A Scalable, Multi-Thread, Multi-Issue Array Processor Architecture for DSP Applications Based on Extended Tomasulo Scheme

Mladen Bereković[1] and Tim Niggemeier[2]

[1] IMEC, Belgium, and TU Delft, Netherlands,
mladen.berekovic@imec.be,
[2] IBM Deutschland Entwicklung GmbH, Germany
niggemei@de.ibm.com

Abstract. A scalable, distributed micro-architecture is presented that emphasizes on high performance computing for digital signal processing applications by combining high frequency design techniques with a very high degree of parallel processing on a chip. The architecture is based on a superscalar processor model with out-of-order execution, that supports specialized, complex DSP function units, and simultaneous instruction issue from multiple independent threads (SMT). Consequent application of fine clustering reduces the cycle-time for wire-sensitive building blocks of the processor like the register file and leads to a distributed architecture model, where independent thread processing units, ALUs, registers files and memories are distributed across the chip and communicate with each other by special networks, forming a "network-on-a-chip" (NOC) [1]. The communication protocol is a modified version of Tomasulo's scheme [2], that was extended to eliminate all central control structures for the data flow and to support multithreading. The performance of the architecture is scalable with both the number of function units and the number of thread units without having any impact on the processors cycle-time.

1 Intoduction

Today's typical embedded DSP systems are built around system-on-chip architectures [3], [4], consisting of one or more DSP and RISC cores, that are interconnected by a standardized on-chip peripheral bus like the ARM-AMBA [5], Multi-AMBA, or a similar proprietary busses.. These system architectures favor a model of lower-performance, lower-power, embedded processor cores that are assisted by one or more hardwired accelerators for specialized tasks like filter or bitstream processing. Making the hardwired coprocessors more programmable leads to a "softening of hardware" [6], with many small, configurable processors and DSPs on a single-chip that share several custom function specific coprocessors or even reprogrammable and reconfigurable hardware blocks [7]. An on-chip communication network is needed to keep as many of these cores busy as possible, leading to networks-on-chip architectures [1].

We propose a different architecture framework for network-on-chip architectures that offers a very high parallelization potential for DSP applications. It is based on a simultaneously multithreaded processor model where multiple independent thread units

simultaneously issue multiple instructions to a large array of function units, where they are processed in a dataflow-manner.

Although this work fits nicely into Meerbergen's definition of an NOC, it differs in three terms. First, it is based on a different processor model, a superscalar, out-of-order processor with SMT extensions instead of a static VLIW architecture. Second, the dynamically scheduled processors use switched router networks for both resource sharing and explicit communication between the processors or thread units. Third, function specific accelerator units that are embedded as application specific instructions are shared between all threads, leading to a higher utilization of these modules. The following section discusses the details of the overall microarchitecture and explains the developed modifications to the scheduling mechanism in contrast to the classical Tomasulo scheme. The third section discusses related work. Finally, Section 4 summarizes the main conclusions of this work.

2 Extending Tomasulo's scheme for Clustered Architectures with explicit operand transport

The proposed architecture takes on Tomasulo's original design and extends it with superscalar and SMT features. Fig. 1 shows the resulting architecture that is distributed along the FUs that serve as basic building blocks. The heterogeneous array of small processing units is connected by a network for operand and instruction communication. In contrast to classic centralized schedulers, forwarding of result tags is replaced with dedicated operand transport instructions. Each FU forms a separate cluster with a private register file. The register files of the thread units are distributed to the local register files of the FU-clusters. This offers two advantages: First, every local register file needs only 2 write and one read port, independent of the number of threads and issue slots. This makes the architecture more scalable in terms of issues per cycle and cycle time. The second is that the total number of registers can be significantly increased, paving the way for large instruction windows that typically support a large number of "in-flight" instructions, without having effecting the cycle time.

This work makes some essential modifications to Tomasulo's original scheme to eliminate the costly tag matching process: After the register renaming table is read during instruction decode stage, all dependence information is available to build the dynamic signal-flow graph. In analogy to the "Transport Triggered Architectures" [28], or TTAs, the instruction is then split into three subinstructions: a (mathematical) function and two helper instructions that move the two source operands from the place of their generation, i.e. the physical reservation station (=register, cluster) read from the renaming table, to the place where the operation is performed, i.e. the destination reservation station (register/cluster) in the function unit to which the instruction has been dynamically assigned. These helper transport instructions inherently have knowledge of the physical location of both, their source and their destination operands, since these are directly reflected by their reservation station numbers and function units. The helper transport instructions, consisting of a simple pair of reservation station or rename register numbers (src, dest), are sent to the physical address of the source. As soon as the source data gets available the helper transfer instruction is executed, i.e. the data is send

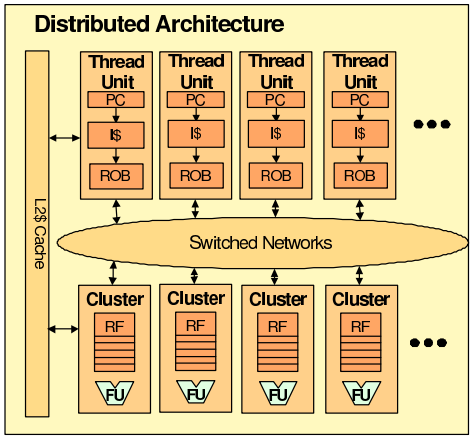

Fig. 1. Distributed SMT processor overview. RF= register file, ROB= reorder buffer, FU= function unit.

to the destination register via the switched operand network. This way, a precise point-to-point communication link is established between producer instruction and producing function unit on one side and consumer instruction and consuming function unit on the other side. The dependence graph is early resolved into a signal-flow graph during the rename stage. This reduces the complexity of the transform process significantly when compared with a central instruction scheduler implementation. The central instruction scheduler checks *all completing* versus *all waiting* instructions for *all threads* using a content addressable memory (CAM), whereas the renaming logic only needs to perform a (renaming) table access for all decoded instructions *within a single thread*.

To explain the mechanism of the helper copy instructions, consider the following example sequence of 4 instructions:

$$a : R4 \;<-\; R0 + R8; \qquad b : R2 \;<-\; R0 * R4$$

The physical registers are addressed with two index sets: one for the FU and one for the register number within the FU. So (5,10) denotes the 10th register in FU (cluster) 5. Suppose that FU1 is an adder and FU2 is a multiplier unit.

After the first clock cycle, logical register R4 is assigned to FU1, register 1 (=R(1,1)), and instruction a is moved to FU1 (adder). A set of copy instructions for the two source operands is generated. After the second cycle, instruction b is moved to FU4 (multiplier) and R2 is assigned to FU4, R1 (=R(4,1). A second set of copy instructions are generated. So the first two instructions become after renaming:

$$\begin{array}{ll} a : R(1,1) \;<-\; R0 + R8; & b : R(2,1) \;<-\; R0 * R(1,1); \\ a1 : cp\ \ R0 \;->\; R(1,1); & b1 : cp\ \ R0 \;->\; R(4,1); \\ a2 : cp\ \ R8 \;->\; R(1,1); & b2 : cp\ \ R(1,1) \;->\; R(4,1); \end{array}$$

The cp instructions are sent to their appropriate source operand destinations, e.g. b2 is sent to FU1, register 1 and stored in the appropriate field of the cp operand buffer for R(1,1). Once the data is available, the cp instruction is executed and the data send to R(4,1) and stored in its input operand buffer.

Fig. 2 shows the simulated timing estimates for different configurations for the fully associative (CAM-based) instruction-scheduler (window) obtained from CACTI 3. What can be seen is the superlinear increase in delay with the number of ports. Also, large register file sizes lead to significantly higher delay. From these numbers it can be

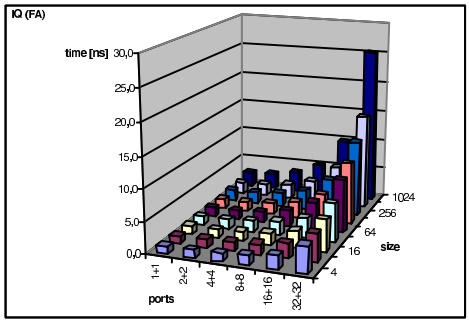

Fig. 2. Timing Estimates for different scheduler configurations using CACTI 3.0.

concluded that an optimized clustered architecture is built using memory blocks with only few ports and issue slots. Fig. 3 shows the resulting cluster data path architecture. The local register file stores the function unit's result data that is addressed by the destination register. Each register file entry has two associated entries in the operand buffer memories, that replace the original reservation stations. A third memory, the cp instruction buffer, contains entries to store a cp instruction for the register's result. This way, a precise point-to-point communication link is established between producer instruction and producing function unit on one side and consumer instruction and consuming function unit on the other side. The dependence graph is early resolved into a signal-flow graph during the rename stage. This reduces the complexity of the transform process significantly when compared with a central instruction scheduler implementation. The instruction scheduler checks *all completing* versus *all waiting* instructions for *all threads* in a content addressable memory (CAM), whereas the renaming logic only needs to perform a (renaming) table access for all decoded instructions *within a single thread*.

The cluster memories are of small size and have a limited port number, e.g. 2 ports for the reservation stations that serve as input buffers. An architecture with 16 function units (=clusters) with a moderate 32 registers each would already have 512 physical registers, which is more than any available processor offers.

The encapsulation of the whole processing logic in small clusters with local interconnects and fully pipelined in- and outbound communication makes very high frequency realisations possible. A recent implementation of a comparable execution core

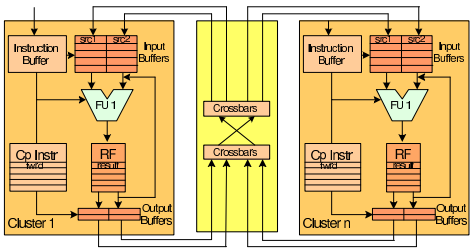

Fig. 3. Modified Tomasulo architecture with distributed registerfile and instruction scheduling.

cluster (2x 32bit ALU, 32-entry x 32-bit register file, 8 entry x 2 scheduler loop) on a 130nm, six-metal, Dual VT CMOS technology consumed 2.3 mm2 and yielded 5 GHz [29].

Fig. 4 shows the basic steps pipelining scheme, although not in full cycle-true details, the delays between the operand transfers are left out for example. The basic pipeline steps are:

1. The Function Units post their free resources via a global resource broadcasting network (FU2Rename network) to the Thread Units (TUs). The TUs pick up these resources autonomously according to their individual needs. A free resource consists of a package containing
   - a free (destination) register
   - An associated entry in the two operand buffers (typically the same register address)
   - An entry in the local instruction window
2. The instruction renaming stage consumes the free resources and the corresponding sub-instructions are created on-the-fly in the decode Issue and Plan stages, where tzey are also sent to the FUs via the Rename2FU Network
3. In the FUs the instructions are stored in the local instruction windows and the necessary data transfer operations are performed (on availability of their source operands) and the data is axchanged directly between the FUs on the FU2FU data network.
4. The FUs notify back the TUs via the FU2Retire network that the instructions have executed and are ready for retirement.
5. The Retire stage in the Tus retires the instructions and notifies back the FUs about the release of the _overwritten_ destination register (from prvious valus, that has to be stored in the rename tables as well).

3 Simulation Results with MPEG-4 Kernels

Cycle-true simulations of two typical and computation-intensive DSP kernels from MPEG-4 [3] were performed on an RT-level simulator of the architecture: global motion compensation (GMC), which is equivalent to an affine warping of a 16x16 pixel

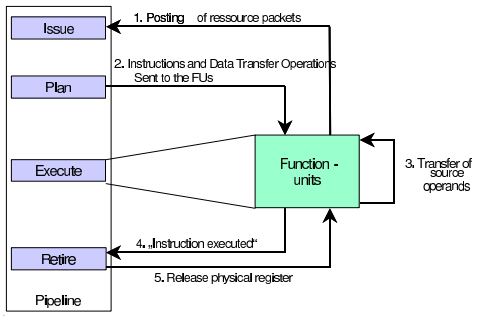

Fig. 4. Pipelining scheme.

block and deblocking. The benchmarks were hand-optimized in Assembler for the architecture.

Fig. 5 and Fig. 6 show the results for the MPEG-4 GMC subtask, which is equivalent to an affine warping of a 16x16 pixel block. The GMC is one of the most performance critical kernel loops of the MPEG-4 standard. The benchmarks were hand-optimized in Assembler for the architecture without compiler-support and exploit multi-threading. This approach is different from General-Purpose processors, but is typically employed on DSPs.

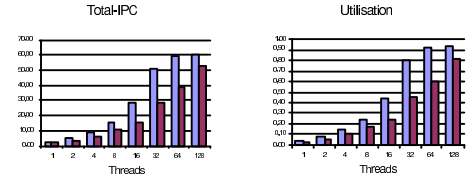

Fig. 5. GMC simulation results for 1 to 128 threads. The left bar shows the Issue-IPC, the right bar retire-IPC.

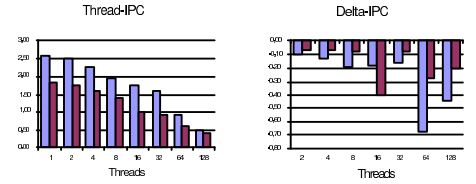

Fig. 6. Simulated Thread IPC results for 1..128 threads.

For all simulation runs the same configuration with 64 FUs was used except that the number of thread units was increased to the number of parallel running threads. The register-files size is 256 registers each, as is the reorder buffer size (and hence the instruction window). Each thread has a local L1 I-cache of 32kB, and a maximal fetch and issue bandwidth of 16 instructions. There is a shared L1-D-cache, that is connected to the Load/Store units. All L1-caches are backed up by a larger 4MB L2-cache. However, for the current simulations, the L1-D cache hit-rate was set to 100% to explore the full performance potential. The branch prediction uses a simple shared array of 2-bit counters.

The simulation results show that if enough data bandwidth is available, the performance scales with the number of threads. The lower single-thread performance is caused by inter-cluster latencies and branch miss-predictions. This benchmark had a 40% miss-prediction rate, which results in significant IPC losses (30%) between issue and retire stages (see the two difference between the red and the blue rows), even in a multi-threaded environment. It is interesting to note that this loss gets smaller as the number of threads increases, which is caused by the fact that resource contention between the thread units leads also to lower issue rates per thread and therefore to fewer unsued or wasted execution cycles. The high utilization rates of the FUs in the above simulations show however that multi-threading effectively compensates the high latencies for data-forwarding between dependent instructions. Another conclusion is that, in the presence of multiple available threads to run, speculation does not yield better results.

The simulations results shown in Fig. 7 were obtained from simulations with different bandwidth configurations of the internal networks. With 1 denoting a single word bandwidth and 16 denoting 16 word bandwidth, for example for the data communication network.

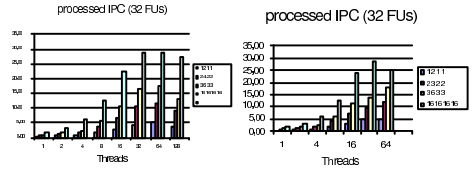

Fig. 7. Simulation results: 32 FUs, 1 to 128 threads. Left: GMC. Right: MPEG-4 Deblocking filter. The different bars represent different network bandwidth.

The simulation results show that if enough network bandwidth is available, the performance scales with the number of threads. The lower single-thread performance is caused by inter-cluster latencies and branch miss-predictions. The high utilization rate of the FUs for higher thread numbers (IPC$>$15) shows however, that multi-threading effectively compensates the high latencies for data-forwarding between dependent instructions.

4 Related Work

Previous work on DSP processor architectures has mainly been focused instruction-level parallelism, particularly on statically scheduled, VLIW processing. These architectures offer good compiler support for a moderate parallelism of 2-8 (peak) instructions per cycle, but run out of steam for larger parallelisms. Furthermore, their main advantage, the simpler hardware compared to dynamically scheduled ooo-execution models diminishes with the huge demand for registers and for more register file ports. A main disadvantage of VLIW architectures is their inability to dynamically react to cache misses and resource contention leading to frequent performance-limiting stalls [30], and making it inadequate for simultaneous multithreading.

Only few studies have been performed so far that investigate multithreading, out-of-order execution, or SMT for digital signal processing applications [31]. With the rapid increase of the transistor budget that is available on chips, DSP chip multi-processors are beginning to surface and it is to be expected that this trend is going to continue.

Previous studies [32] have identified the instruction window design to be a major limiter for more (future) processor speed. Several studies have advocated improvements for larger window sizes and higher frequency designs [33], while other designs eliminate associative compares altogether from the instruction scheduling [34].

Clustering has been proposed by many researches to address the complexity problems of large instruction windows and register files and it is employed in several recent processor designs. Studies of clustered architectures include the Multi-Scalar project, Trace Processors, EDF [35], Hierarchical Scheduling Windows, and Transport Triggered Architectures (TTAs) [36]. In contrast to this work TTAs are based on static VLIW techniques and compiler-generated scheduling. Other approaches include array processors like the UT/Austin Grid architecture [37], MIT's RAW project [38], and Stanford's IMAGINE stream processor project.

5 Conclusion

We propose a distributed SMT processor architecture as a scalable network-on-chip platform for dynamically reconfigurable digital signal processing. Although well suited for multiprocessing, it combines all characteristics needed by DSP applications in a single processor framework. Like the very popular VLIW processors it supports multiple instruction issue and offers a very large number of registers, but it avoids the performance degrading stalls in instruction issue due to cache misses. Just as VLIW processors it supports the embedding of special function units that can perform parallel vector instructions, or complex functions like filtering. Also, fine-clustering offers a path to high frequency implementations, but unlike VLIW processors, multithreading effectively compensates the related IPC losses. The architecture shares the system-on-chip processing model with multiple heterogeneous ASIP cores on a chip, each specialised for a specific task, communicating with each other over specialised on-chip networks. However, it offers more flexibility for resource sharing especially for the costly, high-performance, specialised function units. Furthermore, it offers fast context switches and a fine-grained and fast communication scheme between threads that is based on

hardware CSPs channels. Large scale clustered SMT processors with many shared specialised or configurable function units come very close to the ideal of a softening of hardware and offer an attractive alternative to both, pure FPGA implementations and to costly ASIC designs.

References

1. L. Benini, G. de Micheli, "Networks on chip: A New SOC Paradigm," IEEE Computer, Vol. 35, no. 1, Jan. 2002, pp. 70-78.
2. R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units", IBM Journal on Research and Development, Vol.11, no.1, January 1967, pp. 25-33.
3. M. Berekovic, H.-J. Stolberg, P. Pirsch, "Multi-Core System-On-Chip Architecture for MPEG-4 Streaming Video," Transactions on Circuits and Systems for Video Technology (CSVT), Vol. 12, No. 8, August 2002, pp. 688-699.
4. P. Pirsch, M. Berekovic, H.-J. Stolberg, J. Jachalsky, "VLSI Architectures for MPEG-4 Video," VLSI Conference, Taipei, April 2003.
5. ARM AMBA Specification, www.ARM.com.
6. F. Vahid, "The Softening of Hardware," IEEE Computer, Vol. 36, no. 4, April 2003, pp. 27-34.
7. H. Zhang, J.M. Rabaey et al., " A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications," Proc. Int'l. Solid-State Circuits Conference (ISSCC), San Francisco, February 2000.
8. J. L. van Meerbergen, "Lecture slides: Complex Multiprocessor architectures," www.ics.ele.tue.nl/ jef/education/5p520/index.html.
9. ISO/IEC JTC/SC29/WG11 N4668, "Overview of the MPEG-4 standard," Jeju, March 2002.
10. M. Berekovic, P. Pirsch, J. Kneip, "An Algorithm-Hardware-System Approach to VLIW Multimedia Processors," Journal of VLSI Signal Processing Systems, Vol. 20, No. 1-2, October 1998, pp. 163-180.
11. A. Allan, D. Edenfeld, W. H. Joyner, A. B. Kahng, M. Rodgers, and Y. Zorian, "2001 Technology Roadmap for Semiconductors," IEEE Computer, Vol. 35, no. 1, January 2002, pp. 42-53.
12. M. H. Lipasti and J. P. Shen "Modern Processor Design", McGrawHill, 2002.
13. M. Berekovic, H. J. Stolberg, M. B. Kulaczewski, P. Pirsch, H. Moeller, H. Runge, J. Kneip, B. Stabernack, "Instruction Set Extensions for MPEG-4 Video," Journal of VLSI Signal Processing Systems, Vol. 23, No. 1, October 1999, pp. 7-50.
14. J. P. Wittenburg, W. Hinrichs, J. Kneip, M. Ohmacht, M. Berekovic, H. Lieske, H. Kloos, P. Pirsch, "Realization of a Programmable Parallel DSP for High Performance Image Processing Applications," Design Automation Conference (DAC) 1998, June 1998, pp. 56-61.
15. R. Lee, "Accelerating Multimedia with Enhanced Microprocessors," IEEE Micro, Vol.15, no. 2, March/April 1995, pp. 22-32.
16. N. Slingerland, and A. J. Smith, "Measuring the Performance of Multimedia Instruction Sets," IEEE Transactions on Computers, Vol. 51, no. 11, November 2002, pp. 1317-1332.
17. Texas Instruments, "TMS320DM642 Technical Overview," Application Report SPRU615, Sep. 2002.
18. M. S. Lam, and R. P. Wilson, "Limits of Control Flow on Parallelism", Proc. 19th Ann. Int'l Symp. on Computer Architecture, June 1992, pp. 46-57.
19. D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", Proc. 22th Ann. Int'l Symp. on Computer Architecture, June 1995, pp. 392-403.

20. R. P. Preston, et.al "Design of an 8-wide Superscalar RISC with Simultaneous Multithreading", Solid-State Circuits Conference (ISSCC2002), San-Francisco, Ca, Febr. 2002, pp.469-471.
21. S. Palacharla, N.P. Jouppi, J. Smith, "Complexity Effective Superscalar Processors", Proc. 24th. Int'l. Symp. on Computer Architecture, June 1997, pp. 206-218.
22. B. Ackland et al., "A Single Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP," IEEE J. Solid-State Circuits, Mar. 2000, pp. 412-424.
23. H.-J. Stolberg, M. Berekovic, L. Friebe, S. Moch, S. Fluegel, X. Mao, M. B. Kulaczewski, H. Klussmann, P. Pirsch, "HiBRID-SoC: A Multi-Core System-on-Chip Architecture for Multimedia Signal Processing Applications," Proceedings Design, Automation and Test in Europe (DATE2003) - Designer's Forum, March 2003, pp. 8-13.
24. K.I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic, "The Multicluster Architecture: Reducing Cycle Time through Partitioning", Proc. 30th. Int'l. Symp. On Microarchitecure, Dec. 1997, pp.149-159.
25. R. E. Kessler, "The Alpha 21264 Microprocessor", IEEE Micro 19(2), March 1999, pp. 24-36.
26. R. Ho, K. W. Mai, M. A. Horowitz, "The Future of wires", Proceedings of the IEEE, 89(4): 490-504, Apr. 2001.
27. V. Agarwal, M.S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock Rate versus IPC: The End of the Road for conventional Microarchitectures", Proc. 27th Ann. Int'l. Symp on Computer architecture, June 2000, pp. 248-259.
28. H. Corporaal, "Microprocessor Architectures from VLIW to TTA," John Wiley & Sons, 1998.
29. S. Vangal et. al., "5-Ghz 32-bit Integer Execution Core in 130-nm Dual-VT CMOS," IEEE Journal of Solid-State Circuits, vol. 37, no. 11, November 2002.
30. M. Berekovic, P. Pirsch, J. Kneip, "An Algorithm-Hardware-System Approach to VLIW Multimedia Processors," Journal of VLSI Signal Processing Systems, Vol. 20, No. 1-2, October 1998, pp. 163-180.
31. Y.-K. Chen, R. Lienhart, E. Debes, M. Holliman, and M. Yeung, "The impact of SMT/SMP Designs on Multimedia Software Engineering: A Workload Analysis Study," Fourth International Symposium on Multimedia Software Engineering, December 2002.
32. David W. Wall, "Limits of Instruction-Level Parallelism", Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April 1991, pp. 176-188.
33. M. Brown, J. Stark, and Y. Patt, "Select-Free Instruction Scheduling Logic, " Micro-34, 2001, pp. 204-213.
34. S. Weiss, and J.E: Smith, "Instruction Issue Logic in Pipelined Supercomputers," IEEE Trans. on Comp., vol. C 33, No. 11, Nov. 1984, pp. 1013-1022
35. T. Sato, Y. Nakamura, and I. Arita, "Revisiting Direct Tag Search Algorithm on Superscalar Processors," in Workshop on Complexity-Effective Design, June 2001
36. H. Corporaal, "Microprocessor Architectures from VLIW to TTA," John Wiley & Sons, 1998.
37. R. Nagarajan, K. Sankaralingam, D. Burger, and S. Keckler, "Design Space Evaluation of Grid Processor Architectures, " Micro-34, 2001, pp.40-53.
38. M. B. Taylor et al., "The RAW Microprocessor: A Computational Fabric For Software Circuits and General-Purpose Programs," IEEE Micro, Vol. 22, no. 2, March/April 2002, pp. 25-35.