British Journal of Mathematics & Computer Science
7(1): 40-57, 2015, Article no.BJMCS.2015.101
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

SCIENCEDOMAIN

Overview of Trends Leading to Parallel Computing and Parallel Programming

Charles I. Saidu[1*], A. A. Obiniyi[2] and Peter O. Ogedebe[1]

[1]*Faculty of Science and Technology, Computer Science Department, Bingham University, Karu, Nasarawa State, Nigeria.*
[2]*Faculty of Science, Ahmadu Bello University, Zaria, Nigeria.*

Article Information

DOI: 10.9734/BJMCS/2015/14743
Editor(s):
(1) Ke-Lin Du, Department of Electrical and Computer Engineering, Concordia University, Canada.
(2) Dariusz Jacek Jakóbczak, Computer Science and Management Department, Technical University of Koszalin, Poland.
(3) Tian-Xiao He, Department of Mathematics and Computer Science, Illinois Wesleyan University, USA.
Reviewers:
(1) S. V. Saboji, Computer Science and Engineering, Visveswaraza Technological University, India.
(2) Anonymous, Italy.
(3) Anonymous, USA.
Complete Peer review History: http://www.sciencedomain.org/review-history.php?iid=932&id=6&aid=7976

Review Article

Received: 17 October 2014
Accepted: 19 December 2014
Published: 31 January 2015

Abstract

Since the development and success of the first computer built from transistors in 1955, the quest for faster computers and computations have brought about various technological advancement in the way computers are built and the techniques involved in programming these computers. These trends have seen a radical change in speed, size and power consumption of these computers. A general observation can be seen from the sequential hardware/programs of the early 80s and 90s to the parallel hardware/programs of the present day 21th century computers. In all, there has been a tremendous improvement in computational speedup, size and energy consumed by these computers. This paper discusses these technological innovations/advancements in computer hardware and architecture leading to parallel computers, starting from the first sequential computer to the parallel computers available today with a focus on the key issues that led to the shift in the ways these computers were built, the limitations and future developments. It also discusses the programming models adopted for these parallel computers and how their limitations and gains contribute to a shift in ideology and higher speedup of computations.

Keywords: Parallel architecture; parallel programming; parallel computing; programming models and trends.

*Corresponding author: charlessaidu@binghamuni.edu.ng;

1 Introduction

Parallel computing, also known as concurrent computing, refers to a group of independent processors working collaboratively to solve a large computational problem [1]. According to [1], parallel computing is motivated by the need to reduce the execution time and to utilize larger memory/storage resources. The essence of parallel computing is to partition and distribute the entire computational work among the involved processors.

In parallel computing, mechanism are provided for explicit specification of the portion of the program to execute in parallel and depending on the computer, scheduling of the computational work can be done by the programmer, at runtime or at compile time.

The speed of a computer is a function of the speed of processing unit, speed of memory, architecture of the computer and the way programs are written to run on the computer. The speed of the processing unit is a function of several factors ranging from the rate at which instructions are executed per processor cycle (known as Instruction per cycle IPC), the frequency of the processing unit and processing unit features like caches and overall design which in some context can be referred to as the architecture.

Over the years there has been a shift in the way computers are being built from single processor computers to parallel processor computers [2]. This shift has been majorly powered by the quest for high speed computers, and has seen tremendous advancements on the architecture of computers and also on the way these computers are programmed. The need for high performance computation led to the birth of parallel computers. These computers can be built from special high speed microprocessors to everyday personal computers. Parallel computers can be categorized as follows;

a. Symmetric Multiprocessor Parallel Computers: Computers involving multiple processing units/processors all sharing a common memory,
b. Multicore Parallel Computers: Parallel computers involving a processing unit with multiple cores sharing a common memory,
c. Distributed Parallel Computers: Parallel computers consisting of multiple processing unit with each unit having its own memory,
d. Cluster Parallel Computers: Cluster parallel computers are a group of loosely coupled, not necessarily symmetric computing set that collectively work together as a group to solve a compute task. These computers are connected via a commodity hardware network.
e. Massively Parallel Computers: These types of parallel computers are similar to Cluster parallel computers with the exception that its multiple processors are interconnected via a specialized high speed networks. Massively parallel computers are usually a collection of more number of processors than the cluster parallel computers,
f. Grid Computers: Grid computers are a special kind of parallel computers whose collection of computers are interconnected via the internet. Hence, issues of bandwidth and latency play a great role in limiting speedup of computation.

Another way to classify parallel computers is through processor-memory design architecture. There are Shared Memory Architecture (SMA) Parallel computers and Distributed Memory Architecture (DMA) Parallel Computers. In SMAs, processing nodes access a common globally shared memory. These processing nodes can either consist of multiple individual processors or a single processor with multiple processing units known as cores. A typical model is shown in Fig. 1 where each processing unit (CPU and Cache) access a shared main memory as depicted in the image. Examples are the Symmetric multiprocessor parallel computers and multicore parallel computers mentioned above.

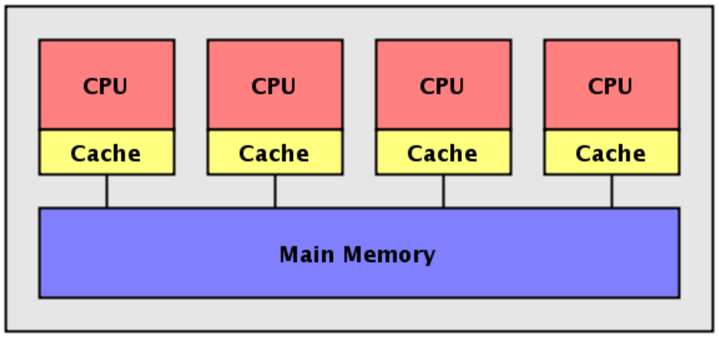

Fig. 1. Shared memory architecture

In DMAs, each processing node has its own private memory and communication among processing nodes is usually accomplish via message passing. Fig. 2 shows a typical model view of a distributed memory parallel computer. Distributed Parallel Computers, Cluster Parallel Computers and Massively Parallel computers can be grouped under this category.

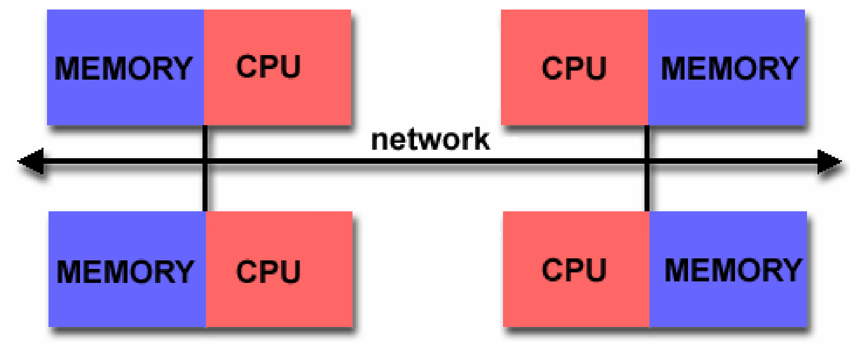

Fig. 2. Distributed memory architecture

Parallel computers and parallel programming is a much more recent approach to achieving faster computations. Early computers systems achieved faster computers by the optimization of the architecture of single processor computers, through increasing the number of transistors per processing unit and improving the instruction set of these computers. Limitations in this trend led computer manufacturers to look into the area of parallel execution of programs and parallel computers involving multiple processors and multicore processors [3,4].

As advancement in the computer design continued so also the evolution and birth of newer ways of programming these computers. Each computer design gave birth to programming models and several programming languages and parallel libraries sprang up.

This paper discusses the trends in single processor computers, parallel computers and programming models. Therefore, in this paper we will be taking a survey into areas bothering around:

- Developments and trends in single processor computers.
- Developments and trends in parallel computers.

- Developments and trends in programming models and programming languages for parallel computers.

2 Trends in Single Processor Computers

The speed of computers is generally determined by the speed of the processor and memory [5]. The speed of the processor in turn is determine by several factors which include the number of operands and operators processed by cycle (Instruction set Architecture) bit-word, caches and registers. Early computers focused more on reducing power and heat dissipated by computers and this trend is still ongoing. The first transistor computer (known as TRADIC) designed by Jean H. Felker and James R. Harrisin 1955 at AT & T Bell Laboratories contained nearly 700 transistors instead of vacuum tubes that contained 10, 000 (Fig. 3). The vacuum tube operated at 1 MHz and required 100 watts of power which was one-twentieth times less than the power required by a vacuum tube computer [3].

Also, several other transistor-computers like the TX-0 by William Papian and ETL Mark III by the Japanese were all built using transistors. While the TX-0 consumed a whooping 1000watts, the ETL Mark III required about 70 watts to operate. The birth of transistor computers also brought about a plethora of computer manufacturers like IBM, Digital Equipment Corporation (DEC), and Univac (now Unisys) [6].

Therefore, the most significant improvement on computers as at the early and late 1950s was in the size and the amount of power consumed. Compared to the ENIAC (Electronic Numerical Integrator and Computer) which consumed 150 kilowattes of power (McCartney & Scott 1999), the typical transistor based computer like the TRADIC consumed just 100 watts [6].

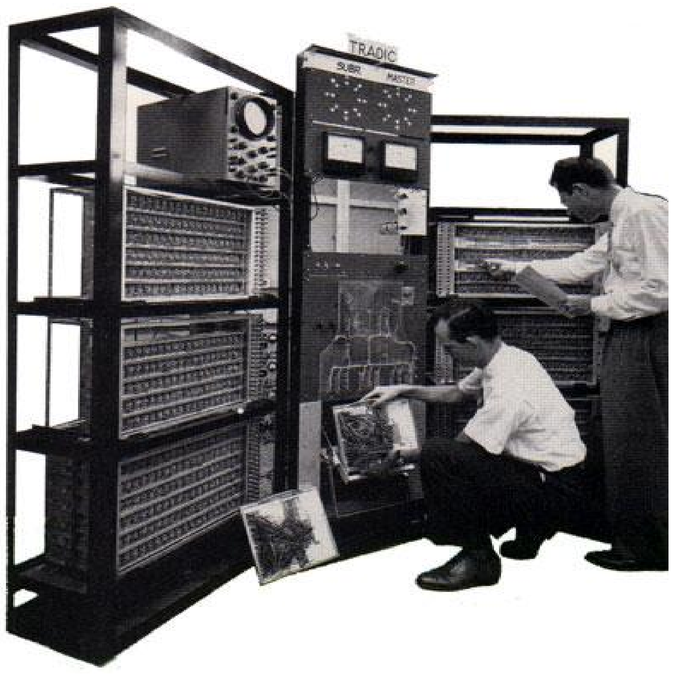

Fig. 3. TRADIC-transistorized computer
Image source: http://www.cedmagic.com/history/tradic-transistorized.html

Around the same time came the birth of transistors made from silicon as silicon devices can function from -55ºC to 125ºC, compared to the germanium transistors which can only operate at temperature between 0ºC to 70ºC, hence silicon made transistors became widely accepted because of its ruggedness [3,6]. The first successful silicon transistor was made in mid 1950s and after then silicon became the dominate semi-conductor material. This technology gave rise to what is now known as the integrated circuit which basically consists of several integrated silicon-made transistors. Silicon based computers made it possible to increase the number of transistors could be imbedded within a microchip and in effect smaller and better performing computers were produced.

3 Integrated Circuit Technology

The introduction of semi-conductors manufactured from silicon made remarkable improvement in the computer industry as more and more transistors could fit into a small area the size of a coin. More transistors translated to more processor functionalities as more logic gate could then be built and by extension the size of caches and registers could then be increased. The ultimate goal was to build faster and smaller computers. This was achieved either by increasing the functionality of the processing units through increasing transistors or improving the instruction set architecture of the computer.

The IBM System/360 family of computers was among the first commercially available systems to be built entirely from solid-state components i.e. transistors [1,7,21].

With the introduction of Integrated-circuit designed (an example of which is shown in Fig. 4) computers came the birth of time-sharing and multiprogramming. This led to the introduction of new operating systems for computers and sparked the first real steps towards program sophistication and advancement in technology bordering around efficiency and speed. Systems like the DEC’s PDP-8 and PDP-11 made computing affordable to smaller businesses and more universities [5,7,8].

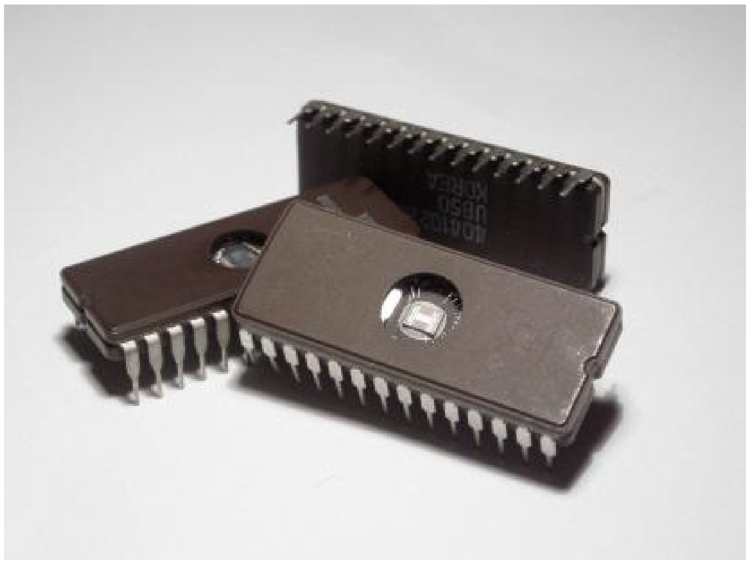

Fig. 4. Integrated circuit
Image source: http://upload.wikimedia.org/wikipedia/commons/5/5c/Microchips.jpg

The first step towards high performance computing was seen in the development of the Cray-1 super-computer by Cray Research Corporation. The Cray-1, could execute over 160 million instructions per second and could support 8 megabytes of memory [6,9,10,11].

The integrated circuit saw the use of process technologies for building transistors namely; the bipolar-junction transistors and the CMOS transistors. The advancement in process technology for manufacturing chips made possible the increase in the number of transistors per chip. Level of transistors per-chip integration spun from SSI(Small scale integration), in which there were 10 to 100 components per chip; LSI (medium scale integration), in which there are 100 to 1000 components per chip; LSI (Large scale integration), in which there are 1,000 to 10, 000 components per chip, VLSI(Very Large Scale Integration) with 10,000 to 5,000,000 transistors per chip and at the moment WSI(Water scale Integration) with over 5,000,000 transistors per chip [8,10,12]. Invariably, as the number of microchip integration increased so did the performance in computing power as more and more transistor count led to more and more registers, caches and features per processing unit. As the size of transistors became smaller, the current required to power the transistors reduced.

To further increase the speed of processing, a technique known as frequency scaling (a technique which involves increasing a processor's frequency in order to increase its performance) was used to scale performance of the microprocessor.

Also, the rate of increase in transistor count per chip led to the famous Moore's law. According to Moore's law, the transistor count in a chip doubles every two years [8,13,14]. This increase in transistor count per chip has led to two fundamental problems in micro-processor design; heat and energy leakage. Due to the miniature nature of microprocessors and integrated circuit design, frequency scaling technique of performance boost became not so viable because the power that would be required to power a small miniature integrated circuit would be so much that the heat generated would not be easily cooled.

There is also the problem of energy leakage which would render the state of the transistor unpredictable. As a result, Moore's Law is seeing a steady decline and some scientist are already predicting the death of Moore's law [15]. This in turn will mean no speedup as long as silicon made transistors and the current processing technology are still in use.

Another technological advancement that contributed to speedup of computers is the Instruction Set Architecture [5,8,10,12]. Early trends saw the transition from Complex Instruction Set Computers like the Intel Pentium, the Motorola MC6800 and the IBM & Macintosh PowerPC to Reduced Instruction Set Computers machines like Sun SPARC and MIPS machines. This shift was observed as a study showed that majority of assignment and conditional branch statements constituted most part computer programs [14,15,16]. Also, in order to reduce CPU cycles, the complexity of instructions was reduced in such a way that more instructions could be executed per cycle hence improving overall speed.

3.1 The Birth of Pipelining, Instruction Level Parallelism, Data Level Parallelism

The need to increase the number of instructions per cycle introduced concept known as Pipelining. According to [17], a pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements.

The increase in the transistor integration saw a decrease in gate delay by 1.5 times. However, to better utilization this increase in frequency, the number of instructions executed per cycle had to be optimized. This contributed to the shift to pipelining [16,17].

A typical microprocessor architecture is designed to divide the microprocessor's functionality into three major components [20].

a. *Instruction Supply:* Fetching instructions, decoding them, and preparing them for execution;
b. *Execution:* Choosing instructions for execution, performing actual computation, and writing results;
c. *Data Supply:* Fetching data from the memory hierarchy into the execution core.

Previous architecture would complete one operation before beginning another. Pipelining revolutionized instruction execution. Pipelining breaks the processing of an instruction into a sequence of operations called stages. The idea in pipelining is to overlap these computing stages, hence improving overall performance. A typical microprocessor with *n*-stages pipeline can deliver *n*-times performance over a non-pipelined architectures. Therefore, with the increase in process technology (Integrated circuit manufacturing technology), and the introduction of pipelined architecture, performance in terms of computational speed was achieved significantly.

However, the pipeline architecture came with its own bottlenecks, because the number of pipeline stages could not be increased indefinitely. [16] outlined some of these challenges as stated below

i. Scaling the frequency of the microprocessor while holding the number of pipeline stage constant can get to a point where there will be decrease performance. Hence, scaling frequency linearly with the number of stages requires good balancing of the overall work among the stages and this is difficult to achieve.
ii. Dependencies among instructions can require stalling certain pipe stages and result in wasted cycles, causing performance to scale less than linearly with the number of pipe stages. These dependencies could either be control dependencies or data dependencies. An instruction is control dependent if the outcome of its previous instruction is dependent of its execution. Also an instruction is data dependent if its evaluation is dependent of the availability of data from a previous instruction.

Conversely, the number of pipeline stages couldn't just be increase arbitrarily. This is because the deeper the pipeline the increase in the number of stalls. The main reason for these stalls is resource contention, data dependencies, memory delays and control dependences. A pipeline operation is said to have been stalled if one unit (stage) requires more time to perform its function, thus forcing other stages to become idle [17].

In order to reduce memory latencies, smaller and faster memories were designed (caches). And with process technology it was possible to integrate a cache memory within a single microprocessor hence reducing latency and in effect reducing the number of stalls in a typical pipelined microprocessor. It should also be observed that the introduction of caches in pipelines brought about the problem of cache misses (a request for data that has not been previously cache), which could be costly since instructions would then have to be re-fetched from the main-memory. The problem of cache misses were reduced as larger cache memories could then be built therefore reducing the possibility of cache misses. This trend saw the birth of level 1 and level 2 caches [17].

The next step towards performance enhancement was the introduction of the superscalar pipeline architecture. Unlike the scalar pipeline execution aforementioned, the superscalar pipeline execution has the ability to execute more than one instruction in the same pipe stage and at the same cycle. This can also be visualized as parallelism at the assembly language level. The idea behind the superscalar architecture was to replicate the functional unit (Arithmetic and Logic Unit, ALU) of the CPU. Each functional unit is not a separate CPU core but an execution resource within a single CPU such as ALU, a bit shifter, or a multiplier. Once, again these features were made possible with the process technology (VLSI) of integrating thousands to millions of transistors within an integrated circuit.

The superscalar architecture exhibited true parallelism at the instruction execution stage of computing process. Theoretically, the n-way superscalar pipeline microprocessor can improve performance by a factor of n over a standard scalar pipeline microprocessor. However, in practice, the speed is much smaller. This is also due to instruction dependencies and resource contention [8,17].

Another advancement was in the scheduling and executing of instructions. Previous architecture designs used the approach known as In-order execution which involves the fetching and execution of instructions at compiler generated order. In this approach, once there is a stall in a pipeline stage, all other stages are affected.

The introduction of a more dynamic approach known as the out-of-order scheduling brought about a significant improvement in performance of microprocessors as it reduces stalls compared to in-order execution. In out-of-order execution, instruction execution order depends on data flow and not on the program order as dictated by the compiler, that is, instructions can execute if its operands are available. For example, while waiting for a cache miss, the microprocessor can execute newer instructions as long as they are independent [8,10,16], with these techniques, cycles are properly managed hence the improvement in overall performance of microprocessor.

The need for high speed computation also led to the concept of Massively Parallel Processors (MPP) that exploited data parallelism and in turn led to the introduction of vector computers also known as processor arrays [9,11]. According to [9], a vector computer is a computer whose instruction set includes operations on vectors as well as scalars. Generally there are two ways of implementing a vector computer, pipelined vector processor and processor array.

A pipelined vector processor streams vector from memory to the CPU, where pipelined arithmetic units manipulate them. The Cray-1 super computer and Cyber-205are typical pipelined vector processor.

A processor array is a vector computer implemented as a sequential computer connected to a set of identical, synchronized processing elements capable of simultaneously performing the same operation on different data [9,11]. Processor arrays did not live long due to the advancement in multiple users computers and multiprogramming techniques. Processor array computers do not naturally support multiple users and they had to be built from custom very large scale integrated circuits (VSLI), this complexity and high cost of production made them unpopular as they could not leverage the performance and cost improvements exhibited by commodity CPUs. Therefore, the focus on multiple processor computers involving commodity hardware CPUs became the order of the day [6,9].

As regards the general principle of pipeline parallelism, it was exploited to the point where, control dependences, data dependencies and stalls limited additional pipeline stages and in effect the possibility of achieving computational speedup even at a higher frequency. In addition, the performance gain from frequency scaling could no longer guarantee a safe environment as clock rate above 4.0 Ghz [10,15,16] was practically not feasible due to the heat generated per square area of the chip. Fig. 5 correlates with this fact. Fig. 5 depicts trends in Intel CPU performance.

From the graph, it was shown that around 2003, it became harder to exploit higher clock speeds due to not just one but several physical issues, notably heat (too much of it and too hard to dissipate), power consumption (too high), and current leakage problems. At this point manufacturers and researchers began to think of other means of achieving computation speedup. This led to the concept of multiprocessor computers and multicore computer [13,15,16].

3.2 Multi processor to Multi-Core Parallel Computers with Parallel Programming

The Integrated circuit technology especially the miniaturization of chips still fuelled a lot of technological advancements. Improved clock speed, increase in processor features but with several drawbacks like the energy required to power these microprocessors and the heat dissipated. On top of these drawbacks, there were also economic limitations. At some point, the cost of making a processor incrementally faster exceeded the price that anyone was willing to pay. Ultimately, manufacturers were left with no feasible way of improving performance except to distribute the computation load among several processors while keeping the frequency at a stable level [15].

The idea behind multiprocessor parallel computers was to design an architecture where computational load are distributed across processing nodes and each computation carried out simultaneously across these processing nodes.

Interestingly, parallel computers and parallel programming is not a recent development, it has been in existence for some decades ago. It only got more attention as manufacturers could no longer improve performance by exploiting instruction level parallelism, frequency scaling, and bringing data stream closer to the processor by including multi-level caches [13,16]. In fact, parallelization of computations has been in existence before the multiprocessor or multicore era. In 1981 a group at Caltech led by Charles Seitz and Geoggrey Fox began work on the Cosmic Cube which is a parallel computer constructed out of 64 Intel 8086 microprocessors [12]. This dramatically illustrated the potential for microprocessor-based parallel computing. The Cosmic cube executed its application programs at about 5 to 10 million floating point operations per second. This benchmark speed made the Cosmic Cube 5 to 10 times faster than the Digital Equipment Corporation Vax 11/780 which was the standard minicomputer at that time.

The success of the Cosmic cube led a flurry of parallel computers built out of microprocessors. New computer firms like Sequent, Meiko, nCUBE, Parstec, Encore Silicon Graphics and a host of other computer firms began producing parallel computers made from microprocessors, unfortunately most of these companies went out of business while some where bought over by bigger firms like IBM and Intel [3,6,8].

Other computer manufacturing companies produced parallel computers with a single CPU and thousands of arithmetic-logic units (ALU) implemented in VLSI process technology. The thinking machine corporation was one of such machines. It contained 65,536 single-bit ALUs.

Meanwhile, driven by the popularity of personal computers, another type of parallel computer known as the Beowulf cluster sprang up. A Beowulf cluster is a parallel computer made up of off-the-shelf personal computers connected via ethernet. Its history dates back 1994 with Thomas Sterling and Donald Becker at the Center of Excellence in Space Data and Information Sciences (CESDIS). Beowulf was built at that time to address problem of large data set. It consisted of 16 DX4 processors connected by a 10 Mbps Ethernet. Beowulf became an instant success because of the low cost in implementation [9].

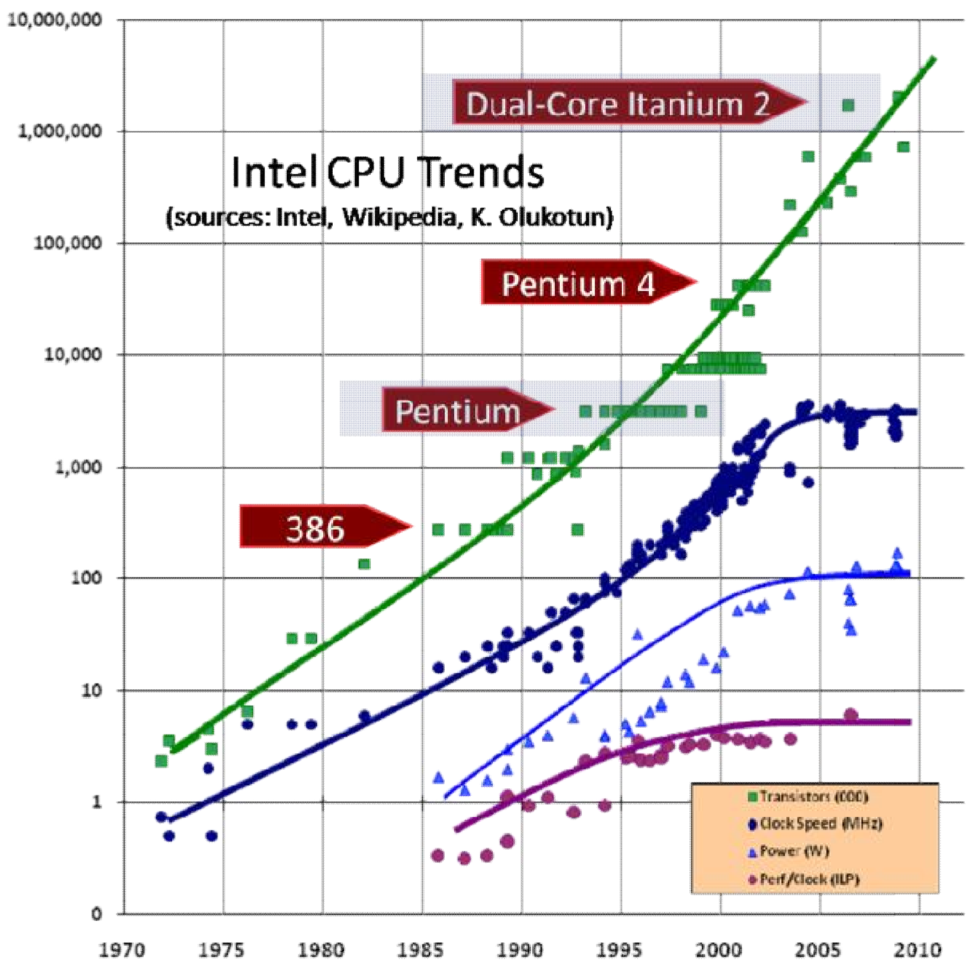

Fig.5. Intel CPU trend
Source: http://www.gotw.ca/publications/concurrency-ddj.htm

Just a few years ago, parallel computers were built either using the multiprocessor approach or multicomputer approach. Multiprocessor approach involves building parallel computers from the combination of multiple microprocessors connected via specialized high speed buses, while multicomputer approach involves building parallel computers from multiple computers connected via a network (in most cases Ethernet network). These two categories perfectly fit into the two parallel programming models namely shared memory architecture parallel computing model and distributed memory architecture parallel computer respectively.

3.3 Parallel Programming Languages and Libraries

The need to program these types of parallel computers led to the development of programming languages and libraries following the two aforementioned parallel programming models (i.e, shared and distributed parallel programming model). In order to achieve parallelism in programs, a new parallel programming language could be built from the scratch but with a steeper learning curve for programmers especially when the parallel language contains a lot of keywords. Also parallelism could be achieved by extending an already existing sequential programming language through providing parallel programming libraries. Message passing libraries fall under this kind of parallel programming language libraries [9,18].

The need to provide a standard for parallel programs involving distributed memory model led to the popular message passing interface (MPI).

Message Passing Interface (MPI), is not a programming language but a standard for message passing mostly for distributed systems. The MPI standard was proposed by a broadly based committee of vendors, implementers, and user namely IBM, Intel, TMC, Meiko, Cray, Convex, Ncube and library writers like PVM, p4, Zipcode, TCGMSG, Chameleon, Express, Linda. This standard was for high performance on both massively parallel machines and on workstations clusters [18]. Versions of MPI include MPI version 1.0, MPI version 1,1, MPI version 1,2 and the MPI version 2.0.

There are several implementations in different programming languages for MPI. The most widely known implementation is MPICH. MPICH is an open source implementation of MPI and it has been tested on Linux (on IA32 and x86-64), Mac OS/X (PowerPC and Intel), Solaris (32 and 64bit) and windows. Current version of MPICH is MPICH version 3.0. Other implementation like MP-MPICH (MP, stands for Multi-platform), WINMPICH (implementation for windows from Mississippi State University), Cray MPI product for the T3D, HP MPI, IBM's MPI for the SP, DeinoMPI and a host of other implementations.

Another popular message passing library is Open MPI. While MPI is well suited for distributed systems, the Open MPI is most suited for shared memory parallel programming model. The Open MPI library project combined technologies and resources from several other projects like (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI) [19]. It is used by many TOP500 supercomputers including roadrunner, which was the world's fasters supercomputer from June 2008 to November 2009, and K computer, the fasters supercomputer from June 2011 to June 2012 [19].

Parallel programming models like the Parallel Virtual Machine (PVM) and the Bulk Synchronous Parallel (BSP) computation model also exist. A typical PVM provides an abstraction of a processor pool. In fact, each processor in the pool can be separate network node. The programmer can use PVM routines to start/stop a task or send message between tasks. PVM exists for different architectures and a typical PVM program written for different network can communicate with other programs thus allowing for building of heterogeneous networking computing systems [20]. Bulk synchronous Parallelism is a programming model that abstracts from low-level program structures in favour of super steps. A super step consists of a set of independent local computations, followed by a global communication phase and a barrier synchronization. A major advantage of BSP is that fact that the runtime can easily be deduced and it also imposes clearly structured communication model upon the program [20]. A-BSP is an implementation of BSP model.

Another parallel programming library worth noting is the Thread Building Block (TBB) by Intel Corporation. The TBB is designed to be used with C and C++ programming languages. Intel's TBB is a high level library that uses threading libraries and hides the details about the threading mechanisms for performance and scalability [20,21].

4 Multicore Parallel Computers

The most recent trend in building parallel computers involves multi-core architectures. The idea is to keep frequency at processing power at a bearable level while duplicating the processing cores of the microprocessor [13,15,16]. This in effect helps in maintaining the threshold of the amount of heat dissipated while at the same time improving computation since task can be distributed among the processing cores. The operating system conceptually see multiple processors but in effect these multiple processors are all integrated into a single chip.

Multi-core processors can be implemented as homogeneous or heterogeneous cores. A multi-core processor is homogeneous when all its cores each have the same functions or features. On the other hand, a multi-core processor is heterogeneous if within the number of cores, some cores have different functions in the overall computational process.

CELL multicore processors from IBM follow the heterogeneous implementation of heterogeneous multi-cores. The CELL process consists of nine cores on a single chip, one of these cores is the Power Processing Element (PPE) and the remaining eight are Synergistic Processing elements (SPEs) [13,15]. The CELL processor's architecture is specifically designed for gaming environment which requires a lot of graphics processing. The PPE is an extension of the 64-bit PowerPC Architecture and manages the operating system and control function, while each SPE has simplified instruction sets which use 128bit-SIMD(Single Instruction Multiple Data) instructions and have 256 KB of local storage [13,15]. Power has always being a problem with micro-processor design so the CELL process provides a Power management Unit (PMU) and Thermal Management Unit (TMU). The PMU allows power management in the form of slowing, pausing, or completely stopping a unit.

Adding an additional processing core to the same chip in theory results in twice the performance and dissipate less heat, though in practice actual speed of each core is slower than the fastest single core processor [15]. In September 2005, the IEEE noted that "power consumption increases by 60% with every 400 MHz in clock speed. But the dual core approach means you can get a significant boost in performance without the need to run at higher switching frequencies (clock rates). According to [16], dual core processor at 20% reduced clock frequency effectively delivers 73% more performance while approximately using the same power as a single-core processor at maximum frequency. This observation is obviously the reason why there is a shift towards multicore processors by leading microprocessor manufacturers like Intel and AMD. However, it should be noted that the performance gain is directly proportional to the problem definition and the dexterity in programming.

Intel and AMD are companies that have played important roles in the multicore era. Intel has produced many different flavours of multicore processors: the Pentium D which is used in desktops, Core 2 Duo used in both laptops and desktop environments and the Xeon processor used in Servers are all produced by Intel. AMD has the Althlon line-up for desktops, Turion for laptops, and Opteron for servers/workstations.

Even more recently there has been a move for more cores. According to [21], the fundamental relationship between power and frequency can be effectively used to multiply the number of cores from two to four, or even more.

4.1 Parallel Graphic Processors

Another wave of performance enhancement is in the separation of graphic intensive processing from the main microprocessor. This was done through the use of graphic processing units (GPU). These units are specifically designed to optimize graphics and could employ multicore in its hardware architecture. According to [21] GPUs are characterized by numerous simple yet energy-efficient computational cores, thousands of simultaneously active fine-grained threads and large off-chip memory bandwidth. One of the leading GPUs manufacturers is NVIDIA with its Fermi GPU architecture which has support for both high-performance graphics and high performance computing. Fermi offers a peak throughput of more than 1 teraflop of single-precision floating-point performance and 175 Gbytes/second of off-chip memory bandwidth [21].

4.2 Multicore/Shared Memory Programming Languages and Libraries

With the proliferation of multicores and graphic processing units come systems with additional set of parallel programming languages for these machines. One of such languages gaining grounds is the Intel Cilk++. The Cilk++ language extends C++ to simplify writing parallel applications that efficiently explore multicore processors. Problems set that involves Divide and Conquer approach are particularly suited for this type of programming environment [22].

With the advent of GPUs came the programming platforms and models known as CUDA and OpenCL. These languages help programmers express parallelism in their codes for parallel computers with GPUs. CUDA, introduced in 2006 [16,23,24] is a parallel computing platform and programming model that enables dramatic increases in computer performance by harnessing the power of the graphic processing unit (GPU). Programmers can add support for GPU acceleration in their own programs using any of the three simple approaches;

i. Drop in GPU-accelerated library to replace or augment CPU-only libraries such as MLK BLAS, IPP, FFTW and other widely-used libraries
ii. Automatically parallelize loops in Fortran or C code using OpenACC directives for accelerators
iii. Develop customer parallel algorithms and libraries using a familiar programming language such as C, C++, C#, Fortran, Java Python, etc.

OpenCl (Open Computing Language) is a low-level API for heterogeneous computing that runs on CUDA-power GPUs and other GPUs. Using the OpenCL API, developers can launch compute kernels written using a limited subset of the C programming language on a GPU [24]. The idea behind OpenCl is to write portable codes that can be executed across different heterogeneous GPU-available devices

4.3 Current Challenges Facing Multicore Parallel Computers

Just like every other technology, there are a host of challenges facing the adoption of multicore computers. These challenges are;

4.4 Energy/Heat Constraints

It is obvious that integrating many cores into a single chip brings to the fore the problem of power. Power must be brought and distributed among the cores of the process. In theory as more cores are added to the chips, more power will be needed to drive each of these cores. This is turn has the tendency of generating immense heat over a short period of time. One way manufacturers combat this problem is to run these cores at reduced frequencies thereby reducing the amount of heat generated. Also an additional cooling unit is introduced to the design; this unit has the authority of shutting down an unused core. This way power is reduced and energy is efficiently managed.

Workload distribution is also another constraint. When process to core scheduling is not down properly, a portion of the processor tends to heat up more quickly than another. This is in general not good for the overall life of the microprocessor and semiconducting material.

Also, leakage energy has seen frequency scaling stopped to a large extent causing energy per operation to now scale only linearly with the process feature. According to [16], because power dominates lifetime system costs for supercomputers and servers, their utility is also determined by performance at the target system power level. For example, the reasonable power envelope for future supercomputers has been projected to be 20 MW. The supercomputing community is now aiming to design exascale (1018 operations/second) systems. To build such a system within 20 MW requires an energy efficiency of approximately 20 picojoules (pJ) per floating point operation. Future servers and mobile devices will require similar efficiencies. Achieving this level of energy efficiency is a major challenge that will require considerable research and innovation. Therefore energy-efficiency improvements beyond what comes from device scaling will require reducing both instruction execution and data movement overheads.

4.5 Programmability

Programming parallel computers has always being a problem. In the history of computing, computer programs have always been written sequentially. For computer programs to experience the speedup available with the introduction of parallel computers, these programs would have to be written specifically to exploit parallelism. This brings about the problem of portability. In [13], it was noted that applications on multicore systems don't get faster automatically as cores are increased. For speedup to be achieved, programmers must explicitly write applications that exploit the increasing number of processors in a multicore environment. Therefore, the great challenge is how to port existing software into the multicore computer. The option of writing a low-level compiler to exploit parallelism in sequential programs is also not a trivial task as different data/control dependencies issues and process schedules among parallel processing nodes is an issue to deal with.

Also, not all computer programs give room for parallelism. With these type of programs, there is little improvement a parallel computer will do to speedup its executions. The only notable improvement will be in the increase in the number of instructions per cycle.

A phenomenon known as Amdahl's law states the effectiveness of a parallel architecture on speedup of computation [8,9,11,25]. It states that the speedup attained with the introduction of parallelism in the architecture is inversely proportional to the number of sequential portion of the programs that can't be parallelize and the number of parallel portion of the program divided by the number of processing unit. That is speedup for parallel computation is$speedup \leq \frac{1}{f + \frac{1-f}{p}}$, where f is the fraction of operations in a computation that must be performed sequentially. In effect, if all the portion of the program must be executed sequentially then improvement in speedup cannot be achieved through the introduction of parallel architecture.

However, there are a category of programs known as embarrassingly parallel programs for which parallelism is very well suited. These types of parallel programs can be efficiently scheduled among the processing units of a parallel computers and the overall speedup of computation can be increased as the number of processing units increases.

In parallel computers like the multicore parallel computers where there are different levels of caches, there is the issue of cache coherence. In a typical multicore architecture, each core is designed to have its own cache. Hence, it becomes difficult for caches in each core to always have up-to-date instance of a shared variable. This phenomenon is known as cache incoherence.

Ronny et al. [16], in his work outlined two approaches to resolving cache incoherence. These approaches are known as the Snoop Protocol and the Directory Based Protocol. The Snoop

Protocol only works with a bus-based system, and uses a number of states to determine whether or not it needs to update cache entries. The Directory Protocol on the other hand can be used on an arbitrary network hence, scalable to many processors or cores.

Also in multicore programming, there is the issue of starvation and multithreading. Multicore workload should be balanced. If one core is being used much more than another, the programmer is not taking full advantage of the multicore system. As outlined in the previous chapter, this in turn means programs will have to be rewritten to take advantage of multicore architecture available.

Another challenge is in the programming of heterogeneous multicore architectures. Programmers would have to determine which core execute part of a programming task better and be able to write programs tailored to such core.

4.6 Future of Parallel Computers and Parallel Programming

The technology world has seen computers evolved from slow very large energy sucking machines to smaller, faster, more energy efficient computers. It has also seen the computers gain tremendous speed from the improvements in computer architecture through the concept of instruction level parallelism, pipelining to superscalar and vector processor computers. The quest for high performance computers also saw the introduction of massively parallel computers, multiprocessor and multicore computers. It may be seen that most options for making computers faster have been exploited; therefore for computers to become faster than the fastest computers of now, newer concepts would have to be discovered not only in the hardware composition but also in the way these computers are programmed.

From the hardware perspective, it has been seen over the years that transistors form the basic building blocks for digital devices. It can be said that, until a newer technology for micro-chip composition is discovered, computational speedup gained from frequency scaling will no longer be viable as the heat dissipated and the power required is still a constraint.

Already there has been a shift in ideologies behind computers. Scientists are beginning to look into non-silicon technologies; materials like Graphene, Molybdenite and indium gallium arsenide are being suggested as possible replacements for silicon made transistors [7]. If successful, these technologies could indeed keep Moore's law and improve computational speedup.

Conversely, a different approach is being considered in producing computing devices. Concepts like molecular computers and quantum computers are gaining grounds. molecular computers use DNA, biochemistry and molecular biology, instead of traditional silicon-based computer technologies while quantum computers uses quantum mechanical phenomena, such as superposition and entanglement, to performance operations on data. Data is represented in quantum computers as qubits which can be either in an off state, on state or both on and off states. Quantum computers seem to have more prospects as it promises computational speed beyond comprehension. However, these newer technologies come with the bottleneck of production and programming.

With these new developments in technology, Moore's law can be saved and computers can become even faster. The concept of parallel computation has come to stay even though newer non-silicon technologies promise high computational speedup. Parallelism will be an added advantage over a very fast hardware device.

Computational speedup is the utmost goal for every research into faster computers. The major problem in achieving faster computation is not just building faster hardware but making good use of this hardware in computation. Needless to say, the major problem still lies with programming these hardware devices. The dependency in the sequence of codes fed into the computer is one of the greatest problems facing the attainment of computational speedup. For problems that are void of dependencies (embarrassingly parallel problems), speedup is only a function of how many processing units are available and how fast these processing units are. For these kinds of problems, optimization of hardware and increase in the processing units will always produce an exponential computational speedup with an efficient scheduling scheme.

However, a good number of computational problems exhibit both data dependency and control dependencies constraints. For these types of problems, there is a limit to which parallelism can add to computational speedup. This bottleneck is not new to parallel computation as Gene Amdal shown in 1967 that the speedup obtainable from a problem is limited by the amount of sequential or non-parallelizable portion of the problem.

At the moment, parallelism seems to be the popular option to making computers faster. This approach will eventually get to a point where even the fastest computer would not be able to improve overall computational speed until a better way to solve dependency issues within the computation is resolved. The big question will then be, of what need would it be to achieve or built a superfast computer when it cannot solve a dependency-intensive problem faster than its predecessor?

Therefore, in the near future, computational speedup would not be a function of how fast the hardware is but how cleverly the computer programmer solves a problem on a fast computer.

5 Conclusion

The quest for faster computers has seen advancements in computer manufacturing technology spanning single processor computers to parallel computers and now multicore computers with the inclusion of graphic processing unit. In this paper, timelines of activities leading to parallel computers and parallel programming have been outlined.

Various factors such as power, heat, process technology have been outlined as the major thrust that led to newer technology and the adaptation of parallel computers as a way to improve computational speedup. Frequency scaling techniques, instruction set architecture improvement, pipelining, superscalar pipelining, vector processing, multiprocessor and multicore technologies have all been explored to make computers fasters.

Until and even after newer technologies involving non-silicon transistors become mainstream, it is obvious that to achieve higher computational speed, manufacturers will continue to explore speedup options inherent in the parallel computers by increasing the number of processing units as the process technology gets better. More and more computers with higher number of cores will be designed.

The focus will then be to better engineer the instruction set architecture to improve performance of these parallel systems, be it multiprocessor parallel computers or multicore parallel computers.

Programming parallel computers is most important if speedup is to be achieved on a parallel computer. Advancements will have to be made in improving the algorithms that will run on these parallel computers. It is obvious that a quick porting of existing software and application on these parallel computers will not be an easy task but a gradual process of re-engineering these software and applications to fit into these parallel architectures. With the birth of heterogeneous parallel computers, it is expected that dynamism in processor to task scheduling will have to be explored so as to reduce some of the parallel programming problems outlined in this paper. As long as programs involving a lot of data and control dependencies exist, there would still be bottlenecks as to speedup of computations. Parallel programs that are embarrassingly parallel will be the ultimate benefactors of computational speedup achievable from parallel hardware.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Xing Cai, Acklam E, Hans PL, Aslak T. Parallel computing; 2014. Retrieved: http://www.diffpack.com/res/doc/Diffpack/Reports/pdp-chap.pdf

[2] Intel Threading Building Blocks -Intel TBB 4.1. (n.d.). Retrieved: http://software.intel.com/en-us/intel-tbb

[3] 1953 - Transistorized Computers Emerge; 2013. Retrieved:http://www.computerhistory.org/semiconductor/timeline/1953-transistorized-computers-emerge.html

[4] Balaj IV. Multi-core processors: An overview. Liver, United Kingdom; 2013. Retrieved:arxiv.org/pdf/1110.3535

[5] John LH, David AP. Computer architecture, a quantitative approach 4th edition. Amsterdam: Morgan Kaufmann Publisher; 2007.

[6] Computer History Museum; 2014. Retrieved:http://www.computerhistory.org/semiconductor/timeline/1953-transistorized-computers-emerge.html

[7] Radisavljevic B, Radenovic A, Brivio J, Giacometti V, Kis A. Single-layer MoS2 transistors. Nature Nanotechnology; 2011. DOI: 10.1038/nnano.2010.279.

[8] Shijjan G. Shiva computer design and architecture. New York: Marcel Dekker; 2000. ISBN0-9876-5432-1

[9] Michael JQ. Parallel Programming in C with MPI and OpenMP. New York: McGraw-Hill; 2003.

[10] Linda N, Lobur J. The essentials of computer organization and archictecture. London: Jones and Bartlett Publishers, Inc; 2006.

[11] Behrooz P. Introduction to parallel processing algorithms and architectures. New York: Kluwer Academic Publishers; 2002.

[12] McCartney Scott. ENIAC: The triumphs and tragedies of the world's first computer. Walker & Co; 1999. ISBN 0-8027-1348-3.

[13] Ramanathan R. Intel® multi-core processors: Making the move to quad-core and beyond. Technology Intel® Magazine; 2006.

[14] Mostafa AEB, El-Rewini H. The fundamentals of computer organization and architecture. Canada: John Wiley & Sons; 2005.

[15] Byan S. (n.d.). Multicore processors – anecessity; 2013.
Retrieved:www.csa.com/discoveryguides/multicore/review.pdf

[16] Ronny R, Avi M, Konrad L, Shih-Lien L, Fred P, John PS. Coming Challenges in Micro architecture and Architecture. Processings of the IEEE. 2001;89.

[17] www.wikipedia.com.org. Pipelining; 2014.
Retrieved:http://en.wikipedia.org/wiki/Pipeline_%28computing%29

[18] Standard Message Passing interface (MPI). (n.d.).
Retrieved:http://www.mcs.anl.gov/research/projects/mpi/

[19] OpenMPI. (n.d.). Retrieved: http://en.wikipedia.org/wiki/Open_MPI

[20] Brown NE. Type oriented parallel programming. Doctoral thesis, Durham University; 2013. Retrieved: http://etheses.dur.ac.uk/106/

[21] Stephen WK, r William JD, Brucek K, Michael G, David G. GPU and the future of parallel computing. IEE Computer Society. 2011;7-17.

[22] Intel. (n.d.). Intel Cilk++ SDK Programmer's Guide. Retrieved: https://software.intel.com

[23] NVIDIA. (n.d.). Developer Zone; 2013. Retrieved: https://developer.nvidia.com/what-cuda

[24] NVIDIA. (n.d.). Developer Zone: OpenCl; 2013.
Retrieved: https://developer.nvidia.com/opencl

[25] Rinat K, Ahmed PHDJTV. (n.d.). Distributed parallel computing in networks of workstations: A survey Study; 2013.
Retrieved:http://citeseerx.ist.psu.edu:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.3661&rep=rep1&type=pdf

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
www.sciencedomain.org/review-history.php?iid=932&id=6&aid=7976