Multi-Core Processors and Systems: State-of-the-Art and Study of Performance Increase

Abhilash Goyal
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000

Abhilash_Goyal@yahoo.com

ABSTRACT

To achieve the large processing power, we are moving towards Parallel Processing. In the simple words, parallel processing can be defined as using two or more processors (cores, computers) in combination to solve a single problem. To achieve the good results by parallel processing, in the industry many multi-core processors has been designed and fabricated. In this class-project paper, the overview of the state-of-the-art of the multi-core processors designed by several companies including Intel, AMD, IBM and Sun (Oracle) is presented. In addition to the overview, the main advantage of using multi-core will demonstrated by the experimental results. The focus of the experiment is to study speed-up in the execution of the 'program' as the number of the processors (core) increases. For this experiment, open source parallel program to count the primes numbers is considered and simulation are performed on *3 nodes Raspberry cluster*. Obtained results show that *execution time* of the parallel program *decreases* as number of core increases. In addition, it was found that if parallelism is introduced in the program beyond the limits that hardware can support, then execution time actually increases.

1. INTRODUCTION

In today's world Parallel Processing is becoming increasingly important. In the simple words, parallel processing can be defined as using two or more processors (cores, computers) in combination to solve a single problem. Parallel processing involves taking a large task, dividing it into several smaller tasks, and then working on each of those smaller tasks simultaneously. The goal of this divide-and-conquer approach is to complete the larger task in less time than it would have taken to do it in one large chunk as illustrated in Figure 1 [1].

Parallel processing not only increases processing power, it also offers several other advantages when it's implemented properly. Few of them are higher throughput, more fault tolerance and better price/performance. Ideally, the parallel processing speedup would track with the number of processors used for any given task. In other words, the ideal speedup curve is a 45-degree line as shown in Figure 2.

The prefect speedup curve is rarely reached because parallelism entails a certain amount of overhead. The inherent parallelism of the application also plays an important role in the amount of speedup. Some tasks are easily divided into parts that can be processed in parallel. In those scenarios, speed up will most likely follow "common trajectory" as shown in Figure 2. If an application has little or no inherent parallelism, then little or no speedup will be achieved and because of overhead, speed up may follow as show by "occasional trajectory" in Figure 2.

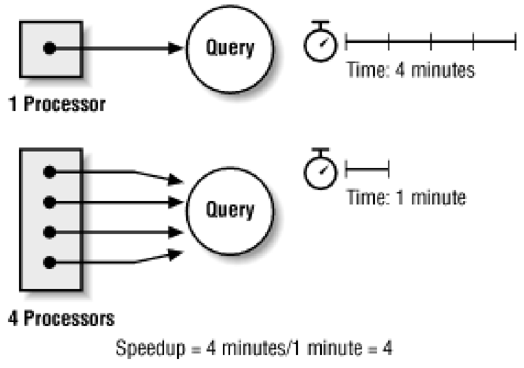

Figure 1. Example of parallel processing to speed up the performance [1].

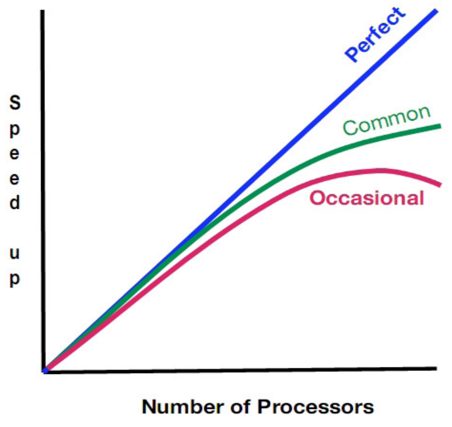

Figure 2. Processing Speed Up with number of Cores/Processors [2].

The subject of parallel processing has attracted attention from scientists and engineers, as well as from commercial vendors.

Over the years, several commercially available parallel hardware platforms have been developed by the several companies (Intel, AMD, IBM and Sun (Oracle)).

In this paper, overview of the state-of-the-art of the multi-core processors designed by these companies is presented. In addition to the overview, the advantage of using multi-core will demonstrated by the experimental results. The focus of the experiment is to study speed-up in the execution of the 'program' as the number of the processors (core) increases. For this experiment, open source parallel program to count the primes numbers is considered and simulation are performed on 3 nodes Raspberry cluster. Obtained results show that *execution time* of the parallel program decreases as number of core *increases*. In addition, *it was found that if parallelism is introduced in the program beyond the limits that hardware can support, then execution time actually increases.*

The rest of the paper is organized as follows. In Section 2, the different kinds of multi-core architectures are explained. In the Section 3, overview of the multi-core processor or system developed my companies is presented. In Section 4, the experiment results are shown. Finally, discussions with conclusions and acknowledgement are presented.

2. MULTI-CORE PROCESSOR AND SYSTEM ARCHITECTURES

Although system designs differ from one another, multicore architectures need to adhere to certain aspects. The basic configuration of a microprocessor is seen in Figure 3.

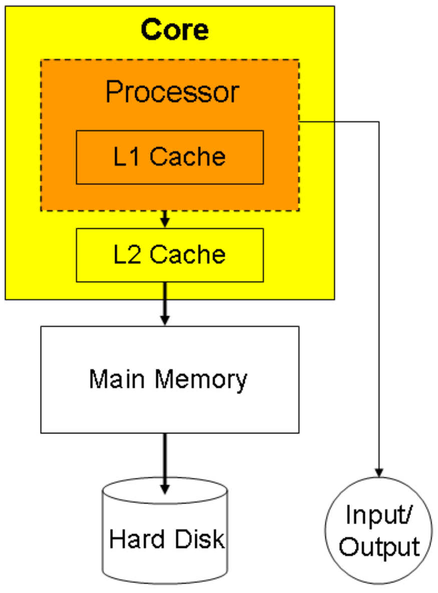

Figure 3 . Generic Single-Core Configuration.

Closest to the processor is Level 1 (L1) cache, this is very fast memory used to store data frequently used by the processor. Level 2 (L2) cache is just off-chip, slower than L1 cache, but still much faster than main memory; L2 cache is larger than L1 cache and used for the same purpose. Main memory is very large and slower than cache and is used, for example, to store a file currently being edited in Microsoft Word. Most systems have between 1GB to 4GB of main memory compared to approximately 32KB of L1 and 2MB of L2 cache. Finally, when data isn't located in cache or main memory the system must retrieve it from the hard disk, which takes exponentially more time than reading from the memory system [3].

At its simplest, multi-core is a design in which a single physical processor contains the core logic of more than one processor. It's as if an Intel Xeon processor were opened up and inside were packaged all the circuitry and logic for two (or more) Intel Xeon processors as shown in Figure 4. The multi-core design puts several such processor "cores" and packages them as a single physical processor.

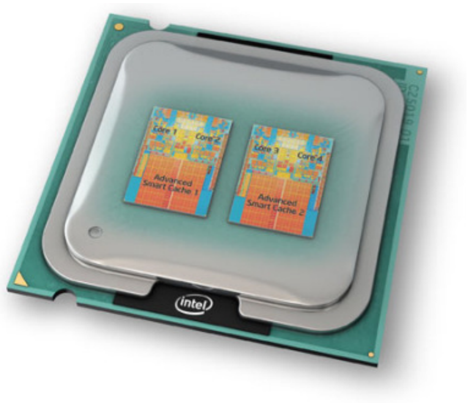

Figure 4. Intel's multi-core with multiple Single Cores packed together [4].

Based on the number of data streams that can be processed at a single point in time, any given computing system can be described in terms of how instructions and data are processed. This classification is known as Flynn's taxonomy (Flynn, 1972) and is graphically shown in Figure 5.

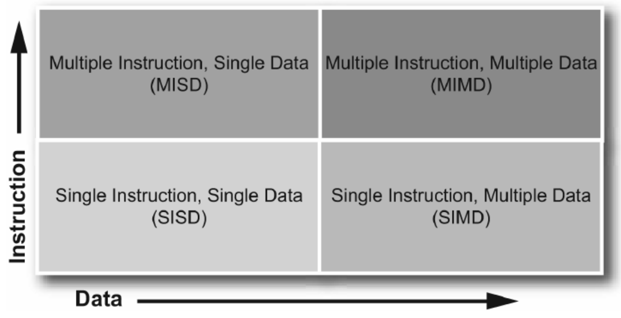
Multiple Instruction, Single Data (MISD)
Multiple Instruction, Multiple Data (MIMD)
Single Instruction, Single Data (SISD)
Single Instruction, Multiple Data (SIMD)
Instruction
Data

Figure 5. Flynn's Taxonomy [5]

Multiple Instructin and Multiple Data (MIMD) is the most adopted architecture and its two classic variants are shared memory and distributed memory. This lead to the two classic software models as shared memory and message passing. In shared memory MIMD architecture all process have direct access to all the memory as shown in Figure 6.A. While in distributed memory MIMD, each processor has its own individual memory as shown in Figure 6.B.

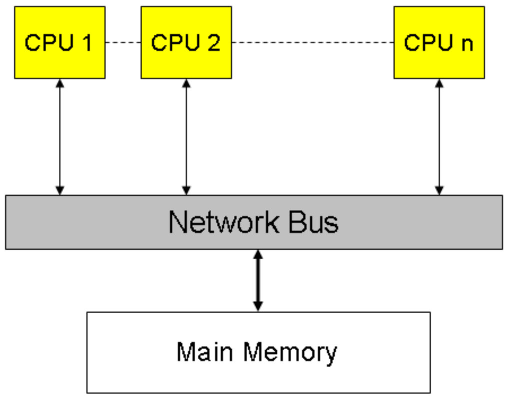

Figure 6A. Shared Memory System.

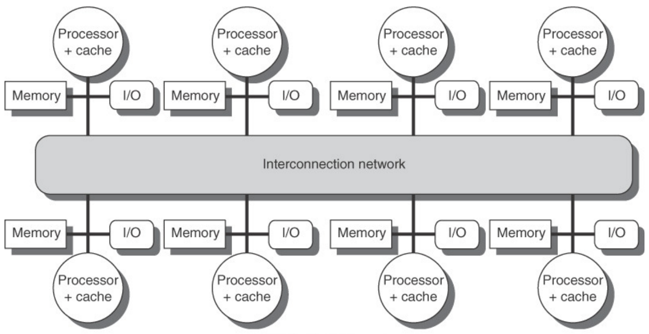

Figure 6B. Distributed Memory System

Figure 6. Different variant of MIMD systems based on memory [2].

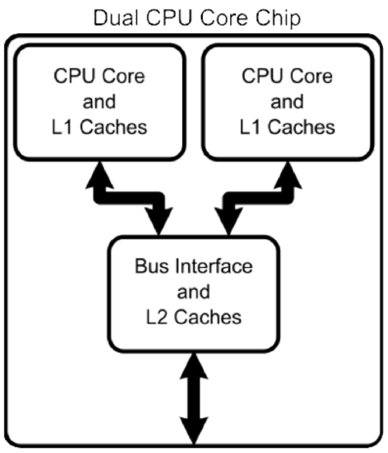

Figure 7B. Micro – (Shared + Distributed)

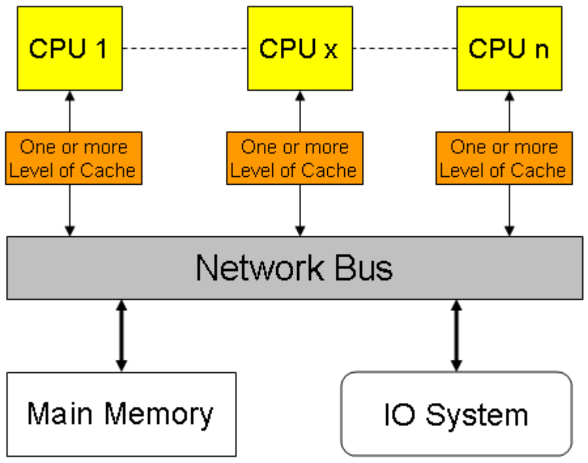
CPU 1
CPU x
CPU n
One or more Level of Cache
One or more Level of Cache
One or more Level of Cache
Network Bus
Main Memory
IO System

Figure 7B. Macro – (Shared + Distributed)

Figure 7. Hybrid of Shared and Distributed Multi-core System. [2]

The hybrid of shared with distributed memory system is also other type of architecture which very common in the research community and industry. This hybrid can be at macro of micro level as shown in Figure 7.

3. STATE-OF-THE-ART

The composition and balance of the cores in multi-core architecture show great variety. Some architectures use one core design repeated consistently "homogeneous", while others use a mixture of different cores, each optimized for a different, "heterogeneous" role. Differences in architectures are discussed below for Intel‟s Core 2 Duo, Advanced Micro Devices‟ Athlon 64 X2, Sony-Toshiba- IBM‟s CELL Processor, Tilera‟s TILE64, Kalray's MPPA 256 and finally Epiphany's E16G401.

3.1 INTEL and AMD Dual-Core Processors

Intel produces many different flavors of multicore processors: the Pentium D is used in desktops, Core 2 Duo is used in both laptop and desktop environments, and the Xeon processor is used in servers. AMD has the Althon lineup for desktops, Turion for laptops, and Opteron for servers/workstations. Although the Core 2 Duo and Athlon 64 X2 run on the same platforms their architectures differ greatly.

Block diagrams for the Core 2 Duo and Athlon 64 X2 is respectively shown in Figure 8A and Figure 8B. Both architectures are homogenous dual-core processors. The Core 2 Duo adheres to a shared memory model with private L1 caches and a shared L2 cache which "provides a peak transfer rate of 96 GB/sec." [3]. If a L1 cache miss occurs, both the L2 cache and the second core‟s L1 cache are traversed in parallel before sending a request to main memory. In contrast, the Athlon follows a distributed memory model with discrete L2 caches. These L2 caches share a system request interface, eliminating the need for a bus.

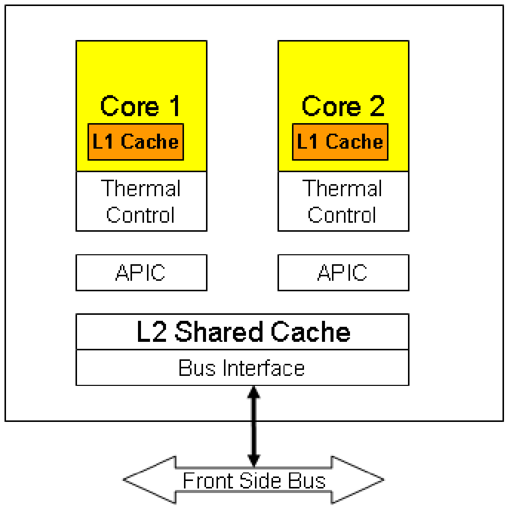

Figure 8A. Intel: Dual Core

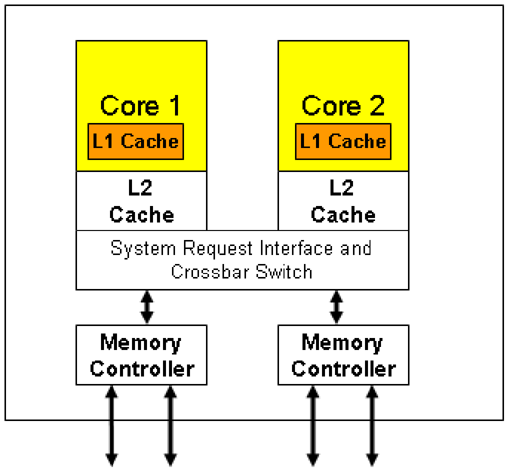

Figure 8B: AMD Dual Core

Figure 8: Architecture of Dual-Core Intel and AMD Processors

The system request interface also connects the cores with an on-chip memory controller and interconnect called HyperTransport. HyperTransport effectively reduces the number of buses required in a system, reducing bottlenecks and increasing bandwidth. The Core 2 Duo instead uses a bus interface. The Core 2 Duo also has explicit thermal and power control units on-chip. There is no definitive performance advantage of a bus vs. an interconnect, and the Core 2 Duo and Athlon 64 X2 achieve similar performance measures, each using a different communication protocol.

3.2 INTEL : Teraflops Research Chip

The Teraflops Research Chip (also called Polaris) is a research processor containing 80 cores developed by Intel Corporation's Tera-Scale Computing Research Program as shown in Figure 9. Features of the processor include dual floating point engines, sleeping-core technology, self-correction, fixed-function cores, and three-dimensional memory stacking. The purpose of the chip is to explore the possibilities of Tera-Scale architecture (the process of creating processors with more than four cores) and to experiment with various forms of networking and communication within the next generation of processors. The processor was officially announced February 11, 2007 and shown working at the 2007 International Solid-State Circuits Conference [6].

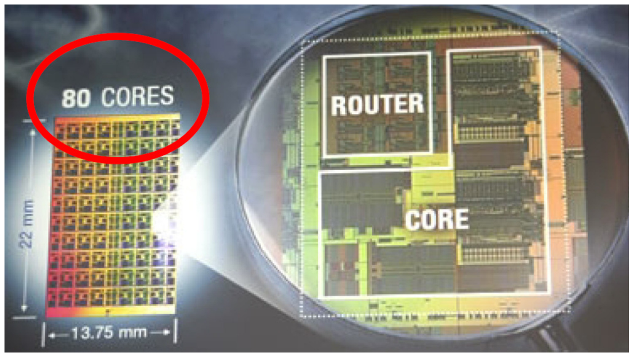

Figure 9. Intel : Teraflops Research Chip [2]

3.3 IBM-SONY-TOSHIBA: CELL

A Sony-Toshiba-IBM partnership (STI) built the CELL processor for use in Sony's PlayStation 3, therefore, CELL is highly customized for gaming/graphics rendering which means superior processing power for gaming applications. The CELL is a heterogeneous multicore processor consisting of nine cores, one Power Processing Element (PPE) and eight Synergistic Processing Elements (SPEs), as shown in Figure 10. With CELL's real-time broadband architecture, it is possible to 128 concurrent transactions to memory per processor.

The PPE is an extension of the 64-bit PowerPC architecture and manages the operating system and control functions. Each SPE has simplified instruction sets which use 128-bit SIMD instructions and have 256KB of local storage. Direct Memory Access is used to transfer data between local storage and main memory which allows for the high number of concurrent memory transactions. The PPE and SPEs are connected via the Element Interconnect Bus providing internal communication [3].

Other interesting features of the CELL are the Power Management Unit and Thermal Management Unit. Power and temperature are fundamental concerns in microprocessor design. The PMU allows for power reduction in the form of slowing, pausing, or completely stopping a unit. The TMU consists of one linear sensor and ten digital thermal sensors used to monitor temperature throughout the chip and provide an early warning if temperatures are rising in a certain area of the chip. The ability to measure and account for power and temperature changes has a great advantage in that the processor should never overheat or draw too much power [7].

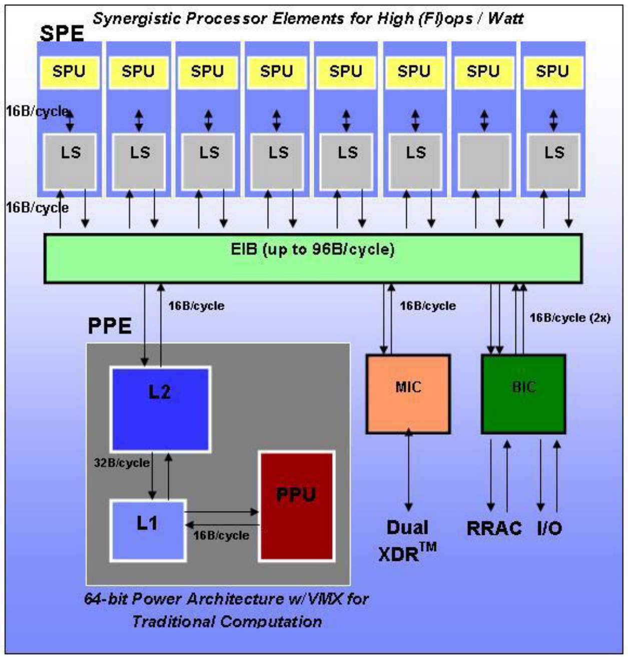
Synergistic Processor Elements for High (Fl)ops / Watt
SPE
SPU
LS
16B/cycle
EIB (up to 96B/cycle)
PPE
16B/cycle
16B/cycle (2x)
L2
MIC
BIC
32B/cycle
L1
PPU
16B/cycle
Dual XDR™
RRAC
I/O
64-bit Power Architecture w/VMX for Traditional Computation

Figure 10A. CELL Processor: Architecture

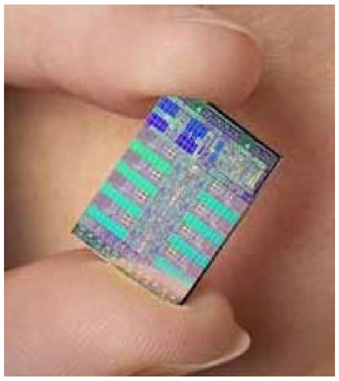

Figure 10B. CELL Processor: Fabricated Chip

Figure 10. IBM-Sony-Toshiba: CELL Processor [3, 6, 7].

3.4 TILERA: TILE64

Tilera has developed a multicore chip with 64 homogeneous cores set up in a grid, shown in Figure 11. An application that is written to take advantage of these additional cores will run far faster than if it were run on a single core. Each processor has its own L1 and L2 cache for a total of 5MB on-chip and a switch that connects the core into the mesh network rather than a bus or interconnects. The TILE64 also includes on-chip memory and I/O controllers. Like the CELL processor, unused tiles (cores) can be put into a sleep mode to further decrease power consumption. The TILE64 uses a 3-way VLIW (very long instruction word) pipeline to deliver 12 times the instructions as a single-issue, single-core processor. When VLIW is combined with the MIMD (multiple instruction, multiple data) processors, multiple operating systems can be run simultaneously and advanced multimedia applications such as video conferencing and video-on-demand can be run efficiently [3,6,8].

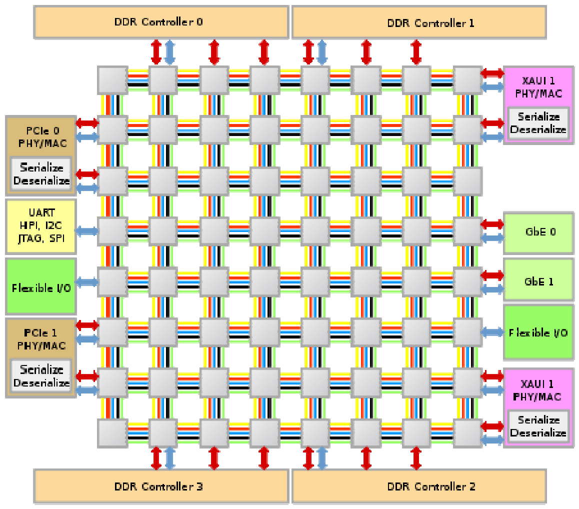

Figure 11A. TILE64 Processor: Architecture

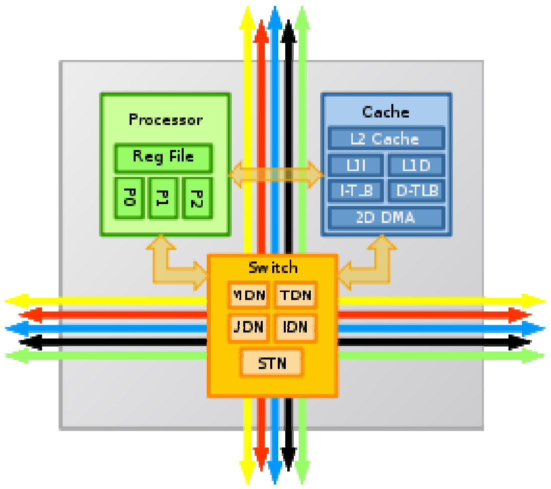

Figure 11A. Sigle Core Unit of TILE64

Figure 11. Tilera: TILE64 [6].

3.5 KALRAY: MPPA 256

KALRAY is a fabless semiconductor & software company developing and selling a new generation of manycore processors for high performance applications. MPPA product feature voltage and frequency scaling, allowing optimal power dissipation and computing performance. The MPPA 256 is composed of an array of clusters connected through a high-speed Network-on-Chip (NoC) as shown in Figure 12. Each cluster contains 16 processing cores, a system core and a shared memory, as shown in Figure x. Multiple MPPA processors can be connected together at board level through a NoC extension, thus enabling virtually any size of processor arrays. The NoC is a 2D-wrapped-around torus structure providing a full duplex bandwidth up to 3.2 GB/s between each adjacent cluster. The NoC implements a Quality of Service mechanism, thus guaranteeing predictable latencies for all data transfers. The MPPA MANYCORE processor combines high performance capabilities with low power dissipation, making it a perfect fit for power efficient and embedded applications such as image and audio processing, signal processing, intensive computing, control command and telecom.

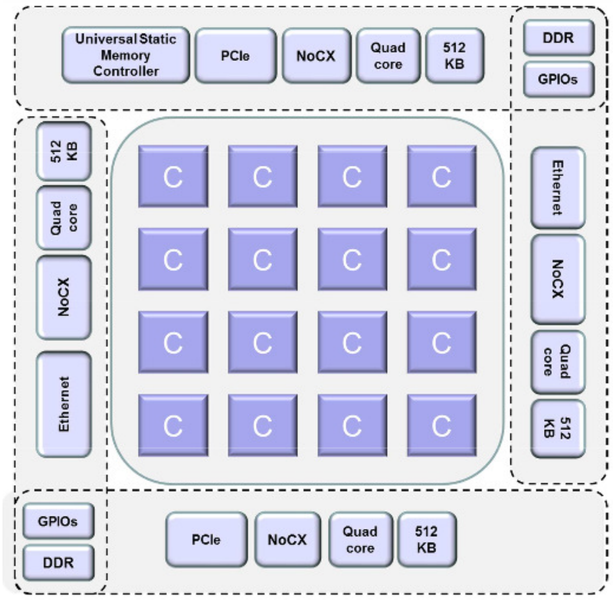

Figure 12A. MPPA Processor: Architecture

Figure 12B. MPPA Processor: Fabricated Chip

Figure 12. Kalray MPPA Processor [9].

The MPPA core is a 32-bit Very Long Instruction Word (VLIW) processor made of One Branch/Control Unit,Two Arithmetic Logic Units, One Load/Store Unit including simplified ALU, One Multiply-Accumulate (MAC) / FPU including a simplified ALU, Standard IEEE 754-2008 FPU with advanced Fused Multiply-Add (FMA) and dot product operators and One Memory Management Unit (MMU). This enables to execute up to five 32bit RISC like integer operations every clock cycle [9].

The MPPA MANYCORE processor communicates with the external devices through I/O subsystems located at the periphery of the NoC. The I/O subsystems implement various standard interfaces. Interfaces of MPPA-256 includes, Two DDR3 channels, Two PCIe Gen3 X8, PCIe Bus master. Two smart Ethernet Controllers, Universal Static Memory Controller, Two banks of 64 General Purpose I/Os and NoC eXpress interfaces (NoCX).

3.6 ADAPTEVA EPIPHANY : E64G401

The E64G401 [10] is a 64-core System-On-Chip implemented in a 65nm as shown in Figure 13. The Epiphany architecture is a scalable, distributed-shared-memory computing fabric comprised of a 2D array of processor nodes connected by a low-latency mesh network-on-chip. The E64G401 includes 64 superscalar floating point RISC CPUs (eCore), each one capable of two floating point operations per clock cycle and one integer calculation per clock cycle.

The CPU has an efficient general-purpose instruction set that excels at compute intensive applications while being efficiently programmable in C/C++. The Epiphany memory architecture is based on a flat shared memory map in which each compute node has up to 1MB of local memory as a unique addressable slice of the total 32-bit address space. A processor can access its own local memory and other processors' memory through regular load/store instructions. The local memory system is comprised of 4 separate sub-banks, allowing for simultaneous memory accesses by the instruction fetch engine, local load-store instructions, and by memory transactions initiated by the DMA engine other processors within system.

The Epiphany's 2D Network-on-Chip (eMesh) handles all on-chip and off-chip communication. The eMesh network uses atomic 32-bit memory transactions and operates without the need for any special programming. The network consists of three separate and orthogonal mesh structures, each serving different types of transaction traffic: one network for on-chip write traffic, one network for off chip write traffic, and one network for all read traffic.

The eMesh network and memory architecture extends off-chip using source synchronous dual data rate LVDS links ("elinks"). Each E64G401 has 4 independent off-chip elinks, one in each physical direction (north, east, west and south). The off chip links allows for glueless connection of multiple E64G401chips on a board and for interfacing to an FPGA.

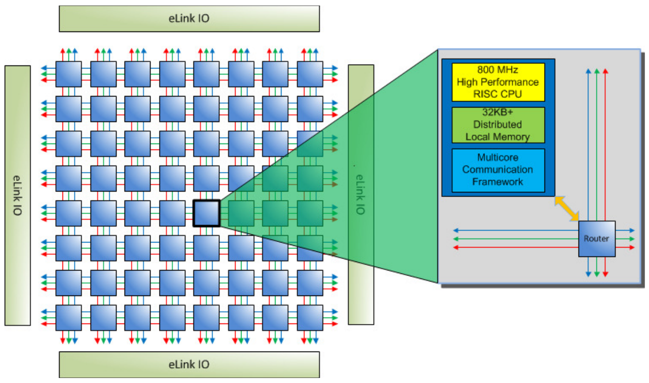
eLink IO
eLink IO
eLink IO
eLink IO
800 MHz High Performance RISC CPU
32KB+ Distributed Local Memory
Multicore Communication Framework
Router

Figure 13A. MPPA Processor: Architecture

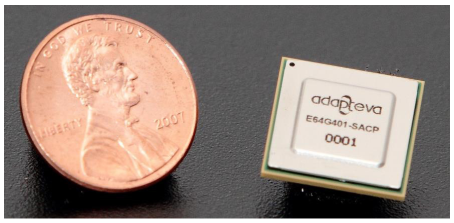

Figure 13B. E64G401 Processor: Fabricated Chip

Figure 13. Adapteva Epiphany: E64G401 Processor [10].

3.7 Additional multi-core processors and systems

In this sub-section, the general trend as seen in the industry over couple of year for multi-core processors/systems is shown with the help of Figure 14 [11]. This figure shows that there are several companies which are commercially building multi-core systems and to achieve higher performance integrating higher number of cores/processors in their systems.

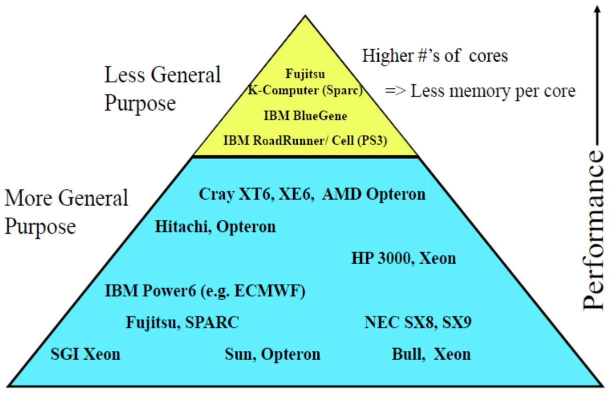

Figure 14. Treads in Multi Core Companies [11]

4. EXPERIMENTAL RESULTS

In this section, the experimental results are presented to show that *as number of cores/processors in the hardware increases the performance increases and execution time of the parallel program reduces*. For this experiment, open source parallel program to count the primes numbers is considered [12-14]. This program gives the flexibility to define the number of nodes in the system and based on the input, it generates parallel code for those number of nodes. Execution of this parallel program was performed on 3 nodes/core Raspberry Pi cluster [15] as shown in Figure 15.

Three different experiments were conducted. Experiment # 1 was to count prime number less than 100M, Experiment # 2 was to count prime number less than 200M and Experiment# 3 was to count prime number less than 300M. For all of these experiments, parallel program was generated and this code was run on Raspberry cluster by enabling only 1 node, 2 nodes and 3 nodes each time. Obtained results as shown in Figure 16 clearly shows that as number of the nodes/cores increased the execution time reduced and speed up of 3X is achieved by the three nodes/cores as compared to the single node/core.

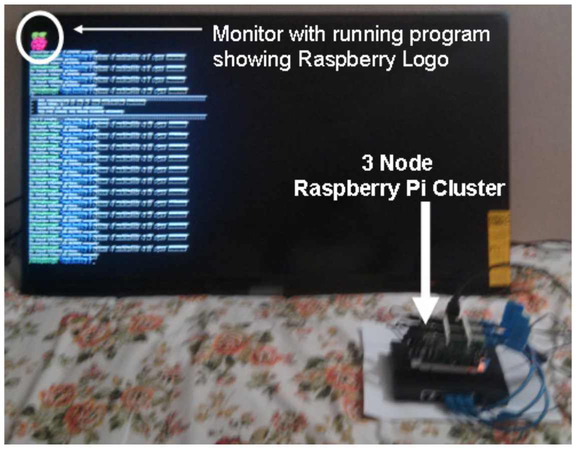
Monitor with running program showing Raspberry Logo
3 Node Raspberry Pi Cluster

Figure 15. Experimental Setup

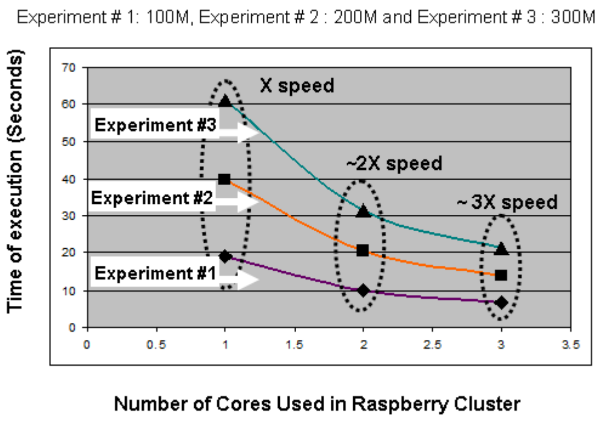

Figure 16. Experimental results to show the performance improvement by multi-cores.

5. DISCUSSIONS

5.1 Speed-up by only parallel program

It is very well known that as the number of cores/nodes increases in the system the execution time of the parallel code reduces. Also this has been validated in the previous Section 4. However, in the literature a very little analysis has been done on the effect of introducing more parallelism in the program than it can be supported by the hardware.

In the sub-section, experiment has been preformed to analyze the effect of more parallelism in the program than the hardware can support. For this experiment, parallel code was generated to count the primes numbers less then 100M as done in the previous Section 4. In this experiment, it was assumed the Raspberry PI cluster [13] has 21 nodes/cores and parallel program was generated for nodes in the system = 3, 6, 9, 12, 15, 18 and 21. The generated programs were executed on the same 3 Node Raspberry PI cluster as the previous Section with all 3 nodes/cores enabled. The obtained results are summarized in Figure 17.

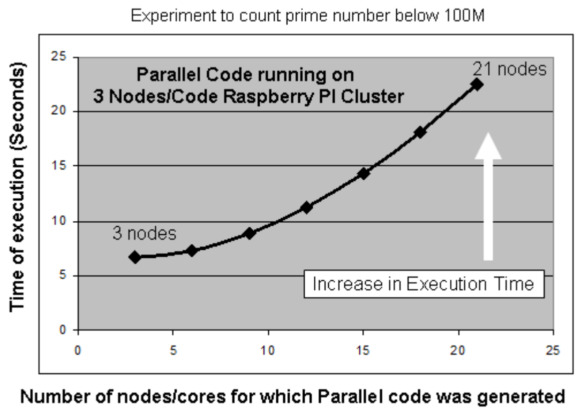

Figure 17 : Analysis to show performance effect when parallelism was introduced in program "software" while no more parallelism/cores in hardware was introduced.

It is important to notice from the results as shown in Figure 17 that the parallel *code execution time increased even when more parallelism was introduced in the program*. This can only happen because of extra overhead in executing super-parallel code when hardware resources are limited. For example, in this experiment, the execution of 21 nodes/cores parallel program on the 3 nodes/cores system. Thus, it can be concluded that for achieving realistic speed up, parallelism in hardware should also be available and by just making parallel program does not always necessarily increases the speed.

5.2 Homogeneous vs. Heterogeneous Cores Architects and Parallelism

Currently, many people are debating whether the cores in a multicore environment should be homogeneous or heterogeneous and there are no conclusions or definitive answers [3]. Homogenous cores are all exactly the same: equivalent frequencies, cache sizes, functions, etc. However, each core in a heterogeneous system may have a different function, frequency, memory model, etc. There is an apparent trade-off between processor complexity and customization. Homogeneous cores are easier to produce since the same instruction set is used across all cores and each core contains the same hardware. But are they the most efficient use of multicore technology ?. There is no conclusion on that.

While, each core in a heterogeneous environment could have a specific function and run its own spe-cialized instruction set. Building on the CELL example, a heterogeneous model could have a large centralized core built for generic processing and running an OS, a core for graphics, a communications core, an enhanced mathematics core, an audio core, a cryptographic core, and the list goes on. This model is more complex, but may have efficiency, power, and thermal benefits that outweigh its complexity. With major manufacturers on both sides of this issue, this debate will stretch on for years to come; it will be interesting to see which side comes out on top.

But parallelism is not the answer for everything. There are some added costs associated with parallelism. Synchronization between parts of a program executed by different processors is an overhead of parallelism that needs to be managed and kept at a minimum. Also, administering a parallel computing environment is more complicated than administering a serial environment.

6. SUMMARY/CONCLUSION

In this class-project paper, overview of the multi-core processors/systems is presented. In addition to discussing the state-of-art, the several experiments have been performed. In these experiments it is demonstrates that as the number of cores/processors in the hardware increases, the performance increases and execution time of the parallel program reduces. Moreover, it is shown when hardware resources are limited the parallel code execution time may increase.

7. ACKNOWLEDGEMENTS

Author would like to thank Edward Ciotic, student at San Jose State University, for allowing experimentation on his personal 3 node Raspberry PI cluster. Author would also like to thank Prof. Robert Chun for his useful feedback. Last but not least, author would like to thanks all people and researches for their previous work in the area of multi-core processors/systems, as their documents helped a lot in putting things together in this paper.

8. REFERENCES

[1] http://oreilly.com/catalog/oraclepp/chapter/ch01.html

[2] Prof. Robert.Chun. CS159 Class notes of SJSU University

[3] Bryan Schauer, “Multicore Processors–A Necessity”, www.csa.com/discoveryguides/multicore

[4] http://www.google/com/imghp

[5] https://noggin.intel.com

[6] http://en.wikipedia.org

[7] D. Pham, et al, “The design and implementation of a first-generation CELL Pocessor”, ISSCC 2005

[8] http://www.tilera.com

[9] http://www.kalray.eu

[10] http://www.adapteva.com

[11] G.Mozdzynski, “Concepts of Parallel Computing”, March 2012.

[12] http://www.github.com

[13] http://www.slac.stanford.edu

[14] http://www.mpitutorial.com

[15] Edward Ciotic, “Raspberry PI”, San Jose State University, CS 159 Class Project 2014.