A Multi-Core Pipelined Architecture for Parallel Computing

Duoduo Liao[*1], Simon Y. Berkovich [2]

Computing for Geospatial Research Institute
Department of Computer Science, George Washington University
801 22nd Street NW, Washington DC 20052 U.S.A.
[*1] dliao@gwu.edu; [2] berkov@gwu.edu

Abstract- **Parallel programming on multi-core processors has become the industry's biggest software challenge. This paper proposes a novel parallel architecture for executing sequential programs using multi-core pipelining based on program slicing by a new memory/cache dynamic management technology. The new architecture is very suitable for processing large geospatial data in parallel without parallel programming. This paper presents a new architecture for parallel computation that addresses the problem of requiring to relocate data from one memory hierarchy to another in a multi-core environment. A new memory management technology inserts a layer of abstraction between the processor and the memory hierarchy, allowing the data to stay in one place while the processor effectively migrates as tasks change. The new architecture can make full use of the pipeline and automatically partition data then schedule them onto multi-cores through the pipeline. The most important advantage of this architecture is that most existing sequential programs can be directly used with nearly no change, unlike conventional parallel programming which has to take into account scheduling, load balancing, and data distribution. The new parallel architecture can also be successfully applied to other multi-core/many-core architectures or heterogeneous systems. In this paper, the design of the new multi-core architecture is described in detail. The time complexity and performance analysis are discussed in depth. The experimental results and performance comparison with existing multi-core architectures demonstrate the effectiveness, flexibility, and diversity of the new architecture, in particular, for Big Data parallel processing.**

Keywords- Multi-Core Architecture; Pipelining; Sequential Programs; Program Slicing; Crossbar Switching; Parallel Computing; Big Data

I. INTRODUCTION

As multi-core architectures gain widespread use, it becomes increasingly important to be able to harness their additional processing to achieve higher performance. However, exploiting parallel cores to improve single-program performance is difficult from a programmer's perspective because most existing programming languages dictate a sequential method of execution. Parallel programming on multi-core processors has become the industry's biggest software challenge.

Because multi-core hardware architectures are changed to parallel structures, single-processor based software has to be optimized or even rewritten with much work to meet the hardware constraints. However, if we can change the hardware to eliminate the constraints, most of existing single-processor software programs may be directly used with minimum changes or even without any change.

For this purpose, this paper proposes a new parallel architecture using multi-core pipelining based on program slicing by crossbar switching and a new memory/cache dynamic management technology. The new architecture can automatically partition data and schedule them onto multi-cores through the pipeline. This architecture provides a simple and effective solution to the on-the-fly computations by transferring the operating states from core to core. The most important advantage is that it only requires practically the same software as currently used based on single-processor system, instead of conventional parallel computing methods, such as threading, load balancing, and scheduling.

The rest of this paper is organized as follows: Section 2 gives a broad overview of the backgrounds and related work thus far. Section 3 describes the detailed design of the new parallel architecture using multi-core pipelining based on program slicing by switch applied. It contains crossbar switch based multi-core memory and cache architecture, multi-core pipeline organization, timing diagram, and program requirements. Section 4 gives the time complexity, performance, and experimental analysis. In particular, the examples of large geospatial data processing, such as Digital Elevation Model (DEM) generation from Light Detection and Ranging (LIDAR) dataset, are discussed. Finally, the summary and advantages are concluded in Section 5.

II. BACKGROUNDS AND RELATED WORK

A. Conventional Parallel Computing

In general, there are three major approaches used for multiprocessor processing [9] [10]:

- Data-parallel: partitions data and schedule them onto the multiple processors,
- Task-parallel: Partitions a program into functions/tasks and schedule them onto the multiple processors,
- Pipeline-parallel: decomposes a program and run each state simultaneously on sequential elements of the data flow.

The first approach is suitable for the data-independent circumstance. However, the scheduling may be complicated depending on the application programs.

For the second method, in practice, it is often difficult to divide a program in such a way that separate CPUs can execute different portions without interfering with each other. Furthermore, this type of parallel processing requires very sophisticated software.

For the third parallel computing method, a pipeline is common paradigm for very high-speed computation. The pipeline parallelism allows for parallelization of a single task when there is a partial or total order in the dataset implying the need for state and therefore preventing the use of data parallelism. This approach is limited by the sequential decomposability of the task and the length of the longest stage.

In this paper, the new architecture uses a new parallel mechanism in combination with data-parallelism and pipeline-parallelism. It can automatically partition data and schedule them onto multi-cores through a pipeline without changing original single-processor program, instead of decomposing the entire program as conventional pipeline-parallel method does or scheduling as conventional data-parallel method needs.

B. Multiprocessor Pipeline by Program Slicing

Another promising parallel computing method is based on the multi-processor pipeline architecture by dividing the program in equal duration by forced interrupts as described in [2] [4] [5]. This technology has US PATENT No. 6145071 issued in 2000 and owned by The George Washington University. It can automatically schedule the program onto the multiple processors. The architecture processes an information flow progressively in a helicoidal pattern by relocating portions of incoming data. This pattern ensures that the incoming data flow will not be interrupted.

The multi-processor pipeline allows an arbitrary algorithm to be performed on-the-fly on a data chunk, given a sufficient number of processors. If an algorithm can be performed by a conventional microprocessor under static conditions, it can be performed on the multiprocessor pipeline. Another advantage of this architecture is to use practically the same software as a sequential computer and to be able to continuously process the intensive information flows.

This multi-processor pipeline is a simple and effective solution to the problem of continuous processing of intensive flows of information. This technique has been proposed to be used for effectively processing the challenging problem of very intensive continuous flows of data [3].

However, there are several limitations for some applications. Frequent data relocation especially for large data block applications like 3D graphics and image processing can cause big overhead costs leading to the overall performance decrease [16]. The system described in [4] uses data overlapping to solve the problem of processed data chunks across the segmentation imposed by the buffer size. The amounts of memories of the processors have to be occupied by these duplicate (i.e. overlapped) data. Moreover, to reduce the bus traffic, the data relocations, i.e. the loadings/unloadings, are arranged so that only one data chunk relocates in one shared bus at a particular time. Additionally, for the data required longer pipeline to process, the pipeline need special handling, such as overflow facility or accumulation and then sending back while the data stream ceases.

In this paper, the core difference between the new multiprocessor pipeline and original one [4] [5] is that the new pipeline is driven by crossbar switching instead of forced interrupts. Hence, the novel crossbar-switching based multiprocessor architecture with a new memory/cache management technology significantly overcomes all the above limitations of the original multiprocessor pipeline.

C. Multi-core Architectures and Programming

Currently multi-core processor architectures are divided into two basic categories: generic multi-core CPUs and Graphics Processor Units (GPUs). The Intel [11] and AMD [1] provide a large number of multi-core CPUs in the market. Both of them released dual-core chips in 2005 and quad-core chips in 2007. However, GPUs were originally designed with special purpose for 3D graphics applications. The hardware GPU architecture differs from multi-core CPUs significantly. The latest Intel's multi-core graphics chip, also known as Larrabee [15], is one of the boldest graphics projects in the world and offers full compatibility with graphics APIs. as well as is capable to process the entire x86 instruction set that comes implemented into modern processor architectures.

Currently, there are several major multi-core programming development platforms. RapidMind [14]'s Multicore Development Platform supports multiple processor architectures, including NVidia's GPUs, ATI's GPUs, IBM's Cell BE, and Intel's and AMD's x86. CUDA (Compute Unified Device Architecture) [7, 8] is a software platform for massively parallel high-performance computing on the company's powerful GPUs. It does require programmers to exert some manual effort and write some explicit code. OpenCL (Open Computing Language) [13] is an open industry standard for general-purpose parallel programming of heterogeneous systems.

However, all of the current multi-core architectures need multi-threading based parallel programming. In this paper, a completely different way is proposed to design multi-core CPU architectures without conventional parallel programming.

D. Crossbar-Based Technologies

The crossbar switch [6] has been used in many areas like telephone exchange. As computer technologies have improved, crossbar switches have found uses in systems such as the multistage interconnection networks that connect the various processing units in a Uniform Memory Access (UMA) parallel processor to the array of memory elements. Crossbar switches have been also designed to line entire computer systems as well.

Crossbar-based memory architecture has been used on mainframe computers to increase memory bandwidth in multi-processor systems since decades. The companies such as Unisys, SGI, and Sun, have brought the technology down to the server and workstation platforms.

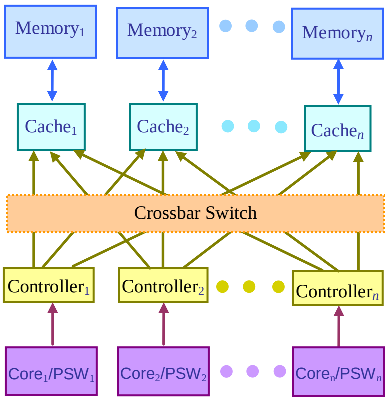
NVidia patented Light-speed Memory Architecture (LMA) [12] has employed the use of a crossbar to maximize the efficiency of data transfer between the graphics processing unit and the graphics memory on the GPU. A memory crossbar can eliminate bottlenecks associated with existing memory architecture as it replaces the conventional system bus architecture. Instead of sharing a bus, communication between the processor and the memory uses dedicated connections.

In this paper, a distinct dynamic memory/cache management technology using crossbar techniques is proposed for the new multi-core architectures.

III. A NEW PARALLEL ARCHITECTURE

A. New Multi-Core Memory and Cache Architecture Based on Crossbar Switching

The memory and cache management is very important for multi-core systems. A novel memory/cache management technology described in this section is one of core parts of this new architecture design. It significantly improves total performance in both space and time. Therefore, this part is introduced first.



**PSW: Program Status Words*

Fig. 1 Crossbar switch based Multi-core memory and cache CPU architecture

Since data blocks could be very large in the parallel applications, if DMA (Direct Memory Access) is used to move big data blocks from the memory in the previous processor to the next processor frequently at each time, the accumulated overhead costs cannot be ignored. To reduce the data relocation costs and bus limitations, the new architecture use a crossbar switch based dynamical management technique for both memories and caches to avoid relocating the data down the pipeline from one processor to another processor. The new memory and cache architecture for the technique is illustrated in Fig. 1.

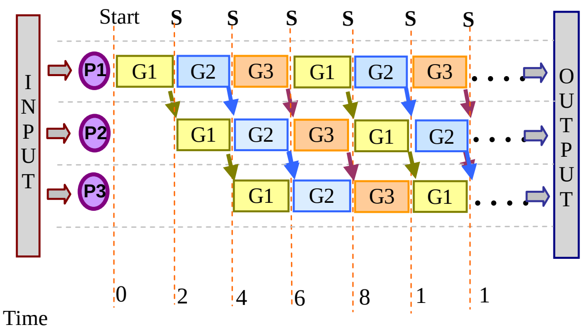
Each processor/core does not have a fixed memory and cache as does an ordinary processor. Instead, each of them will be assigned to connect to a given memory and cache at a given time. There is only one bus between such memory and cache. Both of them can be regarded as a group. The number of the groups is the same as the number of cores. Each core has its own controller, which is employed to switch the entire memory and cache, i.e. one group, for the previous processor into the next processor. The previous processor also passes the Program Status Words (PSW) to the next processor to resume operations where the previous processor stopped. Once the group of previous processors is migrated to the next processor, the current group of the previous processors needs to be cleared for use.

A crossbar switch is the key to carrying out this dynamic memory/cache management technology. It moves two memory and cache groups between two processors at a given time. Correspondingly, the bus crossbar is used for connecting all the cores to all the groups to guarantee data transmission between them at full speed and with no contention. Although the crossbar used for this architecture is similar to NVidia's LMA, they are different in principle. The purpose of the crossbar in our architecture is mainly for memory and cache switch apart from high-bandwidth data transmission. Furthermore, from the view of cores, the "N→N" connection relationship is for all cores and groups. This architecture differs from NVidia's LMA, in which one core has multiple memory controllers connecting to their corresponding memory banks. Actually, from the view of cores, the "1→N" connection relationship relates a given core to all its memory banks.

Additionally, the bus crossbar used in the new architecture can also reduce large bus traffic because each memory and cache group has a dedicated connection to one processor. This not only reduces the bus traffic, but also eliminates the constraint that the data movement has to be arranged at a particular time as mentioned before.

B. New Multi-Core Pipeline Organization

The new multi-core pipeline architecture does not relocate the data down the pipeline as would be the case for the original multi-core pipeline when a switch is applied. Instead, it only switches the memory and cache group of the previous processor, where the data have been loaded and/or processed, into the group of the next processor. These data also contain the small amount of the data of the operating state, such as Program Status Word (PSW) and all registers with the program counter, so that the next processor could resume operations where the previous processor stopped.



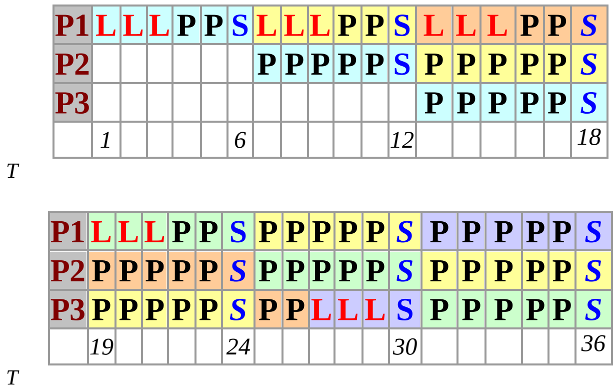
G = Group (memory and cache)
S = Switching (on/off for the connection between each processor and each group)

Fig. 2 New multi-core pipeline organization

The new multi-processor pipeline organization and data flow are depicted in Fig. 2. The major improvement is to replace Unloading (U) with Switching (S) in the organization. However, Loading (L), Processing (P), and Switching (S) do NOT rotate in the columns in the cycle: L →P→S→L in a helicoidal pattern as does the above original pipeline. Instead, the Switching (S) for all the processors occurs at the same time in the new multi-processor pipeline. Accordingly, Loading (L), Processing (P), and Switching (S) may rotate in the columns in the cycle: L→P→S, P→P→S, or P→L→S as shown in Fig. 3.

In Fig. 2, the vertical orange dashed lines show the cycles. Each small rectangle represents one operation in one cycle. All blocks and arrows with the same colors describe the flows of the data chunks. P1, P2, and P3 describe the processors or cores. In theory, there could be any number of processors, depending on the applications and user requirements.

Because the architecture does not specify a particular processor to perform a particular operation, data-dependent branching of the algorithm does not require special handling. A processor working on a particular data chunk behaves just as a standard processor, resulting in variable-length processing times for the data chunks. If a chunk becomes fully processed before the end of the pipeline, the result can be withdrawn. Data which require longer processing can be switched back to connect the first processor to continue processing. This is totally different from the original multiprocessor pipeline [4], which needs to send these data to some sort of overflow processing facility, or accumulate them and send them back through the pipeline when the incoming data stream ceases. This new method would be advantageous for highly variable data processing times. The next section also gives more details to explain this using an example in Fig. 3.



P1	L	L	L	P	P	S	L	L	L	P	P	S	L	L	L	P	P	S
P2							P	P	P	P	P	S	P	P	P	P	P	S
P3													P	P	P	P	P	S
	1					6						12						18

T

P1	L	L	L	P	P	S	P	P	P	P	P	S	P	P	P	P	P	S
P2	P	P	P	P	P	S	P	P	P	P	P	S	P	P	P	P	P	S
P3	P	P	P	P	P	S	P	P	L	L	L	S	P	P	P	P	P	S
	19					24						30						36

T

L = Load (a new data chunk)
P = Process
S = Switch (on/off)

Fig. 3 Timing diagram of a 3-core pipeline

C. Timing Diagram

Fig. 3 shows the timing diagram of an example of a 3-core pipeline. It is assumed that Switching (S) takes one cycle. In fact, S may take more than one cycle or less than one cycle, which depend on hardware and software. Because of the dynamic memory/cache management with the crossbar-based techniques, all the Switchings (S) occur at the same time. That is, all the processed data from previous processors to next processors are switched at the same time.

In the Fig. 3, each color block represents a data chunk being processed in different processors in different time. The same color blocks indicate the data flow for one chunk. A new data chunk can be loaded (L) by any one of the processors as long as the previous data chunk is finished processing. In other word, any one of the processors can load the data as long as it is free. The various lengths or processing time of the data chunks can thus be automatically scheduled onto the processors without considering any load balancing or scheduling issues.

In fact, one data chunk is always being processed within this memory and cache group although the core connected to this group is changed at every switching time. That is to say, the data do not move while the core or processor moves. This is the major difference from the original multiprocessor pipeline, in which the data chunk has to be relocated when a switch is applied.

The new architecture is much more efficient in both space and time. The overall performance can be improved significantly. This is totally different from the multiprocessor pipeline system in [4, 5], which allows only one data chunk relocation at a particular time on one shared bus.

D. Program Requirements

An important feature of this architecture is that it uses practically the same software as a sequential computer. A program for this system can be developed on an ordinary sequential computer. Each processor is distributed with the same application program. To run on the system, the program would just have to incorporate some interrupts. The interrupt is triggered by the crossbar switching.

A processor working on a particular piece of data, upon crossbar switching, will move its memory to the next processor. The processor will also pass the PSW so that the next processor could resume operations where the previous processor stopped. The formation of the PSW in the system is similar to the routine procedure of formatting the PSW for interrupts in ordinary microprocessors.

IV. PERFORMANCE AND EXPERIMENTAL ANALYSIS

A. Performance

Clearly, the time complexity of the algorithm based on the new multi-core pipelined architecture will only affect the total length of the pipeline including the overhead of the memory and cache group switches between processors after switching.

Let me analyse how this new multi-core pipelined architecture improves the overall performance for parallel computing. The principle of the N-core pipeline is descried in Fig. 4.

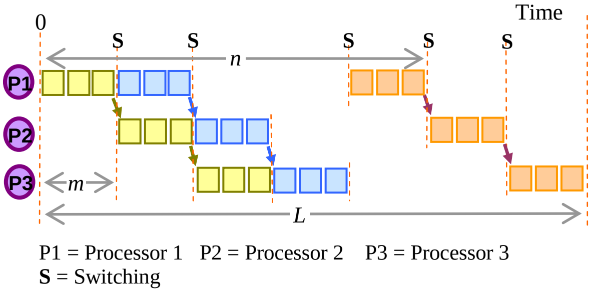


Fig. 4 The principle of the *N*-core pipeline

For the multi-core pipeline system as shown in Fig. 2, to better explain the performance, the pattern of the system operations is assumed to be discretized into cycles with each cycle, C. In fact, it can be discretized into much smaller time unit in modern architectures so that the architectures can be programmable.

Let us consider a program to process a dataset containing data chunks with equal or various sizes. Processing the entire dataset requires K cycles. On a conventional processor, the execution time of a program, T_c, can be expressed by

$$T_c = KC \tag{1}$$

In Fig. 4, let *N* be the number of processors/cores. If the entire dataset can be processed on all *N* cores, on the average, the number of cycles on each processor/core, *n*, can be given by

$$n = \frac{K}{N} \tag{2}$$

Thus, the total time spent on the conventional processor T_c can also be described in combination with above two equations as

$$T_c = nNC \tag{3}$$

For the N-core pipeline, let *m* be the number of cycles for each interval of the equal duration in the pipeline. Thus, the length of the *N*-core pipeline for processing of data, L_p, can be described by

$$L_p = n + (N-1)m \tag{4}$$

Let *q* be the number of the internals for the entire pipeline, it can be

$$\begin{aligned} q &= \frac{L_p}{m} \\ &= \left\lceil \frac{n}{m} \right\rceil + (N-1) \end{aligned} \tag{5}$$

Since switching by crossbar may take some extra time, let *d* be the number of cycles of processing and delays or latencies for each switching based on hardware. The total number of cycles of the overhead of the context switching in the *N*-core pipeline, can be expressed as

$$\begin{aligned} H &= (q-1)d \\ &= (\left\lceil \frac{n}{m} \right\rceil + N - 2)d \end{aligned} \tag{6}$$

Thus, in the *N*-core pipeline system, the total length of the pipeline, noted by *L*, to process the same size of the data including the overhead of content switching, can be described by

$$\begin{aligned} L &= L_p + H \\ &= n + (N-1)m + (\left\lceil \frac{n}{m} \right\rceil + N - 2)d \end{aligned} \tag{7}$$

The total time spent *T* on this multi-core pipelined system can be obtained by

$$T = LC \tag{8}$$

Let's first calculate the speed-up of the execution time on the new *N*-core pipeline system compared to the time on one conventional processor.

Combined with Equation (1), Equation (3), Equation (7), and Equation (8), the speed-up is given by

$$\begin{aligned} Speedup &= \frac{T_c}{T} \\ &= \frac{nN}{n + (N-1)m + (\left\lceil \frac{n}{m} \right\rceil + N - 2)d} \end{aligned} \tag{9}$$

B. Simulation Analysis

We developed a tool to evaluate this new architecture. In this section, some experimental results are reported based on this architecture with various overheads taken into account.

In the first four experiments as shown in Fig. 5, Fig. 6, Fig. 7, and Fig. 8, we assume the total execution of a given sequential program takes K = 10,000 cycles and one-time switching overhead *d* takes 2 cycles.

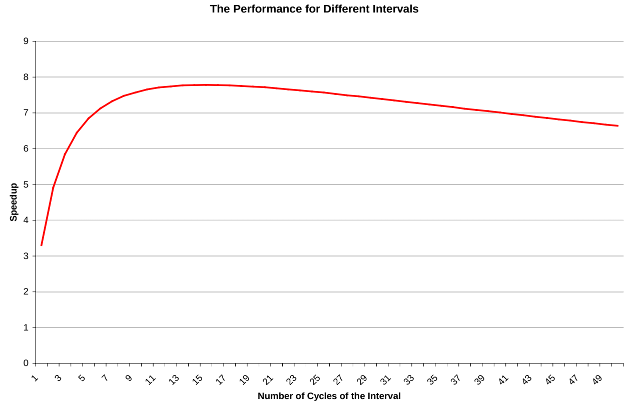


Fig. 5 The performance for different Intervals

Fig. 5 shows the performance increases with interval time *m* increasing for fixed number of cores, *N (N=10)*, and switching time, *d*, and reaches the maximum then decreases slowly. This is because the pipeline will become longer if *m* becomes bigger. The performance is affected by the length of the pipeline.

Fig. 6 shows the performance increases with number of cores increasing for fixed interval time, *m (m=2)*, and switching time, *d*, and reaches the maximum, then decreases slowly. This is also because the pipeline may increase slowly while the number of cores becomes big. That is to say, for a given program with fixed interval time, *m*, and switching time, *d*, the performance may not reach the maximum for some numbers of cores. Put another way, if given an appropriate interval time *m* and switching time, *d*, the performance can reach the maximum speedup. In practice, since *d* is usually fixed for such a multi-core system due to the crossbar switch, the maximum *speedup* can be obtained with an appropriate *m*.

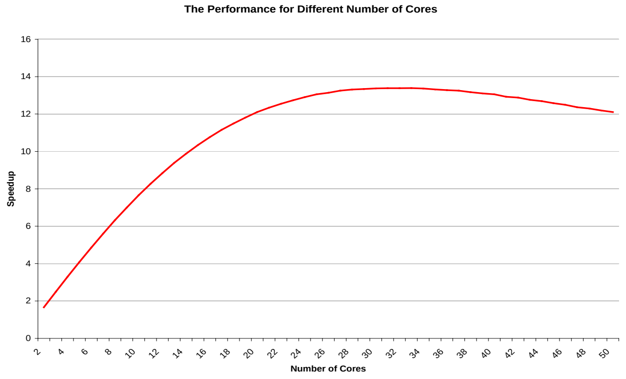


Fig. 6 The performance for different number of cores

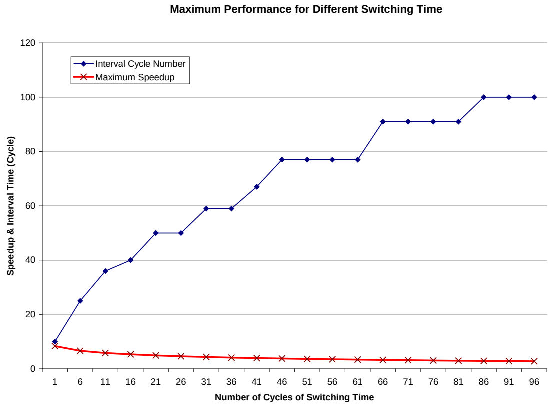


Fig. 7 Maximum performance for different switching time

Fig. 7 shows the maximum performance for different switching time, *d*. In the figure, the blue line shows the interval time, *m*, and the red line shows the maximum performance, *max_Speedup*. For a given program and fixed number of cores, *N (N=10)*, with the increase of the switching time, *d*, the best interval time, m_b, to reach the maximum performance, *max_Speedup*, increases when the switching time, *d*, increases. However, the *max_Speedup* decreases slowly accordingly. This clearly indicates the pipeline length increases with the increase of both interval time and overhead of content switching (i.e. *m* and *d*). Accordingly, the performance decreases.

Fig. 8 shows the performance *Speedup* increases with increasing the number of cores and the size of the data. Note that the *Speedup* is the maximum, *max_Speedup*, in this experiment. The figure clearly indicates when the application program becomes larger, the *Speedup* increases linearly with the number of cores. That is, the *Speedup* is almost close to the number of cores for the large-size applications. This indicates the performance increases the times of number of cores, which is the maximum performance for any multi-core system in theory.

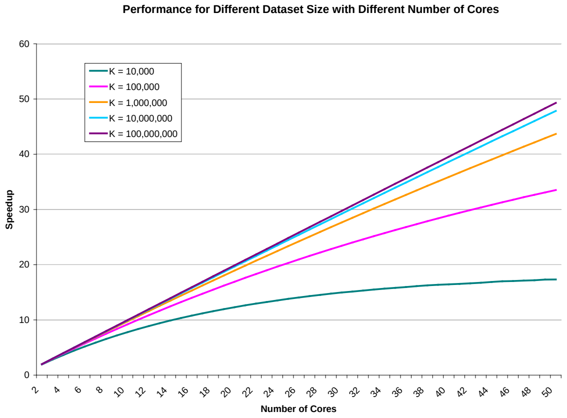


Fig. 8 The performance for different dataset size with different number of cores

Fig. 9 shows the best interval time, m_b, to reach the maximum performance decreases and then become stable with the increase of the number of cores. Furthermore, all the best interval time, m_b, for different size of the application programs is close to each other when the number of cores increases. This implies that the most appropriate interval time, *m*, can be chosen to maximize the performance for all the application programs on a multi-core system with the certain amount of cores. This is a trade-off for a multi-core system to ensure the best performance for all the applications.

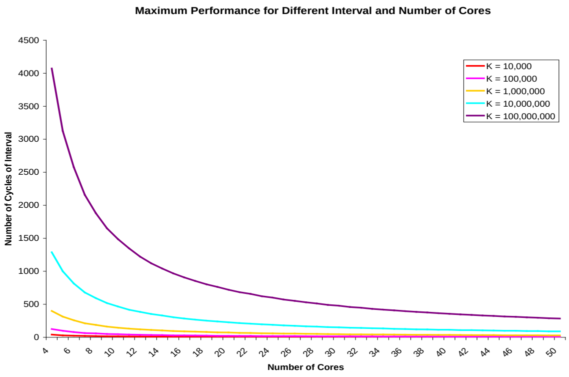
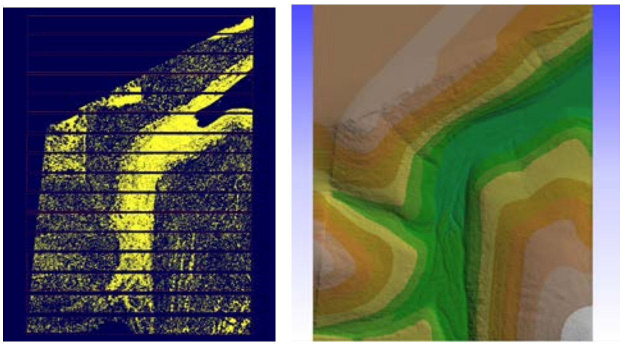


Fig. 9 Maximum performance for different intervals and number of cores

In summary, with the increase of the number of cores, the performance increases totally. However, choosing the best interval time, m_b, is the key to make full use of all the cores to maximize the performance for all the application programs. Such a possible m_b can be found for all the application programs based on the analysis of Fig. 9. In addition to selecting one fixed best interval time, m_b, for the entire system, the dynamical best interval time, m_b, can also be automatically assigned to each application program according to the total application and data size, the number of cores, and the switching time on a multi-core system while the program is compiled.

C. *Experimental Results and Performance Comparison*

We developed a function simulator to evaluate this multi-core pipelined parallel system and compare it with existing multi-core systems. The testing multi-core programs are based on the algorithms of DEM generation from LIDAR dataset designed and developed by me before.

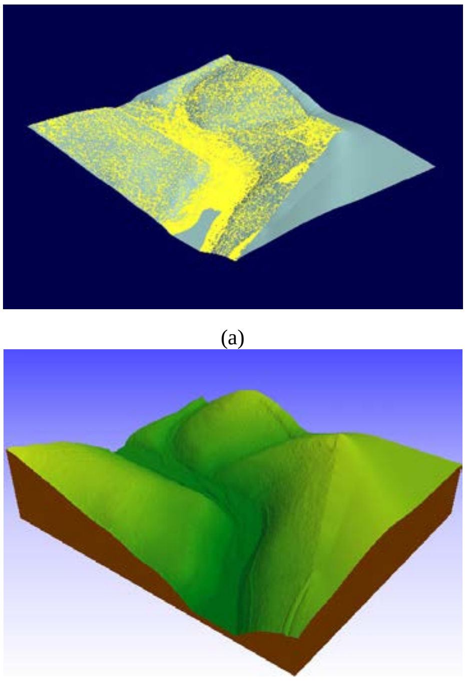


(a) (b)

Fig. 10 16 stripes of the LIDAR data

The tests were run on a Dell PC with Pentium (R) D a single processor 3.0GHZ and 2GB RAM equipped with a GPU, NVidia GeForce GTX 260 featured with 192 CUDA cores.

The LIDAR dataset contains 214,665 points in (x, y, z, value). In order to test it on the multi-core architectures, it is divided into 16 stripes along Y-direction as shown in Fig. 10 (a). The DEM generation algorithm is employed for each LIDAR stripe. The size of each DEM stripe is 534 by 40. The experimental results are reported in the Figs. 10 - 13.



(a)

(b)

Fig. 11 (a) LIDAR matches the generated DEM in 3D space; (b) Generated DEM rendering in color ramps

Fig. 10 (b) shows the corresponding rendering effects of the combined DEM with 16 pieces. Fig. 11 (a) shows the generated DEM matches the original LIDAR data very well in 3D space. The 3D terrain based on the DEM is rendered in color ramping as indicated in Fig. 11 (b).

Fig. 12 and Fig. 13 show the experimental results on the single-processor system using the algorithms to generate a DEM based on LIDAR data. Fig. 12 indicates the time spent on the DEM generation for all LIDAR stripes from 1 to 16. Obviously, the processing time for each LIDAR stripe is different.

Fig. 13 shows the comparison of the new multi-core pipelined GPUs and existing multi-core GPUs to generate the same DEM from the 16 LIDAR stripes with the size of 534x640. It obviously indicates the performance of new multi-core GPU architecture is better than existing ones. This is because data partition and load balancing and scheduling need be considered for existing multi-core GPU system. Moreover, these conventional parallel methods cannot be done easily on current multi-core systems. The performance may be affected by different data partition or load balancing and scheduling methods. However, the new multi-core system can directly use original sequential program for parallel computing. Thus, it does not need to take into account the data distribution and load balancing and scheduling issues. All of the data can be automatically partitioned and scheduled onto the different cores through the pipeline. Furthermore, with the increase of the number of cores, the performance of the new architecture increases much more than the existing ones with the same number of cores.

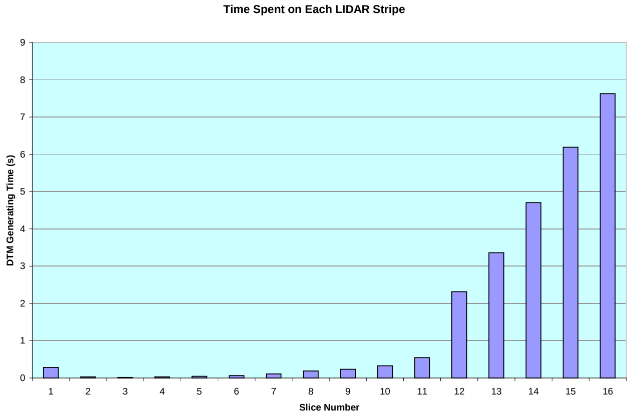


Fig. 12 Time spent on each LIDAR stripe

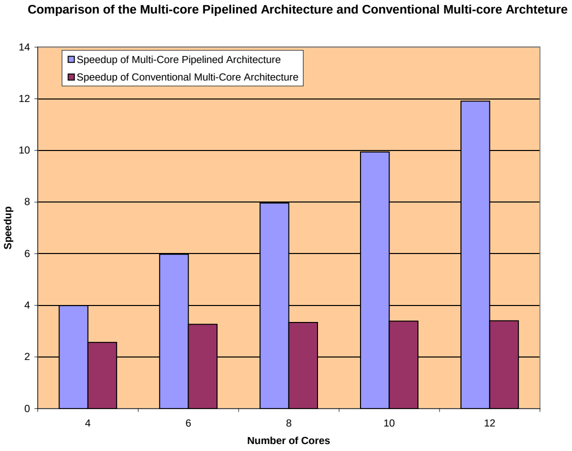


Fig. 13 Comparison of the new multi-core pipelined architecture and existing multi-core architectures

V. CONCLUSIONS AND FUTURE WORK

This paper presents a new multi-core pipelined architecture based on core crossbar switching driven multiprocessor pipelining and dynamic memory and cache management techniques. The new architecture is very suitable for processing large data in parallel without parallel programming in geospatial domain as well as other high performance computing areas. The proposed architecture provides a simple and effective implementation for on-the-fly parallel computing by switching the entire data from core to core through the crossbar switch. This architecture makes full use of the pipeline. It can automatically partition data and schedule them onto multi-cores through the pipeline. It does not need conventional complicated parallel computing methods, such as load balancing, scheduling, and data distribution. This is exactly the advantage of this proposed architecture.

Obviously, the new multi-core pipeline architecture significantly overcomes all these limitations of the original multiprocessor pipeline as described in Section 2.2. Especially, the core difference of these two multiprocessor architectures is the new architecture relocates the cores instead of moving the data in the original one. More specifically, the new pipeline advantages over the original pipeline limitations are summarized as follows:

- No data relocation

  Content switching is employed to minimize big overhead costs due to data relocation while a switch is employed.

- No data overlapping

  Due to content switching, no processed data chunks across the segmentation.

- All the Switches (S) are forced by the crossbar switching control at a regular time

  All the data are switched to their corresponding processors. However, the original multiprocessor pipeline only allows one data relocation at a particular time due to one shared bus.

- No special handling for the data required longer pipeline to process

  The new architecture does not need special handing for the data required longer pipeline to process. But, the original multiprocessor pipeline need overflow facility or accumulation and sending back while the data stream ceases.

Additionally, more specific advantages for the new switch-based dynamic memory and cache-management technology in the new architecture are emphasized as follows:

- avoids moving the data from one memory to another memory
- allows more than one Switching operation at a time; all the Switchings occur at the same time
- reduces bus limitations and large bus traffic using the bus crossbar
- improves performance in both space and time

Finally, to summarize, there are several overall advantages of this new multi-core pipelined architecture as follows:

- provides the continuous data processing of intensive information flows
- requires essentially the same software as ordinary sequential algorithm
- avoids load balancing and scheduling
- avoids the need for synchronization among the processors
- avoids busy waiting of processors on a spin-lock
- avoids the duplication for incoming data stream
- be suitable for highly variable data processing time

Another advantage worth mentioning is that this core multiprocessor/multi-core pipelining technology provides an important solution to processing the continuous intensive information flows without limitation by size. Consequently, this would be very helpful to real-time massive data processing, especially geospatial computing.

REFERENCES

[1] AMD Corporation, White Paper: AMD Multi-core Processors. AMD Corporation. 2006.

[2] S. Berkovich, Z. Kitov, A. Meltzer: On-the-fly processing of continuous data streams with a pipeline of microprocessors. In Proceedings of the International Conference on Databases, Parallel Architectures, and Their Applications (PARBASE-90), IEEE Computer Society, Maiami Beach, Florida, March 1990, pp. 85-97.

[3] S. Berkovich, M. Loew, and M. Zaghloul: On-Line Processing and Archiving of Continous Data Flows. In IEEE Proceedings of 35th Midwest Symposium on Circuits and Systems. Washington DC, Aug. 1992, pp. 777-779.

[4] E. Berkovich, S. Berkovich, M. Loew: A Multi-Layer Conveyor for Processing Intensive Information Flows. The Technical Report, GWU-IIST-94-13, The George Washington University, 1994.

[5] S. Berkovich, E. Berkovich, and M. Loew, 2000. “Multi-Layer Multi-Processor Information Conveyor with Periodic Transferring of Processor’s States for On-The-Fly Transformation of Continuous Information Flows and Operating Method Therefor”, US PATENT No. 6145071, owned by George Washington University. Date issued - November 7, 2000.

[6] Crossbar Switch on Wikipedia. http://en.wikipedia.org/wiki/Crossbar_switch. 2008.

[7] NVidia. NVIDIA CUDA Compute Unified Device Architecture Programming Guide (Version 2.1 Beta), Oct. 2008.

[8] NVidia. NVIDIA CUDA Compute Unified Device Architecture Reference Manual (Version 2.1 Beta), Nov. 2008.

[9] D. Culler, J.P. Singh, Anoop Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann, © 1998. ISBN 1-55860-343-3.

[10] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar,

An Introduction to Parallel Computing, Design and Analysis of Algorithms: 2/e, Addison-Wesley, © 2003. ISBN 0-201-64865-2.

[11] Intel Corporation, White Paper: Intel® Multi-Core Processor Architecture Development Backgrounder. Intel Corporation. 2006.

[12] NVidia Corporation, Technical Brief: GeForce3: Lightspeed Memory Architecture. Nvidia Corporation. 2001.

[13] Aaftab Munshi, The OpenCL Specification (Version 1.0). Khronos OpenCL Working Group. Dec. 2008.

[14] RapidMind. Easily build applications for multi-core. http://www.rapidmind.net/product.php. 2008.

[15] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., and Hanrahan, P. 2008. Larrabee: a many-core x86 architecture for visual computing. In ACM SIGGRAPH 2008 Papers (Los Angeles, California, August 11 - 15, 2008). SIGGRAPH '08. ACM, New York, NY, 1-15.

[16] Stompel, A., Ma, K., Lum, E.B., Ahrens, J., and Patchett, J. 2003. SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering. In Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics (October 20 - 21, 2003). Parallel and large-data visualization and graphics. IEEE Computer Society, Washington, DC, 6.

Duoduo Liao earned a Ph.D. and a M.S. in Computer Science from George Washington University and Purdue University in USA, respectively. Since 2002, she has worked for the federal government agencies and universities on PC-clustered highway driving simulator systems, 3D graphics & visualization, virtual reality, GIS, traffic simulation, air traffic management, multi-core architectures, heterogeneous computing, etc. She ever worked at ESRI and developed the first version of Stereo Viewer for ArcGIS in 2001. In 1996, she pioneered the product development of the PC-based high-resolution quad-buffered 3D stereographic accelerators using the earliest PC graphics chips invented by 3D Labs. Dr. Liao has authorized more than 80 technical publications and two professional books on GPU-based research and OpenGL programming. She has been invited to give the talks by the federal governments, leading industries, and universities. She was an adjunct professor at George Mason University. She is a member of ACM and IEEE, and serves the conference chairs, editorial boards, and committees of several international conferences.

Simon Y. Berkovich earned a M.S. in Applied Physics from Moscow Physical-Technical Institute and a Ph.D. in Computer Science from the Institute of Precision Mechanics and Computer Technology of the USSR Academy of Sciences. He is a Professor of School of Engineering and Applied Science at George Washington University. Prof. Berkovich played a leading role in a number of research and development projects on the design of advanced hardware and software systems. Those projects include construction of superconductive associative memory, development of large information systems for economics, investigation of computer communications for multiprocessor systems, and enhancement of information retrieval procedures. Prof. Berkovich has several hundred professional publications in various areas of physics, electronics, computer science, and biological cybernetics. He is an author of six books and holds 30 patents. Among his inventions is a method for dynamic file construction that later become known as B-tree and extendible hashing. In 2002, he was elected a member of the European Academy of Sciences “for an outstanding contribution to computer science and the development of fundamental computational algorithms”.