Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/42/20230775

High-performance computing: Transitioning from Instruction-Level Parallelism to heterogeneous hybrid architectures

Mingtao Zhang

School of Computer Science and Technology, Harbin Institute of Technology, Harbin, 150006, China

nullptr@stu.hit.edu.cn

Abstract. This paper delves into the shift from Instruction-Level Parallelism (ILP) to Heterogeneous Hybrid Parallel Computing in the quest for optimized performance processing. It sheds light on the constraints of ILP, emphasizing how these shortcomings have catalyzed a move toward the more adaptable and proficient framework of heterogeneous hybrid computing. This transformation's advantages are explored across diverse applications, notably in deep learning, cloud computing, data centers, and mobile SoCs. Additionally, the study underscores emerging architectures and innovations of this era, including many-core processors, FPGA-driven accelerators, and an assortment of software tools and libraries. While heterogeneous hybrid computing offers a promising horizon, it isn't without challenges. This paper brings to the fore issues like restricted adaptability, steep development costs, software compatibility hurdles, the absence of a standardized programming model, and vendor reliance. Through this in-depth exploration, our aim is to present a holistic snapshot of the present and potential future of high-performance processing.

Keywords: High Performance Processing, From Instruction-Level Parallelism, Heterogeneous, Hybrid Parallel Computing.

1. Introduction

The unwavering quest for advanced performance in processing has consistently propelled computer architecture's evolution. From the inception of single-core processors to the rise of multi-core and many-core variants, this journey has been punctuated by ceaseless innovation and adaptability. Central to this progression has been the harnessing of parallelism at diverse scales. Instruction-Level Parallelism, one of the initial forms of parallel computing, was pivotal in bolstering computational velocities by facilitating concurrent instruction execution. However, the inherent constraints of ILP, such as challenges in inter-thread communication, the nuances of thread synchronization, and the delicate balance between energy and performance, have prompted a deeper exploration into novel architectural models. This paper delves into the transition from ILP to heterogeneous hybrid parallel computing, shedding light on architectural specialization and its practical applications in domains like deep learning, cloud computing, data centers, and mobile System on Chips. Moreover, we discuss prevailing

© 2023 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

architectures and technologies capitalizing on this paradigm shift and underscore the hurdles to be surmounted for broader acceptance and refinement.

2. Limitations of Instruction-Level Parallelism

2.1. Definition and Explanation

Instruction-Level Parallelism is a technique employed in computer architecture to enhance computational speed. By optimizing the use of processor resources and speeding up the clock cycle, ILP significantly boosts overall system performance.

There are two primary methodologies to realize instruction-level parallelism, which can either function independently or in tandem [1]: The first method encompasses the simultaneous dispatch of multiple instructions. This demands a diverse set of functional units like adders and multipliers, each tailored for autonomous instruction execution. The realm of multiple issue machines can largely be bifurcated into superscalar and Very Long Instruction Word (VLIW) machines. Superscalar machines rely on hardware to dynamically decide which instructions to dispatch during runtime. In contrast, VLIW machines lean on compilers to predetermine these decisions. The second approach, termed instruction pipelining, segments individual instructions into a cascade of stages, each processed in a set sequence. A standard instruction might be broken down into stages such as decoding, data fetching, execution, and results write-back. As the first instruction transitions from the initial to the subsequent stage, a new instruction can embark on the first stage. This setup allows instructions to overlap partially. In essence, an n-stage pipeline can concurrently execute up to n instructions, with each at a distinct stage. Adding more stages fosters finer overlap, leading to a swifter clock cycle.

2.2. Discussion on the Limitations

In the field of computer architecture, ILP has been a critical factor in the pursuit of improving computational performance. This technique has been the focus of numerous research efforts and has resulted in significant enhancements in processor speed. However, the theory and application of ILP are not without constraints. This section aims to provide a brief discourse on the limitations of ILP. Understanding these boundaries offers a realistic perspective on ILP's capabilities and may inspire novel approaches to overcome these limitations.

The study of Ehsan et al. indicates that in the case of certain applications that involve extensive inter-thread communication, each stage is significantly reliant on the one before it, which limits ILP [2]. Thread synchronization semantics can also significantly influence program performance, which could lead to slowdowns of up to six times the original speed. As programs scale to incorporate more cores, the performance overhead associated with thread synchronization semantics becomes increasingly evident.

Vladimir et al. examined the implications of critical infrastructure applications, including in-memory databases [3], key-value stores, and graph analytics, on hardware resources. Such applications are defined by extensive working sets with multi-level address redirection and pointer navigation. Their findings suggest that multi-level caches and branch predictors fall short in keeping processor delays at a minimum. Additionally, out-of-order windows with hundreds of instructions, also fail to accommodate all the required instructions to uphold a substantial number of simultaneous memory requests, which is crucial for concealing extensive latency accesses.

Another challenge is the trade-off between increasing demand of energy and smaller performance acceleration. For single-thread performance, architects are forced to accept increasing complexity and resource requirements with diminishing performance returns due to the constraints of Complexity Wall and ILP Wall [4]. For multicore performance, the failure of Dennard scaling and the slowdown of Moore's Law limit the benefit of performance scaling from adding more cores [5].

3. The Shift Toward architectural specialization

3.1. Explanation

As the marginal gains in performance from enhancing general-purpose processors diminish in comparison to the escalating resource expenditure, a growing number of platforms are electing to design Application-Specific Integrated Circuits (ASICs) at the cost of flexibility. This strategy aims to attain superior performance within identical power constraints.

Analysis of Ehsan et al. reveals significant differences on performance behaviors among benchmarks, and anticipates that no single, uniform micro-architecture will deliver optimal performance across all scenarios, highlighting the need for designs that are reconfigurable and diverse. Simultaneously, as the number of cores in a system continues to scale up, it is increasingly likely that load imbalances will occur. This imbalance can lead to certain threads experiencing more pressure, making them more critical than others. This situation suggests the need for heterogeneous designs, where threads are mapped to high-performance cores when they become critical. This approach can potentially optimize the system performance by ensuring that critical tasks are assigned to the most capable resources.

Hence, domain-specific accelerators are one of the solutions to continue to improve performance and efficiency. A domain-specific accelerator is a hardware computing engine that is specialized for a particular domain of applications. Compared to general-purpose processors, these accelerators can provide exponential improvements in performance-to-cost and performance-to-power ratios. Domain-specific accelerators enhance performance and efficiency through four key techniques: data specialization, which uses specialized operations on domain-specific data types for faster processing; parallelism, utilizing high degrees of parallelism at multiple levels for performance gains; local and optimized memory, storing key data structures in numerous small, local memories for high memory bandwidth at low cost and energy; and reduced overhead, where specialized hardware minimizes the overhead of program interpretation [6].

Over the past 15 years, accelerators and architecturally diverse systems have been drawing more attention both in academia and industry. By utilizing different accelerators for various parts of an application, higher efficiency can be achieved in comparison with on a single general-purpose processor [7].

3.2. Deep Learning

In the field of deep learning, heterogeneous computing is becoming increasingly popular. Traditionally, Graphics Processing Units (GPUs) are widely used in accelerating neural network computations. Initially, GPUs were designed to accelerate three-dimensional graphics especially in computer games. In September 2014, NVIDIA released cuDNN, a library of GPU-accelerated primitives specifically for deep neural networks (DNNs). This library implements supports for functions such as forward and backward convolution, pooling, normalization, and activation layers.

The architectural support provided by GPUs for training neural networks has proven to be notably effective in comparison with on general-purpose CPUs. This is because matrix multiplication is the most computationally intensive part of neural networks [8], which can be processed in parallel. GPUs are highly effective for parallel deep neural network computations due to their abundant resources, faster memory bandwidth, and acceleration of massive matrix multiplications [9].

In recent years, heterogeneous hybrid architectures have been applied in accelerating deep learning kernels. Compared to GPUs, these types of architecture have unique advantages. Typical on-chip communication infrastructure used on traditional multicore platforms cannot effectively manage CPU and GPU communication requirements simultaneously [10]. Also, the existing manycore platform's metallic electrical interconnections may not be sustainable in handling massive increase in bandwidth demand driven by big data AI applications [11]. In the study of Eriko et al. [12], recent innovations on DNNs introduce irregular parallelism on custom data types. These data types could be easily handled by Field-Programmable Gate Array (FPGA) due to its customizability, which are difficult for GPUs instead.

3.3. Cloud Computing and Data Center

In the past few years, FPGAs have been used for a variety of different applications in cloud computing and data centers. One of the earliest examples was the Catapult fabric at Microsoft used for accelerating Bing searches, whose large-scale reconfigurable structures improved the throughput and latency of each server under high load [13].

Cloud FPGAs can be deployed at the level of Infrastructure as a Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). It allows for multiple tenants to deploy hardware-accelerated workloads independently over the same pool of shared FPGA-enabled cloud node, which is useful in containerized and serverless environment [14]. The corresponding software ecosystems also enable the increased introduction of specialized hardware. The versatility and computational prowess of a reconfigurable fabric, in tandem with the high-speed I/O performance of ASICs, enable the deployment of FPGA-based systems in virtually any part of the cloud. These systems enhance the efficiency of computation, networking, and storage [15].

In the future, it is possible to build a fully disaggregated data center, where every type of resource is disaggregated [16]. With cloud applications becoming fine-grained and specialized and the disaggregation of computing resources [17, 18], heterogeneous architecture will be more appealing and utilitarian.

3.4. Mobile SoCs

Mobile SoCs typically integrate a variety of specialized hardware accelerators. The trend towards hardware specialization is transforming the role of CPUs in computational architecture. Tasks rich in parallelism are assigned to accelerators (for example, GPUs and image encoders/decoders), leaving the CPU to handle non-parallelizable code. This makes the CPU available to offload the ILP and locality of these mechanisms to specialized accelerators, leaving itself greater design and optimization space [19]. On the other hand, heterogeneous hybrid architectures allow to achieve lower energy consumption at the same level of efficiency [20, 21].

4. Popular Architecture and Technologies

4.1. Many-core Processors

Many-core processors generally have many more computing cores than CPUs and are commonly used as accelerators. Due to the difficulty in enhancing performance by increasing the clock frequency of the CPU, one solution is to integrate more parallel computing units on a single chip to leverage parallelism for performance improvement [22]. In comparison with multi-core CPU, a many-core CPU has dozens to hundreds of simpler cores with lower-frequency and sequential execution. Many-core processors usually adopt Non-Uniform Memory Access (NUMA) architecture to accelerate memory access, where each processor has its local memory and can have access to the other parts via the memory controller of other processors. This feature provides many-core processors some benefits:

Scalability: NUMA architectures can scale more effectively than other designs as the number of processors increases. This is because each processor (or a small group of processors) can have its own dedicated memory, which it can access more quickly than memory attached to other processors [23].

Avoidance of Bottlenecks: Because each processor or set of processors has its own memory, NUMA can help avoid bottlenecks that can occur when multiple processors attempt to access a shared memory pool simultaneously. This can help maintain system performance as the number of processors increases.

Concurrency: NUMA allows for higher levels of concurrency, as different nodes can process different tasks simultaneously without having to wait for shared resources.

4.2. FPGA-based Accelerators

FPGAs are integrated circuits designed to be configured by a user or a designer after manufacturing, hence the name "field-programmable". They are essentially programmable silicon chips that utilize a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGA-based accelerators are used in a variety of applications such as convolutional neural networks [24], cryptography [25], bioinformatics [26], computer hardware emulation and modern digital systems. The advantages of FPGA-based accelerators can be categorized into several key aspects:

Reconfigurability: Unlike ASICs, FPGAs are not limited to a predefined hardware function. They can be reprogrammed to take on a new role, supporting a new algorithm or application. This makes them highly adaptable to changing technological or application requirements.

Performance: FPGAs can deliver superior performance for certain tasks because they can be programmed to carry out complex computational tasks as parallel processes. This is in contrast to general-purpose processors, which execute instructions sequentially. For tasks that can be parallelized, FPGAs can offer significant performance advantages.

Power Efficiency: FPGAs are often more power-efficient than general-purpose processors for certain tasks. They can be programmed to carry out the task at hand with no more or less functionality than is strictly necessary, which can lead to significant power savings.

Longevity: Given their reconfigurability, FPGAs can "outlive" specific technologies. As new algorithms or processing techniques emerge, FPGAs can be reprogrammed to implement these, thus extending their useful life.

4.3. Software Tools and Libraries

Programming frameworks of heterogeneous parallel computing can be roughly divided into the following categories [27]:

Heterogeneous extensions of existing languages: This model represents a class of programming paradigms that augment existing programming languages with constructs and abstractions to manage and exploit heterogeneous computing resources. Examples such as CUDA and OpenCL extend traditional languages like C and C++ with additional syntax and APIs to control data placement and parallel execution on GPUs and other accelerators. This approach provides a low-level control to the programmer, allowing them to optimize code for specific hardware characteristics.

Directive-based Heterogeneous Programming: In this model, the heterogeneity of hardware is abstracted by the use of compiler directives. These directives, embedded within traditional programming languages, provide hints to the compiler about parallelization and distribution of computation. OpenMP is a prominent example, using pragma annotations to instruct the compiler about potential parallel regions. This model allows developers to maintain a single, portable source code, while the compiler manages the specifics of executing it on heterogeneous hardware.

Cooperative Programming with Container Models: This paradigm involves the decomposition of a complex application into a set of simpler, loosely coupled components. Each component is encapsulated within a software container, providing an isolated execution environment. Communication between components is typically achieved via network protocols. This model offers significant advantages in terms of scalability and resilience, as components can be independently scaled and failures are isolated. It's particularly relevant in the context of distributed, cloud-native applications and microservices architectures.

5. Challenges

While specialized accelerators excel at specific tasks, particularly in fields like artificial intelligence, graphics processing, data analysis, and scientific computing, they are not without their challenges. Despite their advantages, several limitations emerge in the context of heterogeneous hybrid architectures:

Limited Flexibility: Specialized hardware accelerators are tailor-made for optimal performance in specific tasks, often at the expense of versatility. This means that while they're efficient in their designated tasks, they lack the general adaptability of broad-purpose processors [28]. Their dataflow and the coupling between tasks and accelerators are typically rigid [29], which hinders any programmatic repurposing outside their designated specialties.

Constrained Optimization Space: The avenues to allocate computational tasks to fixed chip resources are finite, thereby capping the benefits attainable for a set field and budget. With the impending end of

CMOS scaling, which will place constraints on chip transistor budgets, the efficiency of these optimizations will plateau, leading to diminishing returns. In essence, accelerators will soon hit a sweet spot between computational proficiency and hardware optimization [30].

High Development Costs: Crafting and producing a heterogeneous architecture often comes with a hefty price tag, especially for specialized chips like ASICs. Such substantial investments can deter smaller enterprises and academic entities.

Software Compatibility: Harnessing the power of parallel computing on these architectures often demands tailor-made software. The need for this bespoke software can spark compatibility issues with existing platforms, making the integration of accelerators into current systems labor-intensive. It's not uncommon for development boards to come equipped with NPUs and other specific accelerators, but without the essential drivers and software, they remain underutilized.

Absence of a Unified Programming Framework: These diverse architectures necessitate varying programming paradigms or languages to truly shine. The fragmentation in coding practices complicates the development process, creating a daunting learning curve for newcomers. This lack of uniformity can retard the adoption of novel hardware accelerators and augment the intricacies of software maintenance and updates.

Vendor Dependency: Firms adopting hardware accelerators might find themselves tethered to their vendors. Any sudden business decisions, be it discontinuation, price hikes, or worse, business dissolution, can profoundly impact the firms relying on these accelerators.

6. Conclusion

In conclusion, the transition from singular Instruction-Level Parallelism to diverse hybrid parallel computing marks a pivotal evolution in the realm of high-performance processing. The constraints of ILP, combined with escalating complexity and resource demands, have driven this shift. Architectural specialization, manifested through the adoption of ASICs and domain-specific accelerators, stands out as a hopeful strategy to navigate these waters. This method's proven efficacy across areas like deep learning, cloud computing, data centers, and mobile SoCs underscores its potential. Yet, fully capitalizing on the advantages of hybrid parallel computing is riddled with obstacles. Challenges such as restricted adaptability, elevated development expenses, software compatibility concerns, the absence of a cohesive programming model, and reliance on specific vendors loom large. As we chart this terrain, the future of high-performance processing relies on our prowess to innovate, adapt, and fine-tune. The shift from ILP to hybrid parallel computing speaks volumes about our journey thus far, and the road ahead appears just as, if not more, exhilarating.

References

[1] Aiken, A., Banerjee, U., Kejariwal, A., Nicolau, A. (2016). Introduction. In: Instruction Level Parallelism. Springer, Boston, MA. https://doi.org/10.1007/978-1-4899-7797-7_1.

[2] Fatehi, E., & Gratz, P. (2014, August). ILP and TLP in shared memory applications: A limit study. In Proceedings of the 23rd international conference on Parallel architectures and compilation (pp. 113-126).

[3] Kiriansky, V., Xu, H., Rinard, M., & Amarasinghe, S. (2018, November). Cimple: Instruction and memory level parallelism: A dsl for uncovering ilp and mlp. In Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (pp. 1-16).

[4] Zaidi, A. M., Iordanou, K., Luján, M., & Gabrielli, G. (2021, March). Loopapalooza: Investigating Limits of Loop-Level Parallelism with a Compiler-Driven Approach. In 2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (pp. 128-138). IEEE.

[5] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., & Burger, D. (2011, June). Dark silicon and the end of multicore scaling. In Proceedings of the 38th annual international symposium on Computer architecture (pp. 365-376).

[6] Dally, W. J., Turakhia, Y., & Han, S. (2020). Domain-specific hardware accelerators. Communications of the ACM, 63(7), 48-57.
[7] Chamberlain, R. D. (2020). Architecturally truly diverse systems: A review. Future Generation Computer Systems, 110, 33-44.
[8] Hegde, V., & Usmani, S. (2016). Parallel and distributed deep learning. May, 31, 1-8.
[9] Madiajagan, M., & Raj, S. S. (2019). Parallel computing, graphics processing unit (GPU) and new hardware for deep learning in computational intelligence research. In Deep learning and parallel computing environment for bioengineering systems (pp. 1-15). Academic Press.
[10] Choi, W., Duraisamy, K., Kim, R. G., Doppa, J. R., Pande, P. P., Marculescu, R., & Marculescu, D. (2016, October). Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In Proceedings of the international conference on compilers, architectures and synthesis for embedded systems (pp. 1-10).
[11] Kim, Y. W., Choi, S. H., & Han, T. H. (2021). Rapid topology generation and core mapping of optical network-on-chip for heterogeneous computing platform. IEEE Access, 9, 110359-110370.
[12] Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., ... & Boudoukh, G. (2017, February). Can FPGAs beat GPUs in accelerating next-generation deep neural networks? . In Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays (pp. 5-14).
[13] Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., ... & Burger, D. (2014). A reconfigurable fabric for accelerating large-scale datacenter services. ACM SIGARCH Computer Architecture News, 42(3), 13-24.
[14] Damiani, A., Fiscaletti, G., Bacis, M., Brondolin, R., & Santambrogio, M. D. (2022). Blastfunction: A full-stack framework bringing fpga hardware acceleration to cloud-native applications. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 15(2), 1-27.
[15] Bobda, C., Mbongue, J. M., Chow, P., Ewais, M., Tarafdar, N., Vega, J. C., ... & Tessier, R. (2022). The future of FPGA acceleration in datacenters and the cloud. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 15(3), 1-42.
[16] Shan, Y., Lin, W., Guo, Z., & Zhang, Y. (2022, August). Towards a fully disaggregated and programmable data center. In Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems (pp. 18-28).
[17] Gouk, D., Lee, S., Kwon, M., & Jung, M. (2022). Direct access, {High-Performance} memory disaggregation with {DirectCXL}. In 2022 USENIX Annual Technical Conference (USENIX ATC 22) (pp. 287-294).
[18] Vuppalapati, M., Miron, J., Agarwal, R., Truong, D., Motivala, A., & Cruanes, T. (2020). Building an elastic query engine on disaggregated storage. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20) (pp. 449-462).
[19] Halpern, M., Zhu, Y., & Reddi, V. J. (2016, March). Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 64-76). IEEE.
[20] Zhu, Y., Mattina, M., & Whatmough, P. (2018). Mobile machine learning hardware at arm: a systems-on-chip (soc) perspective. arXiv preprint arXiv:1801.06274.
[21] Zhu, Y., Samajdar, A., Mattina, M., & Whatmough, P. (2018). Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. arXiv preprint arXiv:1803.11232.
[22] Hassan, M. (2017). Heterogeneous MPSoCs for mixed criticality systems: Challenges and opportunities. arXiv preprint arXiv:1706.07429.
[23] Majo, Z., & Gross, T. R. (2011, May). Memory system performance in a NUMA multicore multiprocessor. In Proceedings of the 4th Annual International Conference on Systems and Storage (pp. 1-10).

[24] Mittal, S. (2020). A survey of FPGA-based accelerators for convolutional neural networks. Neural computing and applications, 32(4), 1109-1139.

[25] Agrawal, R., de Castro, L., Yang, G., Juvekar, C., Yazicigil, R., Chandrakasan, A., ... & Joshi, A. (2023, February). FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption. In 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 882-895). IEEE.

[26] Haghi, A., Marco-Sola, S., Alvarez, L., Diamantopoulos, D., Hagleitner, C., & Moreto, M. (2021, August). An FPGA accelerator of the wavefront algorithm for genomics pairwise alignment. In 2021 31st International Conference on Field-Programmable Logic and Applications (FPL) (pp. 151-159). IEEE.

[27] Wang-dong, Y. A. N. G., & Hao-tian, W. A. N. G. Survey of Heterogeneous Hybrid Parallel Computing. Computer Science, ChongQing. China, 47, 5-10.

[28] Nowatzki, T., Gangadhan, V., Sankaralingam, K., & Wright, G. (2016, March). Pushing the limits of accelerator efficiency while retaining programmability. In 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 27-39). IEEE.

[29] Hill, M. D., & Reddi, V. J. (2021). Accelerator-level parallelism. Communications of the ACM, 64(12), 36-38.

[30] Fuchs, A., & Wentzlaff, D. (2019, February). The accelerator wall: Limits of chip specialization. In 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 1-14). IEEE.