop has not yet been engineered to make efficient use of multicore CPUs and neither does it fully exploit threads. This is probably because Hadoop's primary goal is scalability (for very large amounts of data) and fault tolerance, rather than CPU efficiency.

Pervasive DataRush

Pervasive DataRush, from Pervasive Software, offers a more versatile approach than Hadoop and MapReduce. It is more flexible in several ways. It does not insist on a specific file system, happily accessing data from any source and delivering it to any target. It does not impose a specific approach to how data is processed - it does not force a "Map-and-then-Reduce" logic on the programmer or anything similar. Also, it provides a broader approach to parallelization.

Pervasive DataRush is not a direct Hadoop competitor, as it was designed primarily to optimize the performance of a single server rather than to process vast amounts of data on hundreds or thousands of servers. Its focus is distinctly different, although there are applications that could be built using either framework.

An example is provided by the results of the Malstone B-10 benchmark, shown in the adjacent box.

In one way, the benchmark plays to Hadoop's strengths because it involves a very large volume of identical records (960.6 Gbytes). DataRush would be favored more by a complex benchmark consisting of a data flow to which several different processes are applied.

The server that DataRush used had 4 CPUs x 8 cores per CPU giving a 32 core total. Because Hadoop doesn't currently make use of multiple cores, Hadoop's 40 CPUs amounted to just 40 effective cores.

The difference between the two benchmarks is surprising. DataRush completed the task over 20 times faster using fewer cores. The speed it achieved is remarkable by any standard and it merits some explanation.

DataRush and Hadoop Compared

The MalStone B-10 Benchmark

The benchmark uses a file of 10 billion records that represent visits to a web site over a one year period. Each record consists of an Event ID, Timestamp, Site ID, Compromise Flag and Entity ID. The benchmark computes a ratio as follows: For each site ***w***, and for all entities that visited the site in week ***d*** or earlier calculate the percentage of visits for which the entity became compromised at any time between the visit and the end of the week ***d***.

Hadoop With MapReduce

Hardware: 20 node cluster with 2 dual core CPUs on each node

Time Taken: **840 minutes**

Pervasive DataRush

Hardware: SMP node with 4 CPUs each with 8 cores on a single board

Time Taken: **31.5 minutes**

Data Partitioning and Process Partitioning

The primary tactic of Hadoop with MapReduce is to partition the data in order to process it in parallel. This approach is illustrated in the diagram at the top of Figure 5, which shows data being evenly split across four cores and each core running exactly the same program or process. If handled efficiently then this will speed up the processing of data by a factor of four. However, the data first has to be partitioned and, after it has been processed, it will probably need to be merged. That activity adds to the processing load.

Hadoop with MapReduce is designed to implement this approach to parallelism and thus it is most appropriate to data processing problems where this is the best tactic.

An alternative to partitioning the data to be processed is to partitioning the processes themselves, rather than data. This is illustrated in the lower diagram of Figure 5. Imagine a task that involves four stages. First extracting data from several files, then merge it with other data, then transform some of the values in the data records, then calculating some summary totals; ***extract-merge-transform-summarize.***

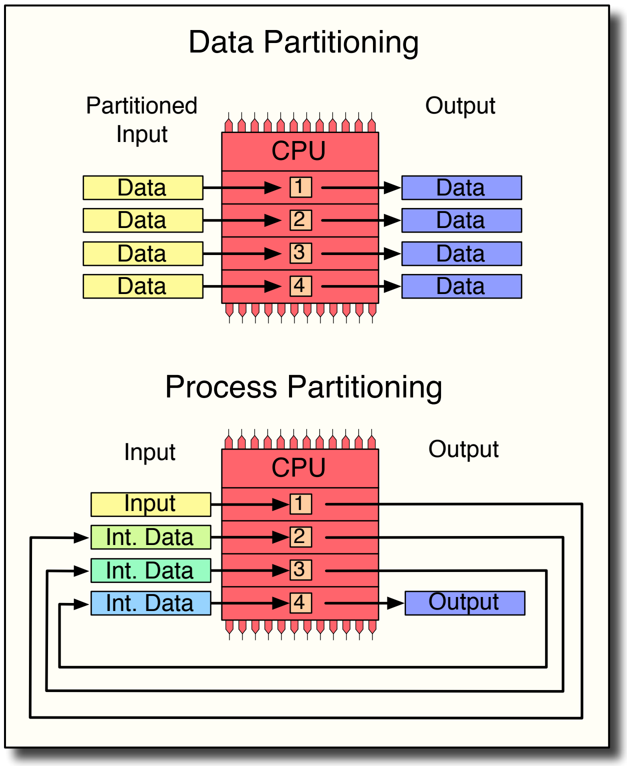

Figure 5. Data and Process Partitioning

We can begin any one of these processes as soon as the previous process starts to produce data, so we can work on all four at once. The data flows from one process to the next, and it stays in memory as it passes from one process to another. Process partitioning naturally complements data partitioning. For example, if one process is going too slow and data starts to queue up in front of it, then it is only necessary to give that slow program more power (an extra core for example) and the workload will become more balanced.

DataRush employs both data and process partitioning, whereas Hadoop only uses data partitioning. DataRush is organized so that whole data flows can be specified from end-to-end and hence parallelization is optimized over the whole work flow. A typical application is the moving of data into a data warehouse or operational data store. Depending on circumstances, there will normally be several steps to such a process such as: extract data, merge the extracted data, apply transformations, profile and cleanse, write to target database.

To make intelligent design decisions, the programmer needs to understand the properties of the data in respect of the whole data flow. So in the case of: ***extract - merge - transform - profile - cleanse - write***, the programmer needs to consider how much data is coming from which sources and whether it can be segmented into appropriate similar sized subsets. Will it be possible to run the merge concurrently with the extract? In fact, will it be possible to begin any stage while the previous stage is in progress? What are the ratios, in terms of file size between the outputs of each stage? And so on.

DataRush lets the programmer specify how the data flow will be parallelized for the whole process or any stage of the whole process, or any part of any stage. It includes a library of parallel operators that the programmer can select from. Once the end-to-end process is specified and the individual Java components of the process are written, the DataRush Engine manages the fine details of parallelizing threads, making the best use of the resources available. So the programmer defines the approach to parallelizing the processing of the data flow and the engine implements it. The parallel operators adjust their mode of operation at run time to conform to the resources available. So, if an 8 core server is replaced by a 16 core server there is no need to change any of the code, yet the application will automatically scale to utilize 16 cores.

DataRush runs in a Java Virtual Machine (JVM) with programmers using Java operators to manipulate data and the Pervasive DataRush Engine implementing the operations that the programmers specify. The programmer specifies the approach to parallelization.

By contrast, if you write programs for the Hadoop framework you need to have a fundamental understanding of how MapReduce works and how the HDFS works, but you don't need to consider how to parallelize the processing. It will be done via data partitioning. With DataRush you choose the most appropriate approach to parallelization. Happily, this turns out to be a skill that is easily learned. Programmers can normally understand how to use DataRush in a few days. They become productive quickly.

Thread-Level Parallelism

DataRush's approach to parallelization begins at the level of program threads. As we have already described, a program can have multiple threads which run within the same core at the same time. Of course a core can only process one thread at a time, but normally a thread will "hang" while it waits for the computer to do something such as fetch more data into cache from memory or even issue a request to disk.

At that point a context switch takes place, with the core unloading that thread from its pipeline and loading a different thread so that it has something to work on. When that next thread comes to a halt, it can load yet another thread and work on that. We have illustrated this simplistically in Figure 6, which represents a 4 core CPU with different numbers of threads running on each of the cores. The exact number of threads that a CPU core can work on at any time varies according to what each is doing and how frequently each thread is likely to have to wait for data.

Managing the threads in such a way as to ensure that the CPU core is rarely idle is a complex task beyond the capabilities of most programmers. This is what DataRush's parallel engine

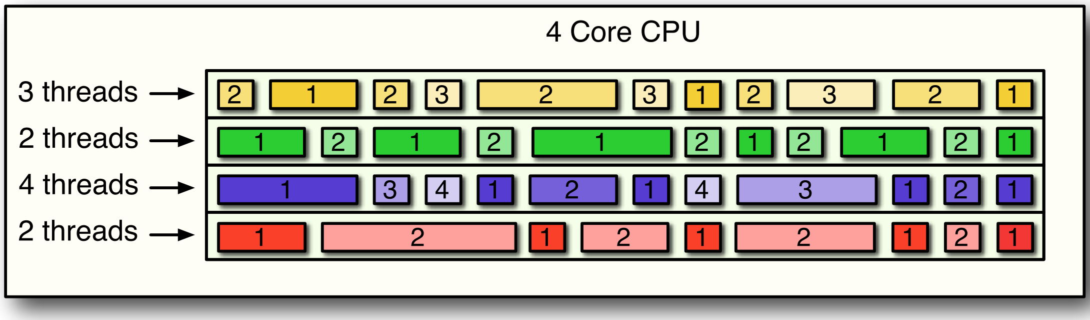

Figure 6. The Interweaving of Threads

does. But that isn't all that it does. To make most efficient use of a CPU core it is also necessary to try to ensure that data that needs to be processed is "close to the CPU core." By close, we mean really close.

It is best if the data is in the data cache on the chip (Level 1 cache) or very close to the chip (Level 2 cache) rather than main memory. Processing data in Level 1 cache is faster than processing data in Level 2 cache which in turn is faster than processing data in main memory. The exact ratio depend upon technical factors such as the type of chip its clock rate - but as a rough guide, data in cache can be processed four times as fast as data in memory.

There is thus a data flow pipeline and the whole arrangement is most efficient if the two caches are kept full of relevant data. DataRush applies some intelligence here, limiting data movement and combining operations so that, if possible, all processing of data is done once data moves into the cache.

Areas of Application

Figure 7. below roughly illustrates the typical areas of application for DataRush and Hadoop/ MapReduce. Note that the horizontal axis of the graph (data volume) is logarithmic, ranging from megabytes through to exabytes. The areas shown for different kinds of applications are approximate, so there are exceptions. Corporate computing usually gives rise to few high volume applications. Currently, corporate databases are generally smaller than a terabyte in size, although some are larger and data volumes are forever expanding. High performance computing tends to deal with higher data volumes and highly complex applications using

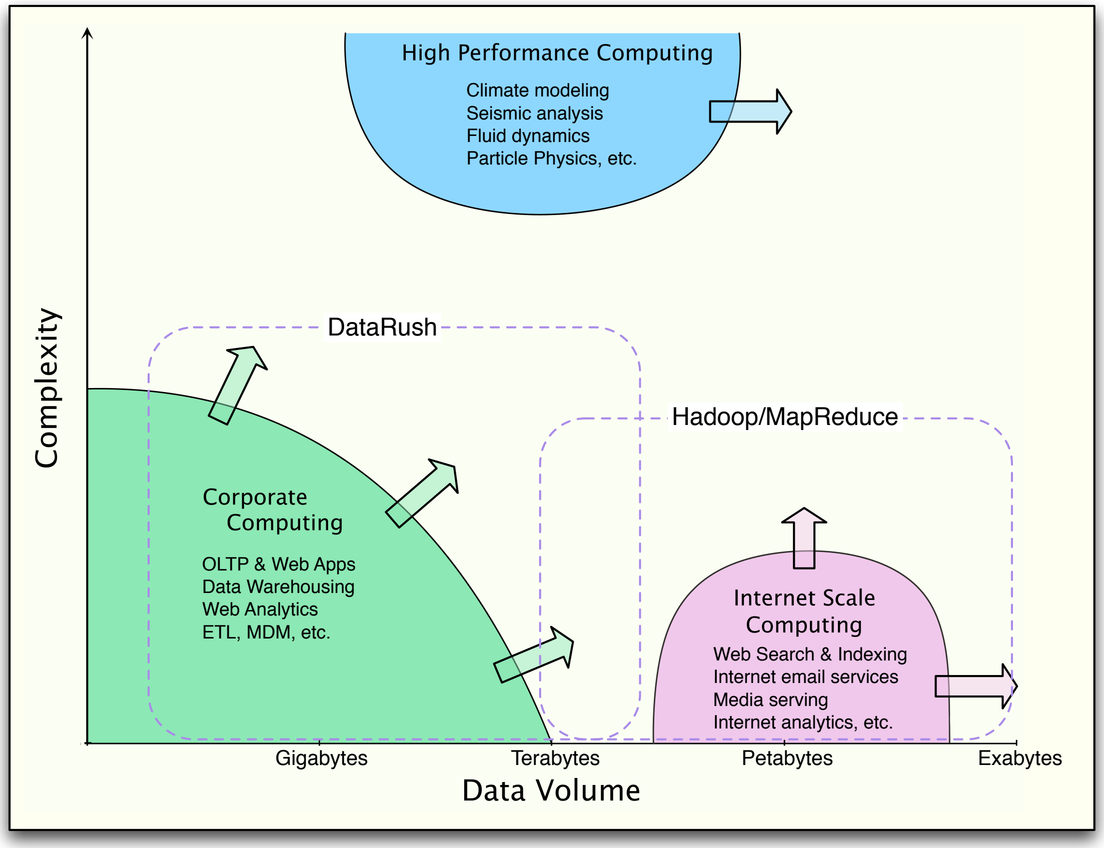

Figure 7. A Map of Applications, by Complexity and Data Volume

numerical data. Normally, machines are configured specially for such applications and the parallelism of the software is hand-crafted. Finally there is Internet scale computing characterized by the activities of Google, Yahoo!, Facebook and other cloud-based companies. Data volumes are up to the tens and even hundreds of petabytes. The natural areas of application of both DataRush and Hadoop/MapReduce have some overlap in the area of single terabyte to tens of terabyte data volumes.

DataRush is built to be most effective in single server configuration where a number of multicore CPUs share memory. Nowadays such servers are built inexpensively from commodity circuit boards. In such environments it excels with workloads that can be handled on the single server; with data being fed from local disk and the end-to-end process requiring no network access network. It will inevitably be faster in such an environment than when the workload is spread across multiple servers. This suits the majority of corporate computing applications and could be used to effect on complex applications with relatively small data volumes.

The mode of operation of Hadoop and MapReduce, in comparison, put it at a disadvantage in terms of outright speed. Between the ***map*** step and the ***reduce*** step there is an exchange of data which must inevitably impose network latency. Additionally, the resilience of the system, enabling recovery from any kind of failure, also imposes a performance penalty. Furthermore, the highly formal structure of the MapReduce process means that CPU cores will have to devote time to data partitioning. DataRush does not insist on data partitioning, so no processing need be devoted to that activity unless it is required. With DataRush, the designer has the choice of when to partition and when to avoid doing so.

When you add all of this up, it is clear why DataRush outperforms Hadoop so dramatically in the MalStone B-10 benchmark. DataRush is optimized for end-to-end performance (it scales up) while Hadoop is optimized for a partitioned approach (it scales out) and resilience. It's also worth noting that, in order to work at all for the benchmark we illustrated, Hadoop required a long data load of the data into the HDFS partitioned file system - a fact which is not taken into account in the benchmark result.

Given the way that Hadoop is architected it is unlikely that it will ever be able to scale up in the way that DataRush does. It has partitioning burned into its DNA and partitioning is often not the best way to go.

A Parallel IT Universe?

Conveniently for DataRush, the number of server CPU cores just keeps on growing. For example, this year will see the introduction of commodity servers with 24 cores and even 48 cores (based on AMD's 12-core Magny-Cours processors). Additionally we will see 32 core servers (based on Intel's Nehalem EX processors with 8 cores and hyperthreading). On an SMP server, DataRush scales almost linearly. On a 48-core server it could be expected to perform the MalStone B-10 benchmark in about 21 minutes. It will not stop at 48 cores. It will not be long before there are servers with over 100 cores - unless companies cease to buy such hardware.

The trend to more and more cores poses an interesting question:

Is parallelism an approach to programming that will eventually become the norm, simply because it has to?

At first blush this seems unlikely, simply because there is such a vast amount of software already written and deployed, which does not and cannot get much advantage from parallel operation. In the past corporations have rarely rewritten software just for the sake of running it on different hardware. On the other hand, the chip industry needs to keep innovating in order to make current chips obsolete or else it will be condemned to see its margins fall and its markets decline. Its primary direction right is to simply add more cores, and it will continue to do so until it is clear that the newest generation of chips is not delivering more power than the previous generation.

There are two reasons why parallelism should become the norm:

1. DataRush demonstrates that enabling programmers and designers to use parallelism can be achieved without significant amounts of retraining. If there is no significant barrier to adopting parallelism, then the chances of it coming into general use are far greater.
2. There are many commercial applications, particularly in the area of Business Intelligence, which are essentially a data flow from end to end. All such applications will run far faster if they run in parallel. In some cases 10 times as fast. In other cases 100 times as fast.

The economic advantages of running software in parallel are clear: You use less resource and therefore pay less for the resource, and the resource consumes less electricity, and floor space and management effort. This is a significant advantage and might on its own justify exploring the possibilities of products like DataRush. But there is a bigger advantage.

The primary benefit of parallel processing is that software runs an order of magnitude faster.

The benefits that huge increases in speed can deliver are more profound the more you consider them. So let's take a considered look at the business advantages of speed.

The Competitive Advantage of Speed

It is easy to see why the acceleration of a business process will usually deliver business benefits. A business consists of many different processes. Increase the speed of any one of these, even in a small way, and you either end up delivering a service to someone faster (another department, a business partner or a customer) or you increase capacity. You may even increase accuracy, since instead of sampling data it may be possible to process the whole file. In any event, the user wins.

Information technology has delivered the benefit of speed in many ways over many decades as the speed of technology itself has increased. Sometimes businesses find dramatically successful ways of putting this extra power and speed to use, but often it makes less difference - everything goes a little faster, but no dramatic advantage is obtained by anyone. To understand why, we need to examine the ways that automation supports business processes.

Almost all business processes proceed in repeating cycles. Some time cycles are short, such as the taking of an order from a customer and some are long, such as completing an R&D

project. A few cycles are fixed by external forces, such as the legal requirement to file financial results on an annual basis and the market requirement to declare financial results on a quarterly basis. However most business process cycles are not fixed at all and businesses can gain competitive advantage by speeding up such cycles.

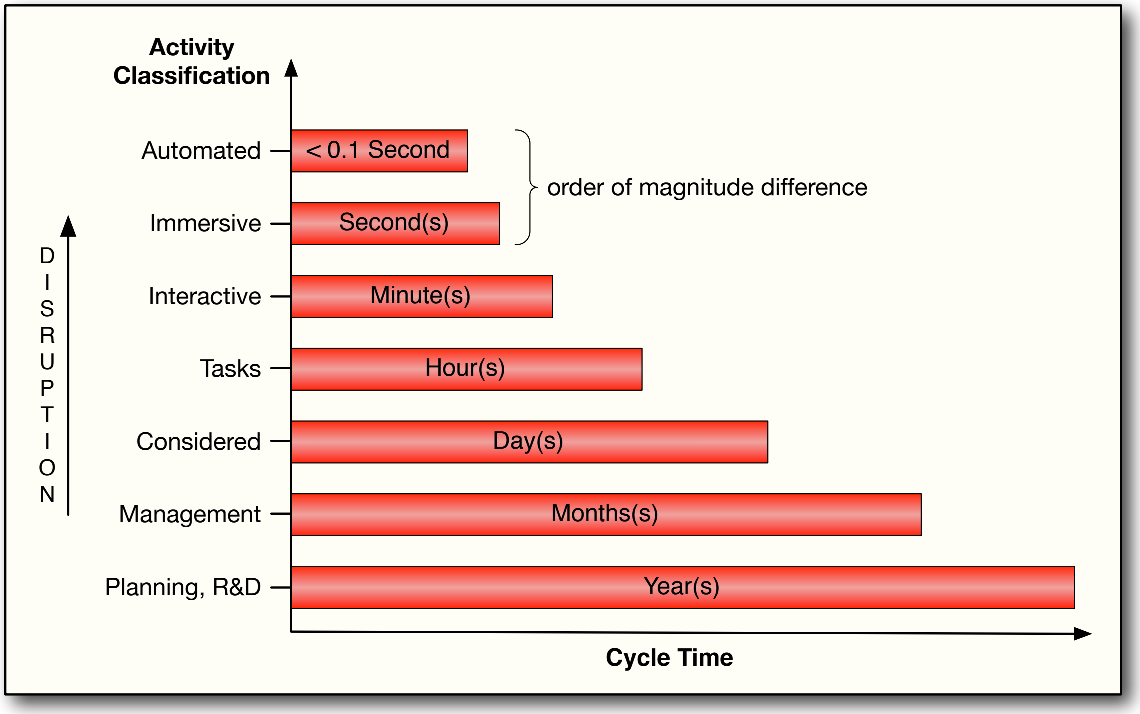

Figure 8. Human Activity Cycles and Business Process Cycles

We can categorize business process cycles according to their periodicity. Figure 8. illustrates a scheme for doing this, using the following cycles:

1. **Yearly:** There are multiyear cycles (such as election cycles) but most long business cycles are annual or "of the order of a year." The annual cycle of corporate reporting tends to encourage other business cycles to beat to the same rhythm. Yearly activities tend to be planning or project based. A typical example is Product R&D. It is notable that Nokia came to dominate the mobile phone market in the late 1990s by pulling the Product R&D cycle down to 6 months. This enabled it to introduce 2 new ranges of handsets in a single year. Its major competitor at the time, Ericsson, was taken by surprise and never recovered. In engineering R&D, the use of parallel processing is increasing the speed at which new engineering prototypes can the built and this in turn is accelerating Product R&D cycles.

2. **Month(s):** While there are quarterly cycles (often harmonizing with the seasons) the typical management "pulse" is monthly, with management targets, say for manufacturing, happening in a convenient rhythm with monthly accounting and monthly sales. No matter how convenient such a rhythm may be, having manufacturing beat to a much faster rhythm offers clear business advantage. Dell is one of several obvious manufacturing operations that pulled this monthly rhythm down very close to a daily rhythm, with time from an order being place on the web to the PC leaving the factory being reduced to around a day. This made it possible for the

customer to have a "configured PC" rather than a standard PC from a range. For a while it gave Dell dominance of the PC market.

3. **Day(s):** The cycles here usually involve a high level of human activity and thought. Examples include many activities from creating spreadsheets to analyzing sales figures. Such activity often involves BI. Indeed for some large web sites, for example, it can take a day to load the web logs and analyzing those logs is painfully slow. Analyzing such data can take hours to days to come to a conclusion. The desire for much faster analytics to profile web site visitors is generating interest in approaches to "real time customer profiling."
4. **Hours(s):** The individual tasks within a project, whether it involves recruiting new staff, making a sales call, fixing a program bug or waiting or running a query on a large database tend to fall into this category. Loading data into a data warehouse normally takes hours.
5. **Minute(s):** Here we encounter much shorter tasks that tend to be interactive, such as responding to an email, entering transactional information into a computer or looking up information on the web. Nowadays this also includes some BI queries, to the point where commentators now sometimes talk of real-time BI.
6. **Second(s):** Once we get to this level we are dealing with "immersive" interface activity. In many computer activities from CAD to word processing, individual steps in the process happen in seconds. In this area, the value that near real-time analytics can bring is clear. Business intelligence becomes truly interactive. This kind of application demands parallelism since users quickly tire of waiting for responses. Within a second is what we expect. Half a second is better.
7. **<0.1 Second:** Biologically, less than one tenth of a second constitutes "realtime." One tenth of a second is speed of human thought. Human beings cannot respond intelligently to any activity faster than this. Once you have processes running at this level of response the natural thing to try to do is eliminate the human being and make it fully automated - like a process control system or an automated trading system.

Time intervals which mark out the human appreciation of time also determine how an activity is carried out. Consider the difference between writing a letter (measured in hours), writing an email (minutes), sending a text (seconds). Communications have migrated in this way, from letter to email to text with, in some cases, no change in the amount of important information passed. If you text, it's faster than email which is a lot faster than a letter.

Clearly there is business advantage in faster communication. However, it wasn't until the advent of Twitter that the full impact of the texting revolution hit home on most people. Twitter has become the fastest news channel in the world. It has millions of volunteer "reporters" working in parallel to bring you the news a lot faster than any other source. When "Sully" Sullenberger landed the US Airways Airbus on the Hudson in January 2009, it was reported first on Twitter.

So now there are parallel engines that process Twitter streams trawling them for news of any kind, from earthquakes to company announcements. The reporting of news had collapsed from a latency of minutes to a latency of seconds. Reuters and AP may not have realized it immediately, but they surely know now. There was a new player on the field that could sprint a lot faster than they could.

Orders of Magnitude

The reality is this:

Moving an activity from one time cycle classification to a faster classification will always deliver a big business pay-off.

And it should do. The time intervals that we have chosen to highlight are at least one order of magnitude (a factor of 10) apart. Nokia obtained huge business advantage simply by cutting its product R&D cycle by 50 percent. It never changed the business process, it just accelerated it. When you speed up a business process by a factor of 10 or more, you are likely to change the whole nature of the process, as Twitter has done.

In the immediate future parallelism will dramatically speed up all computer processes that run without human intervention in cycles of day(s), hour(s), minutes(s) and even second(s). Products like DataRush can and will collapse those cycle times by at least a full order of magnitude and possibly two orders of magnitude. And in doing that they will create business opportunities and change the nature of whole business processes.

An obvious area of impact will be in the data warehouse processes of ETL and MDM. With such processes executing at lightning speeds it will be possible to rethink the whole of data warehouse and many of the BI capabilities it feeds. The old established analytics products will be challenged by new speedier products that not only run faster but redefine the nature of analytics. Most likely, they will process data flows rather than simply apply algorithms to data sets. The peripheral activity (reading data, joining, merging, sorting, transforming, filtering and so on) will probably be done by the analytics tool itself.

And because analytics can itself run much faster, rather than just identifying correlations and building customer profiles it may become predictive or may work on a real-time model of the whole market. The stage is being set for real-time BI and real-time decision management.

There may be a huge disruption in the offing here. BI processes have always run in the wake of transactional systems, providing feedback on what took place days "after the fact." With the speeds that parallelism delivers it will be possible to run BI processes concurrently with transactional systems, enabling their analysis to be fed directly into transactional systems and acted on immediately.

We have already seen early attempts at real-time intelligence systems with the data streaming products like StreamBase, Coral8 and others. These can now be seen as a first generation of products that precede much more comprehensive capabilities based on fully fledged data flow architectures.

An era of parallelism is about to begin and it will inevitably be highly disruptive to established technologies and architectures.

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 ❏

w w w . T h e V i r t u a l C i r c l e . c o m

w w w . B l o o r G r o u p . c o m