Flynn's taxonomy

Flynn's taxonomy is a classification of computer architectures, proposed by Michael J. Flynn in 1966.[1] and extended in 1972.[2] The classification system has stuck, and it has been used as a tool in design of modern processors and their functionalities. Since the rise of multiprocessing central processing units (CPUs), a multiprogramming context has evolved as an extension of the classification system. Vector processing, covered by Duncan's taxonomy,[3] is missing from Flynn's work because the Cray-1 was released in 1977: Flynn's second paper was published in 1972.

Contents

Classifications

The four initial classifications defined by Flynn are based upon the number of concurrent instruction (or control) streams and data streams available in the architecture.[4] Flynn later defined three additional sub-categories of SIMD in 1972.[2]

Single instruction stream, single data stream (SISD)

A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches a single instruction stream (IS) from memory. The CU then generates appropriate control signals to direct a single processing element (PE) to operate on a single data stream (DS) i.e., one operation at a time.

Examples of SISD architectures are the traditional uniprocessor machines like older personal computers (PCs; by 2010, many PCs had multiple cores) and mainframe computers.

Single instruction stream, multiple data streams (SIMD)

A single instruction is simultaneously applied to multiple different data streams. Instructions can be executed sequentially, such as by pipelining, or in parallel by multiple functional units. Flynn's 1972 paper subdivided SIMD down into three further categories:[2]

- **Array processor** – These receive the one (same) instruction but each parallel processing unit has its own separate and distinct memory and register file.
- **Pipelined processor** – These receive the one (same) instruction but then read data from a central resource, each processes fragments of that data, then writes back the results to the same central resource. In Figure 5 of Flynn's 1977 paper that resource is main memory: for modern CPUs that resource is now more typically the register file.
- **Associative processor** – These receive the one (same) instruction but in each parallel processing unit an *independent* decision is made, based on data *local* to the unit, as to whether to perform the execution or whether to skip it. In modern terminology this is known as "predicated" (masked) SIMD.

Some modern designs (GPUs in particular) take features of more than one of these subcategories: GPUs of today are SIMT but also are Associative i.e. each processing element in the SIMT array is also predicated.

Array processor

The modern term for an array processor is "single instruction, multiple threads" (SIMT). This is a distinct classification in Flynn's 1972 taxonomy, as a subcategory of SIMD. It is identifiable by the parallel subelements having their own independent register file and memory (cache and data memory). Flynn's original papers cite two historic examples of SIMT processors: SOLOMON and ILLIAC IV.

Nvidia commonly uses the term in its marketing materials and technical documents, where it argues for the novelty of Nvidia architecture.[6] SOLOMON predates NVidia by more than 60 years.

The Aspex Microelectronics Associative String Processor (ASP)[7] categorised itself in its marketing material as "massive wide SIMD" but had *bit-level* ALUs and bit-level predication (Flynn's taxonomy: associative processing), and each of the 4096 processors had their own registers and memory (Flynn's taxonomy: array processing). The Linedancer, released in 2010, contained 4096 2-bit predicated SIMD ALUs, each with its own content-addressable memory, and was capable of 800 billion instructions per second.[8] Aspex's ASP associative array SIMT processor predates NVIDIA by 20 years.[9][10]

Pipelined processor

At the time that Flynn wrote his 1972 paper many systems were using main memory as the resource from which pipelines were reading and writing. When the resource that all "pipelines" read and write from is the register file rather than main memory, modern variants of SIMD result. Examples include Altivec, NEON, and AVX.

An alternative name for this type of register-based SIMD is "packed SIMD"[11] and another is SIMD within a register (SWAR). When predication is applied, it becomes associative processing (below)

Associative processor

The modern term for associative processor is "predicated" (or masked) SIMD. Examples include AVX-512.

Multiple instruction streams, single data stream (MISD)

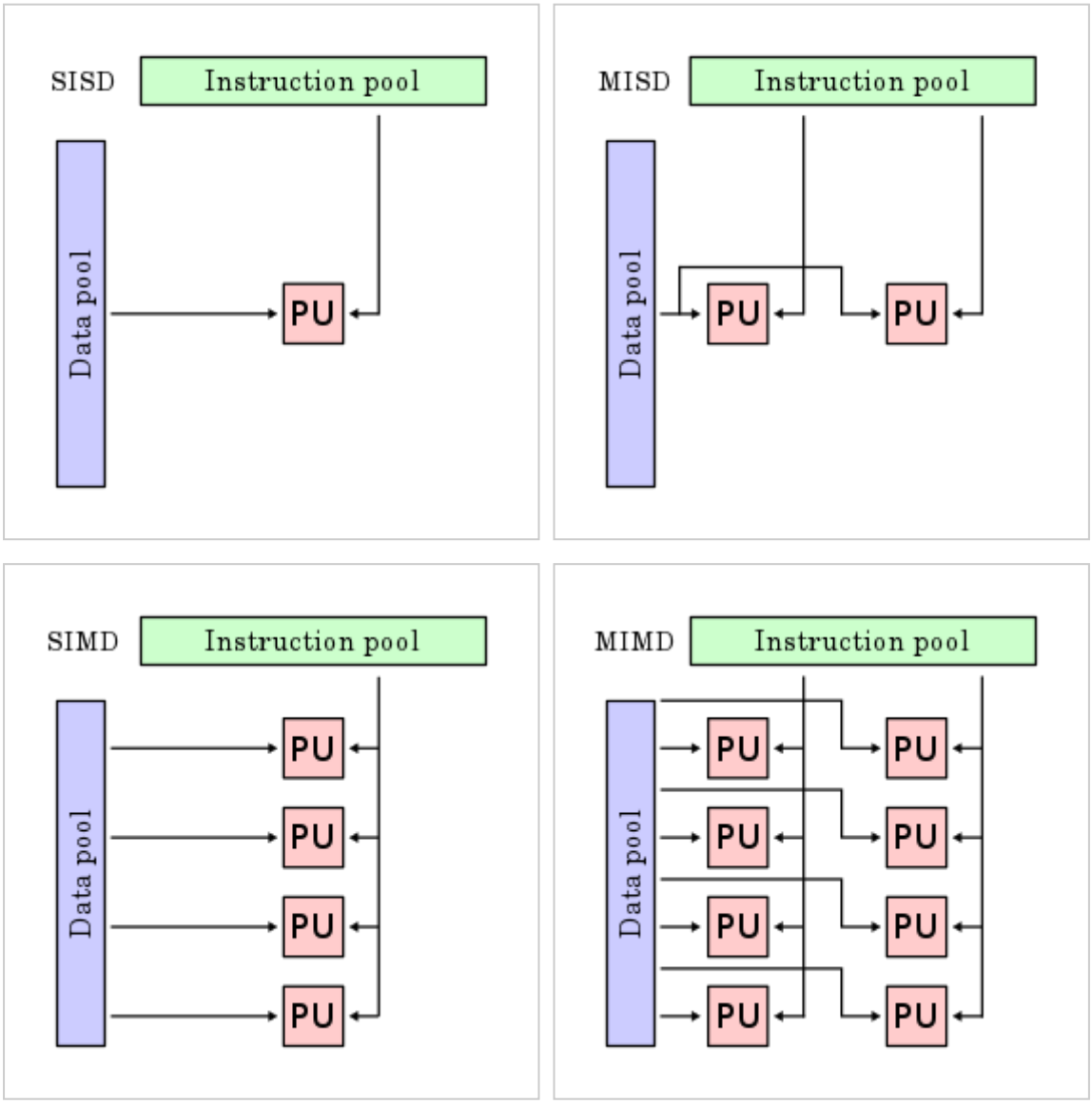
Multiple instructions operate on one data stream. This is an uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.[12]

Multiple instruction streams, multiple data streams (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. MIMD architectures include multi-core superscalar processors, and distributed systems, using either one shared memory space or a distributed memory space.

Diagram comparing classifications

These four architectures are shown below visually. Each processing unit (PU) is shown for a uni-core or multi-core computer:



Further divisions

As of 2006, all of the top 10 and most of the TOP500 supercomputers are based on a MIMD architecture.

Although these are not part of Flynn's work, some further divide the MIMD category into the two categories below,[13][14][15][16][17] and even further subdivisions are sometimes considered.[18]

Single program, multiple data streams (SPMD)

Multiple autonomous processors simultaneously executing the same program (but at independent points, rather than in the lockstep that SIMD imposes) on different data. Also termed *single process, multiple data*[17] - the use of this terminology for SPMD is technically incorrect, as SPMD is a parallel execution model and assumes multiple cooperating processors executing a program. SPMD is the most common style of parallel programming.[19] The SPMD model and the term was proposed by Frederica Darema of the RP3 team.[20]

Multiple programs, multiple data streams (MPMD)

Multiple autonomous processors simultaneously operating at least 2 independent programs. Typically such systems pick one node to be the "host" ("the explicit host/node programming model") or "manager" (the "Manager/Worker" strategy), which runs one program that farms out data to all the other nodes which all run a second program. Those other nodes then return their results directly to the manager. An example of this would be the Sony PlayStation 3 game console, with its SPU/PPU processor.

See also

- Feng's classification
- Händler's Erlangen Classification System (ECS)
- SWAR

References

1. Flynn, Michael J. (December 1966). "Very high-speed computing systems" (https://course.ece.cmu.edu/~ece447/s13/lib/exe/fetch.php?media=01447203.pdf) (PDF). *Proceedings of the IEEE*. **54** (12): 1901–1909. doi:10.1109/PROC.1966.5273 (https://doi.org/10.1109%2FPROC.1966.5273).
2. Flynn, Michael J. (September 1972). "Some Computer Organizations and Their Effectiveness" (https://www.cs.utah.edu/~hari/teaching/paralg/Flynn72.pdf) (PDF). *IEEE Transactions on Computers*. **C-21** (9): 948–960. doi:10.1109/TC.1972.5009071 (https://doi.org/10.1109%2FTC.1972.5009071). S2CID 18573685 (https://api.semanticscholar.org/CorpusID:18573685).
3. Duncan, Ralph (February 1990). "A Survey of Parallel Computer Architectures" (http://www.eng.ucy.ac.cy/theocharides/Courses/ECE656/Duncan90.pdf) (PDF). *Computer*. **23** (2): 5–16. doi:10.1109/2.44900 (https://doi.org/10.1109%2F2.44900). S2CID 15036692 (https://api.semanticscholar.org/CorpusID:15036692). Archived (https://web.archive.org/web/20180718181803/http://www.eng.ucy.ac.cy/theocharides/Courses/ECE656/Duncan90.pdf) (PDF) from the original on 2018-07-18. Retrieved 2018-07-18.
4. "Data-Level Parallelism in Vector, SIMD, and GPU Architectures" (http://www.cse.msu.edu/~cse820/lectures/CAQA5e_ch4.pdf) (PDF). 12 November 2013.

5. Flynn, Michael J. (September 1972). "Some Computer Organizations and Their Effectiveness" (https://www.cs.utah.edu/~hari/teaching/paralg/Flynn72.pdf) (PDF). *IEEE Transactions on Computers*. **C-21** (9): 948–960. doi:10.1109/TC.1972.5009071 (https://doi.org/10.1109%2FTC.1972.5009071).
6. "NVIDIA's Next Generation CUDA Compute Architecture: Fermi" (http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf) (PDF). *Nvidia*.
7. Lea, R. M. (1988). "ASP: A Cost-Effective Parallel Microcomputer". *IEEE Micro*. **8** (5): 10–29. doi:10.1109/40.87518 (https://doi.org/10.1109%2F40.87518). S2CID 25901856 (https://api.semanticscholar.org/CorpusID:25901856).
8. "Linedancer HD – Overview" (https://web.archive.org/web/20061013175702/http://www.aspex-semi.com/pages/products/products_linedancer_hd_overview.shtml). *Aspex Semiconductor*. Archived from the original (http://www.aspex-semi.com/pages/products/products_linedancer_hd_overview.shtml) on 13 October 2006.
9. Krikelis, A. (1988). *Artificial Neural Network on a Massively Parallel Associative Architecture*. International Neural Network Conference. Dordrecht: Springer. doi:10.1007/978-94-009-0643-3_39 (https://doi.org/10.1007%2F978-94-009-0643-3_39). ISBN 978-94-009-0643-3.
10. Ódor, Géza; Krikelis, Argy; Vesztergombi, György; Rohrbach, Francois. "Effective Monte Carlo simulation on System-V massively parallel associative string processing architecture" (https://core.ac.uk/download/pdf/25268094.pdf) (PDF).
11. Miyaoka, Y.; Choi, J.; Togawa, N.; Yanagisawa, M.; Ohtsuki, T. (2002). *An algorithm of hardware unit generation for processor core synthesis with packed SIMD type instructions*. Asia-Pacific Conference on Circuits and Systems. Vol. 1. p. 171–176. doi:10.1109/APCCAS.2002.1114930 (https://doi.org/10.1109%2FAPCCAS.2002.1114930). hdl:2065/10689 (https://hdl.handle.net/2065%2F10689).
12. Spector, A.; Gifford, D. (September 1984). "The space shuttle primary computer system". *Communications of the ACM*. **27** (9): 872–900. doi:10.1145/358234.358246 (https://doi.org/10.1145%2F358234.358246). S2CID 39724471 (https://api.semanticscholar.org/CorpusID:39724471).
13. "Single Program Multiple Data stream (SPMD)" (http://www.llnl.gov/casc/Overture/henshaw/documentation/App/manual/node36.html). Llnl.gov. Retrieved 2013-12-09.
14. "Programming requirements for compiling, building, and running jobs" (https://web.archive.org/web/20060901114042/http://www.cisl.ucar.edu/docs/lightning/program.jsp). *Lightning User Guide*. Archived from the original (http://www.cisl.ucar.edu/docs/lightning/program.jsp) on September 1, 2006.
15. "CTC Virtual Workshop" (http://web0.tc.cornell.edu/Services/Education/Topics/Parallel/Design/SPMD.aspx). Web0.tc.cornell.edu. Retrieved 2013-12-09.
16. "NIST SP2 Primer: Distributed-memory programming" (https://web.archive.org/web/20131213105449/http://math.nist.gov/~KRemington/Primer/distrib.html). Math.nist.gov. Archived from the original (http://math.nist.gov/~KRemington/Primer/distrib.html) on 2013-12-13. Retrieved 2013-12-09.
17. "Understanding parallel job management and message passing on IBM SP systems" (https://web.archive.org/web/20070203153908/http://www.cisl.ucar.edu/docs/ibm/ref/parallel.html). Archived from the original (http://www.cisl.ucar.edu/docs/ibm/ref/parallel.html) on February 3, 2007.
18. "9.2 Strategies" (https://web.archive.org/web/20060910222800/http://www.tc.cornell.edu/Services/Education/Topics/Parallel/Distributed/+9.2+Strategies.htm). *Distributed Memory Programming*. Archived from the original (http://www.tc.cornell.edu/Services/Education/Topics/Parallel/Distributed/+9.2+Strategies.htm) on September 10, 2006.

19. "Single program multiple data" (https://xlinux.nist.gov/dads/HTML/singleprogrm.html). Nist.gov. 2004-12-17. Retrieved 2013-12-09.
20. Darema, Frederica; George, David A.; Norton, V. Alan; Pfister, Gregory F. (1988). "A single-program-multiple-data computational model for EPEX/FORTRAN". *Parallel Computing*. **7** (1): 11–24. doi:10.1016/0167-8191(88)90094-4 (https://doi.org/10.1016%2F0167-8191%2888%2990094-4).
