How Variation of Parallelism is Exploited to Increase Performance and Efficiency

Wang Peiyu

George Washington University

Introduction

Parallelism has long been exploited by processor designers and researchers in order to achieve high-performance computing. Nowadays, with the appearance of big data and more and more advanced technologies like Machine Learning, Cryptocurrency and other cutting-edge techniques, more and more data needed to be processed, Machine Learning Algorithms need to be trained on larger and larger data sets, blockchain needs its security and privacy maintained with real-time generation to fill up and extend the blockchain. Efficiency, power/energy consumption, and consequently heat generation of the machine have become more and more of a concern in recent years, which made parallel computing the dominant paradigm in computer architecture. In this paper, different forms of parallelism for performance boosting and power/energy saving purposes have been reviewed, discussed and compared. This paper, however, focuses more on the review and comparison of different parallelisms, the advantages and disadvantages to a certain type of parallelism and how the disadvantages can be mitigated with the incorporation of some other techniques.

Keywords: Parallelism, Instruction Level Parallelism, Performance, Efficiency

Parallelism and the Different forms

One of the major designing goals of computer system architecture is great performance and/or efficiency. Researchers and designers are trying to look for ways that would enhance the average performance of the computer systems. Legendarily, clock speed, or the rate at which basic computer operations are performed, is used to measure the performance increase. However, increasing clock speed is hard. At the same time, new materials are no longer expected to significantly drive clock speed up (Brandalero & Beck, 2016). In this case, clock speed might not be adequate for the measurement of system performance anymore. For example, for highly concurrent systems, actual delivered performance is program and algorithm specific (Kuck, Davidson, Lawrie, & Sameh, 2019). Even though we can use one way or another to measure the peak performance of a designed system, actual delivered performances are often much lower than the peak performance rates. Peak performance rates are only achievable when "hand optimization and assembly language programming are used on well-suited programs." Said Kuck et al.. While the most accurate way of measuring performance changes over time, there are some widely known and consistently used concepts in the field to help improve the performance. One of those concepts is Instruction Level Parallelism. How much Instruction Level Parallelism can increase the performance versus if it is worth researchers spending so much time and energy doing researches on has long been a hot and debatable topic. Researchers like Thomas Gross of Carnegie-Mellon University said that even if you were able to get a 20-times speed improvement over a wide range of programs (which according to multiple studies, is not proved. On the other hand, studies show that Instruction Level Parallelism can bring a maximum overall improvement of 2 to 8 times and even that may be difficult to obtain and maintain), it would still be small compared with the improvement you could get with multi-processor parallel computing

(Freedman, 1991). David, Gelernter, A Yale University Computer Scientist who researches ways of linking workstations in parallel networks, said that Instruction Level Parallelism should be seen not as an alternative to general purpose parallel computing but as an adjunct to it (Freedman, 1991). However, Fisher argued that for a user with access only to a single workstation, Instruction Level Parallelism was the only road to parallelism (Freedman, 1991), which specifies the importance and necessity of researches being conducted on Instruction Level Parallelism. The vision behind Instruction Level Parallelism, is that a single microprocessor is built which would act like a parallel computer but uses conventional code. In Instruction Level Parallelism, "a single, specially designed processor simultaneously runs different tiny chunks of a standard program." Said Freedman (1991).

Instruction Level Parallelism is a way, in which, with the help of pipelining, the executions of instructions are overlapped to help improve performance, or according to the paper "*Potential Analysis of a Superscalar core employing a reconfigurable array for improving Instruction-Level Parallelism*", reflects how often processor instructions can be executed concurrently (Brandalero & Beck, 2016). The reason why Instruction Level Parallelism can improve performance is because the executions of the overlapped instructions are concurrent, which means that instead of the older and more traditional way of executing instructions, which is executing instruction one at a time in their original sequence, Instruction Level Parallelism takes several instructions in at a time and execute all of them concurrently (Brandalero & Beck, 2016). A conventional microprocessor, by contrast, executes a program's instructions one at a time in an order strictly dictated by the program (Freedman, 1991). The problem with the conventional microprocessor is that, any single instruction barely needs the chip's full capabilities, which means that most of the chip's capabilities just stand idle most of the time. Instruction Level

Parallelism, however, makes use of some of the wasted power by asking the chips to carry out multiple instructions at the same time. In this case, before the Instruction Level Parallel chip knows the results of the operations, it can go ahead and do some speculations and follow one of the paths, if it turns out to be the wrong one, then the chip simply discards the results and starts new with the correct path (Freedman, 1991). However, "a sufficiently powerful Instruction Level Parallel chip can speculatively execute both paths in order to cover its base," argued Freedman (1991), which means that with the idle power put into use and both paths executed, the chip will take the right path along with the wrong path every time, but as long as the power can cover the cost of executing both paths, even though that means that one of them has to go to waste every time, the chip actually can make sure that it will execute the right path every time to increase the performance.

In the paper "*Potential analysis of a superscalar core employing a reconfigurable array for improving instruction-level parallelism",* it talks about the major limitation of Instruction Level Parallelism, which is the natural upper bound to the amount of Instruction Level Parallelism available from application given data dependency which is discussed previously. Because of the upper bound imposed, even though it can barely be reached by modern processor designs, it actually takes huge area and power increases to only achieve marginal gains in performance. In order to do better in this situation, the authors Brandalero and Beck (2016) proposed caching the entire dynamic scheduling of instructions in the loop by employing a reconfigurable array. The reconfigurable array uses a matrix of functional units, which is then used to implement the hotspots which can be the most frequently executed loops. Those code sequences then can be executed right in the circuit, which can eliminate intermediary registers and dependency checks among instructions, and this could improve performance and energy usage (Brandalero & Beck,

2016). Because the system they proposed is entirely dynamic, no compiler support is demanded, and because the proposed system would cache decoded instructions ahead of time onto the pipeline, no fetch and decode instructions for the same instruction or code sequence is needed anymore, which could not only boost performance but also save energy.

Data Dependency and Data Hazards

However, in order to execute the instructions concurrently, the machine will need to make sure that there are no data dependencies and hazards in the instructions being executed concurrently, or, in other words, the instructions are parallel, so that the executions can be done without any stalls.

Data hazards is a big issue when it comes to parallelism because when data hazards happen, pipeline has to be stalled, which means that, during the time, instructions further down the pipeline are delayed, no new instructions are fetched till the stall is cleared and the data hazard is resolved. This means that, if data dependencies or data hazards happen, the advantages gained by deploying parallelism are lost, and what is worse is that, when data hazards happen, not only the efficiency or performance is reduced, but the result given back might also be completely wrong, and that is why, “the real cost of Instruction Level Parallelism comes in identifying instructions that are independent of earlier-results – hence free to be executed ahead of time.” Said Freedman (1991). However, data dependency is hard to detect out of reasons like data values may flow among instructions through registers or memory locations, or even though the instructions share the same registers or memory locations, there actually is no data flow associated with the registers and/or memory locations with the same name among the instructions (This kind of dependency is actually called name dependency). Name dependency, despite how it looks or sounds, actually will not cause data hazard if being executed concurrently, but it could be easily

mistaken for data dependency. In order to avoid data hazards, preserving data flow and making sure that the actual data flow of values between instructions producing results which would be used by later instructions are very important.

Two Approaches and Pipelining to Instruction Level Parallelism

In order to make the best use of Instruction Level Parallelism, Computer Systems Architecture generally adapts two approaches or a combination of both. One of them is the dynamic approach, which depends on the hardware to locate the parallelism, or the Instruction Level Parallel chip itself "looks ahead as it works, scanning upcoming code for instructions that can safely be pulled down for early execution." According to Freedman (1991). The other is the static approach, which is a fixed solution generated by the compiler. "As the compiler translates the high-level language of the software into the language of the chip, it can search out interdependencies and flag them for the chip." Said Freedman (1991). To make things even simpler, with pipelining, the processor and the compiler look no further ahead than the next instruction, and if that instruction does not depend on the result of the ongoing one, the processor starts working on it early. By overlapping the executions of two or more successive instructions, pipelining alone can double the speed of a processor (Espasa & Valero, 1997).

Data Level Parallelism

Besides the Instruction Level Parallelism which is one of the approaches to high-performance computer in computer architecture, there is another approach called Data Level Parallelism. While Instruction Level Parallelism approach seeks to execute several instructions per cycle by exploring a sequential instruction stream and extracting independent instructions to send to several executing units in parallel, the Data Level Parallelism uses vectorization techniques

(Espasa & Valero, 1997). A vector instruction specifies a series of operations to be performed on a stream of data and each operation performed on each individual element is independent of all others, and therefore, a vector instruction is highly parallel and can be easily pipelined (Espasa & Valero, 1997). In the paper "*Exploiting Instruction- and Data-Level Parallelism*", the authors Espasa and Valero (1997) proposed merging Instruction Level Parallelism and Data Level Parallelism in a single-processor architecture that combines three key technologies: 1. Vector Instructions; 2. Out-of-order execution with register renaming; 3. Simultaneous multithreaded execution to further enhance the performance of the Computer System Architecture. The authors Espasa and Valero (1997) believed that this combination yielded a simultaneous multithreaded vector, or SMV architecture. Vector instruction sets and vector architecture are an excellent match for the characteristics of Data Parallel codes and because vector instruction sets use fewer processor control resources and they directly and easily convey parallelism to the hardware, thus, the vector instruction set is the basic building block for high performance. The reason why they are trying to combine Instruction Level Parallelism and Data Level Parallelism into their proposed model is that, when there are large memory latencies, the Data Level Parallelism models suffer severe performance degradation, and the Instruction Level Parallelism model's out-of-order execution and register renaming could help resolve this issue. Historically speaking, useful Instruction Level Parallelism techniques like out-of-order execution and register renaming, which "fight memory latency and improve processor throughput in the microprocessor world," said Espasa and Valero (1997), had never been used in commercial vector computers. Out-of-order execution reorders instructions and reduces the interference between computation and memory instructions while reordering makes better use of the memory ports and masks memory latency. Register renaming, on the other hand, helps manage the processor state and

provide precise exceptions, which is always difficult in vector processor. In addition, renaming could also improve the use of the register pool.

How Data Level Parallelism can help better Instruction Level Parallelism is that, when it reaches the point where a program's intrinsic Instruction Level Parallelism limits the number of function units that can be used simultaneously, multithreading comes to the rescue. By multiplexing several threads onto the same processor, all resources can be fully utilized. Simultaneous multithreading indicates that there are no explicit context switches between threads, rather, instructions from different threads can coexist at the same time in the order buffer (Espasa & Valero, 1997). Simultaneous multithreading logically follows in a Data Level Parallel machine that already has out-of-order execution and register renaming. In this situation, just a few bits of thread information in the instruction queues and per-thread rename tables are enough to achieve the desired SMV architecture (Espasa & Valero, 1997).

As for the challenges and/or concerns faced by researchers in building billion-transistor processors, which include wire delays, processor-to-memory performance gap and parallelism inside a program, one scalar instruction from the Instruction Level Parallelism paradigm is compared with the one instruction from the Data Level Parallelism paradigm.

A scalar instruction requires many stages to be processed yet specifies only a very simple operation, while the vector instruction has a few setup and shutdown stages but have many more useful stages involving actual computation (Espasa & Valero, 1997). In a scalar instruction, most of its execution phases are devoted to book-keeping activities required to maintain the processor's consistency. Adding more execution resources to a superscalar processor requires many extra control resources (Espasa & Valero, 1997). On the contrary, in a Data Level Parallel processor we can speed up vector computation simply by replicating functional units.

Data Level Parallelism lets us reduce the number of instructions by making each vector instruction longer, even though that means that each vector instruction performs many operations and can take many cycles to complete (Espasa & Valero, 1997). A vector program can specify many fewer address computations, loop counter increments and branch computations because they are normally very implicit in vector instructions (Espasa & Valero, 1997). On average, the Data Level Parallel paradigm performs much better than that of an Instruction Level Parallel paradigm.

When it comes to wire delays, it takes an increasing amount of today's microprocessor cycle time. Researchers believe that high performance can only be achieved by replication of very fast building blocks that work more or less asynchronously. Instruction Level Parallel processors have clear scalability problems when the wake-up and select logic is needed in wide issue. Machines scales unfavorably when designers decrease feature size (Espasa & Valero, 1997), and that is where Data Level Parallelism comes to the rescue. Compared to the Instruction Level Parallel model, the Data Level Parallel model supports independent, replicated building blocks very well, for the operations inside the same vector that do not have information flow among each other, they can be divided into independent sub-blocks that can be executed independently since they do not need to communicate with each other.

As for the other challenge, memory access time, the general sense is that, the larger the difference between CPU and memory speed, the longer it takes for each memory instruction to complete. Scalar instruction sets have severe shortcomings when high memory latencies are taken into consideration. Vectors, on the contrary, have inherent advantages in memory usage. Since a single instruction can precisely specify a long sequence of memory addresses, the hardware has considerable advance knowledge regarding memory references, in which case it

can schedule these accesses efficiently and need access no more data than required. In addition to that, vector memory operations can mortise start-up latencies over a potentially long stream of vector elements (Espasa & Valero, 1997). Researches have shown that, performance only slightly degraded even when there are 100 cycles of main memory latencies when combining Instruction Level Parallel techniques and Data Level Parallel engines. Also, a Data Level Parallel machine can use whatever amount of memory bandwidth it has more effectively and efficiently because it can request multiple data items with a single memory address (Espasa & Valero, 1997).

When it comes to parallelism inside the program, which is the last but not the least challenge researchers are facing in building billion-transistor processors, if designers want to achieve better performances, they would need to extract large amount of operation parallelism. In order to exploit the advantages associated with Instruction Level Parallelism, hardware techniques are applied to discover instruction parallelism. At the same time, machines exploiting Data Level Parallelism relies on the compiler to discover parallelism and convey that with vector instructions to the hardware (Espasa & Valero, 1997). In this sense, exploiting Instruction Level Parallelism and Data Level Parallelism is not in conflict with each other, and for better machine performance, it is better for researchers and designers to exploit as much and as many forms of parallelism as possible.

Cedar Supercomputer and Pipelining

In the paper "*Parallel Supercomputing Today and the Cedar Approach"*, the authors Kuck et al. mentioned that technology was no longer reliable when it came to the improvement of the system speed, due to the dramatic shift from uniprocessors to multiprocessor systems, which also meant a much tighter coupling in architecture, hardware, software and application development

expertise. Even though, with multiprocessor, a different job can be run simultaneously on each processor, which could reduce the waiting time and improve single-job turnaround time. Improving single-job turnaround time can only be achieved by parallel processing, which means using parallel algorithms and restructuring the code of a single job to spread it over a number of cooperating processors (Kuck, Davidson, Lawrie, & Sameh, 2019). In order to reduce the gap between the actual delivered performance and the peak performance of the supercomputer system, which is considered the most critical in supercomputing today, Kuck et al. believed that automatic software restructuring and the use of parallel algorithms were the keys. Supercomputer processor's performance is boosted by pipelining arithmetic functions and employing vector instruction and multiple function units. Sometimes multiple identical vector units are deployed in order to increase peak performance. At the same time, the use of multiple identical vectors adds significantly to the start-up overhead cost since vectors are broken into smaller pieces for each vector unit (Kuck, Davidson, Lawrie, & Sameh, 2019). This means that, the more procedure required, the longer, denser array operations with regular memory addressing are demanded to approach peak performance. This indicates that the performance gained by pipelining and utilizing vector instructions and functions units is then limited by the declining returns from the increasing number of pipeline segments (if the number of pipeline segments exceeds six or eight), the inability to concurrently utilize more than two function units on average and the large start-up vector instruction overhead cost (Kuck, Davidson, Lawrie, & Sameh, 2019). However, If irregular memory addressing prevails, which happens often for sparse matrix operations, the performance will degrade. In order to tackle this key problem and reduce the gap between the actual delivered performance and the peak performance, the Cedar supercomputer proposed was made of a few clusters but with a globally shared memory. The parallel systems provided by the

clusters then did not require that algorithms to be as much uniform while the execution efficiency remained the same.

Horizontal Parallelism and Vertical Parallelism for Greater Performance

Moreover, with the development of technology nowadays, not only performance is the critical concern for embedded systems and portable devices anymore, so is energy efficiency. Even though multi-issue processors can make use of Instruction Level Parallelism to boost the performance, it comes at the cost of energy and power consumptions. In order to maintain the high performance while cutting down the energy and power consumption, in the paper "*Orchestrating Horizontal Parallelism and Vertical Instruction Packing of Programs to Improve System Overall Efficiency*", the authors Lin and Fei (2009) proposed to "apply the instruction register file IRF techniques from single-issue processor to VLIW architecture," in which case, the most often executed instructions are chosen and placed in the IRF which is on-chip, so that they can be accessed faster and with less energy consumption. The enhanced VLIW architecture is therefore able to "orchestrate the horizontal instruction parallelism and vertical instruction packing for programs to improve system overall efficiency," said Lin and Fei (2009), and as the name of the paper suggests. In their paper, Lin and Fei (2009) mentioned that Instruction Level Parallelism was considered horizontal parallelism, which was exploited to reduce the pressure on system clock frequency increase. They mentioned that the researches that had been done on the IRF techniques already showed that by placing the most frequently executed instructions in the on-chip IRF or multiple entries in the IRF, the number of instructions fetched and the program energy consumption both significantly decreased. However, those researches did not exploit Instruction Level Parallelism, which means that, by integrating the IRF techniques and Instruction Level Parallelism, the performance can potentially be better while the energy and

power consumptions stay low. In order to solve their concern of violations of original ILP when introducing simple slot-based sub-instruction packing, they introduced an enhanced ISA which had all the IRF-related instructions classified into two hierarchy levels.

Talking about Computer System Architecture might sound abstract and distant to Computer Scientist, for computer architecture is at a lower level which makes it harder for computer scientist to relate. Computer Scientist, most of the time, think more about the algorithmic performance and/or efficiency by trying to reduce the running time of the algorithms and the space used by the algorithms. But as the paper "*A Faster Integral Image Computing Hardware Architecture with High Power and Area Efficiency*" pointed out, it might actually not be as abstract and distant as it sounds. When executing vision applications, intensive integral image computing and frequent memory accessing is often required. In order to make Integral Image Computer and the data flow of the computation faster, a dual-direction data-oriented integral image computing was proposed along with a pipelined parallel architecture designed to support this mechanism. As Computer Vision becomes a more and more popular research topic, and is used in many fields today, the efficiency of the integral image computing also becomes more and more of a concern in addition to the performance of the algorithm. (meaning the accuracy and/or other measurements or indicators the computer scientists choose) As Ouyang et al. (2015) pointed out in the paper, "in embedded vision application such as object detection in automotive systems, biomedical systems, and some portable systems, real-time processing is required with a limited power budget and hardware implementation size." The authors thought that if the parallelism in the computation was fully deployed and exploited, a faster speed of computation could be achieved. Of course, in order to exploit the parallelism, data dependencies should be fully analyzed to prevent data hazards from happening. In their study, they used affine loop

transformation to transform the original loop into a new form. Their results showed that "the maximal parallelism is m as the iterations in the red circle have no dependence between each other and can be processed in parallel." Therefore, they suggested the dual-direction (which includes both row direction and column direction) data-oriented method for parallelism exploitation.

At the same time, because they designed the strip-based unique memory architecture in the CU and exploited high parallelism for the computing unit, the power and hardware cost per operation in integral computing were largely reduced, and the power efficiency and area efficiency were related to processing speed, power and hardware resource. They were also able to boost the power efficiency and area efficiency by carefully choosing the w in their CALU design, which was exactly because their parallel architecture improved the data reuse and reduced the memory access cost per operation, that the power and area efficiencies were high (Ouyang, Yin, Zhang, Liu, & Wei, 2015).

Conclusion

This paper mainly focuses on Instruction Level Parallelism and talks about the advantages of Instruction Level Parallelism, which include executing concurrently so that it can boost the performance and efficiency and etc., and disadvantages of Instruction Level Parallelism, which include data dependencies, data hazards that might happen because of careless design and execution of code sequences and etc.. This paper also compares some other forms of parallelism, like Data-Level-Parallelism, and listed some of the advantages and disadvantages they each has, and how a better performance can be achieved by combining them both.

In addition, new processor designs are proposed which make great use of Instruction Level Parallelism while including some other techniques to mitigate the disadvantages Instruction

Level Parallelism has, in order to further boost the performance, while cutting back on energy and power consumption to improve efficiency. The importance of parallelism is also discussed in the paper.

Bibliography

Brandalero, M., & Beck, A. S. (2016). Potential analysis of a superscalar core employing a reconfigurable array for improving instruction-level parallelism. *Design Automation for Embedded Systems, 20(2)*, 155-169.

Espasa, R., & Valero, M. (1997). EXPLOITING INSTRUCTION- AND DATA-LEVEL PARALLELISM. *IEEE Micro, 17(5)*, 20-27.

Freedman, D. H. (1991). A Painless Route to Parallel Computing? *Science, 253(5025)*, 1209-1209.

Kuck, D. J., Davidson, E. S., Lawrie, D. H., & Sameh, A. H. (2019). Parallel Supercomputing Today and the Cedar Approach. *Science, 231*(4741), 967-974.

Lin, H., & Fei, Y. (2009). Orchestrating Horizontal Parallelism and Vertical Instruction Packing of Programs to Improve System Overall Efficiency. *IEEE Transactions on Computers, 58(9)*, 1211-1220.

Ouyang, P., Yin, S., Zhang, Y., Liu, L., & Wei, S. (2015). A Fast Integral Image Computing Hardware Architecture With High Power and Area Efficiency. *IEEE Transactions on Circuits and Systems II: Express Briefs, 62(1)*, 75-79.