

Design Principles for a Virtual Multiprocessor

Philip Machanick
School of ITEE
University of Queensland
St Lucia, Qld 4068
Australia
philip.machanick@gmail.com

ABSTRACT

The case for chip multiprocessor (CMP) or multicore designs is strong, and increasingly accepted as evidenced by the growing number of commercial multicore designs. However, there is also some evidence that the quest for instruction-level parallelism, like the Monty Python parrot, is not dead but resting. The cases for CMP and ILP are complementary. A multitasking or multithreaded workload will do better on a CMP design; a floating-point application without many decision points will do better on a machine with ILP as its main parallelism. This paper explores a model for achieving both in the same design, by reconfiguring functional units on the fly. The result is a *virtual multiprocessor* (or *vMP*) which at the software level looks like either a uniprocessor with n clusters of functional units, or an n -core CMP, depending on how the data path is configured. As compared with other proposals, the vMP design aims to be as simple as possible, to maximize the probability of being able to use the alternative modes, while minimizing the cost versus a non-reconfigurable design.

Categories and Subject Descriptors

C.1.2 [Processor Architectures]: Multiple Data Stream Architectures—*hybrid architectures, design principles*

General Terms

chip multiprocessor, instruction-level parallelism

1. INTRODUCTION

The quest for instruction-level parallelism (ILP) is on the decline, as increasing design efforts are focused on multicore designs, also called chip multiprocessors (CMP) [19, 9, 17, 6, 15]. The argument for CMP is that replication of relatively small design units, achieving the same theoretical peak throughput as an aggressive ILP design, has significant design and implementation advantages. These advantages include simpler design, simpler design debugging and shorter critical paths (and hence greater ease of scaling up the clock

speed). The downside of the CMP approach is that it is not helpful in speeding up single-threaded applications. The argument from the CMP camp is that multithreading compilers are feasible [19] – but this does not help with legacy code, and recompiling with significantly different semantics implies another round of testing. Further, the range of applications which does significantly better with aggressive ILP is not large enough to justify mass-market designs, as evidenced by the switch by Intel from aggressive pipelines to more than simpler pipelines in multicore designs.

Some of the later attempts at achieving high ILP were *clustered* designs, in which the functional units were divided into clusters. Each instruction was steered to a particular cluster. If the steering policy was accurate, unrelated instructions would go to different clusters, and the register operands they needed would be in the right place. A significant fraction of work on clustering was on minimizing the mismatches between the location of a register value and an instruction which needed it [26]. Another significant area was load balance between clusters [7].

Clustered designs were promoted as an alternative to monolithic designs because they had a simpler design for each cluster, which could be treated as a design element on its own, resulting in shorter critical paths (and hence greater ease of scaling up the clock speed) [4]. Of course the simpler design components should also simplify debugging.

Have we heard all that somewhere before?

Figure 1 illustrates how similar the CMP and clustered approaches can be.

The major differences as illustrated are that the clustered architecture has a global front end which steers instructions from a single first-level cache (L1) to clusters, and a global bypass network for moving register contents across clusters, as opposed to the local L1 and bypass networks of the CMP.

These differences are comparatively minor, which raises the question: can we combine the two designs?

If we could, the end result would be a system which could run programs which do well with an aggressive ILP design as well as multitasking or multithreaded workloads which did not do well with aggressive ILP.

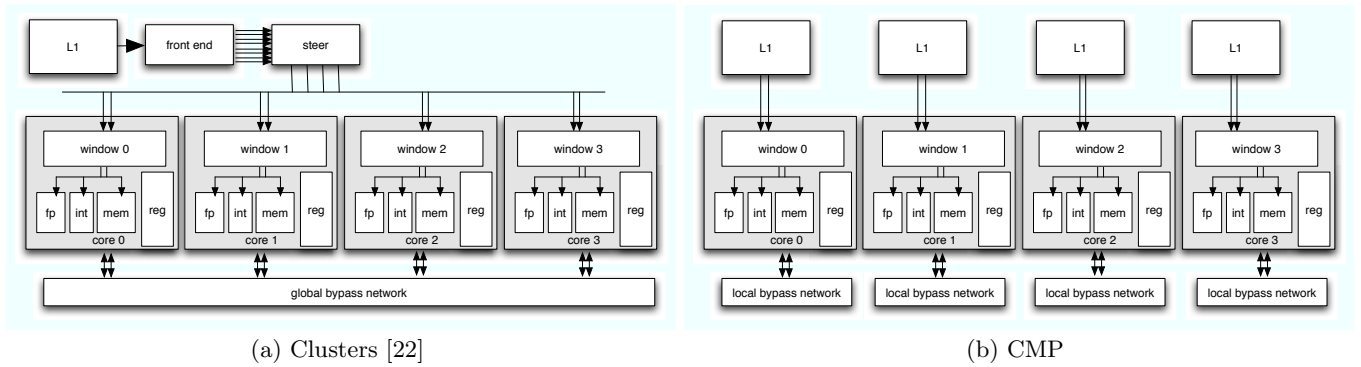


Figure 1: Clustered architecture *vs.* chip multiprocessor (CMP): there are more similarities than differences.

How could this be done?

One approach would be to design a package which could be reconfigured by a small set of changes to data and control paths, so it could run in either uniprocessor clustered mode, or CMP mode. As far as the operating system was concerned, it would have $n+1$ processors, but, at any given time, only either n simple processors or 1 complex processor was awake, and the others asleep. In the cause of minimizing complexity, the same instruction set would run in all modes. A benefit of that approach is that choosing modes becomes a performance optimization – if the wrong mode is chosen, the worst consequence is a loss of performance, rather than inability to run a given workload.

Assuming the hardware side could be implemented, the software side would be a matter of adjustments to the operating system’s ability to handle a dynamic variation in the mix of processors, including scheduling floating point-intensive programs (or any identified as ILP-centric) on the cluster, and other processes or threads in CMP mode.

The overall effect is described as a *virtual multiprocessor* or *vMP*, since, to the software, there appears to be more processors than are implemented in the hardware.

The remainder of this paper is structured as follows. Section 2 briefly surveys previous work on clustered microarchitectures, chip multiprocessors and relevant reconfigurable technologies, to provide some background. Section 3 examines how the two ideas could be combined, followed by Section 4, which assesses the problems to be solved to make the idea practical. In conclusion, the paper wraps up with a summary of findings and some options for moving ahead.

2. RELATED WORK

The obvious related work is chip multiprocessor and clustered designs. However, there are also some reconfigurable designs which have some relationship to the proposed architecture. In general, reconfigurable designs assume a more general ability to change the architecture, as in FPGA-based designs. However, reconfigurable designs with some fixed logic are closer to the general idea proposed here.

The remainder of this section reviews cluster designs, fol-

lowed by CMP work and finally related reconfigurable designs and technologies.

2.1 Clusters

All right then, if it’s resting I’ll wake it up. (*shouts into cage*)
– Monty Python Dead Parrot Sketch

Is ILP dead?

That’s a useful question to ask before delving into details of clusters (by which a particular microarchitecture organization should be understood, not the unfortunately similar term *server cluster*).

While there is no doubt that the momentum in ILP has dropped, there are still those who argue that the gains from ILP have not been fully explored and that studies to identify parallelism have been too limited [22].

Further, the argument that there are some workloads that perform better on ILP-oriented machines has not been refuted. The original CMP paper measured some examples as losing significant performance (30%) on a CMP design over the superscalar competitor they modeled [19].

For these reasons, it could be argued that the abandonment of the cluster idea – one of the more promising approaches to ILP – is premature.

As previously noted, the cluster idea was introduced as an alternative to monolithic designs.

There are several variations on proposed clusters designs and the problems they have attempted to solve. For example, a clustered design can reduce problems of clock skew across complex logic by partitioning the logic, as well providing a fine-grained model for reducing energy use [18]. Variations include a partitioned [26] versus monolithic cache architecture [4, 18].

There has been considerable work on VLIW designs, including various code scheduling schemes [20, 27] and further wrinkles on cache organization [5]. VLIW work is not specifically relevant here, and is not considered in detail.

Clustered designs have been used in real machines, including the Alpha 21264 [14].

2.2 Chip Multiprocessors

Chip multiprocessors were originally proposed as an alternative to aggressive superscalar designs [19], and developed as parts for a scalable supercomputer [9]. In this guise, they necessarily had to perform well on workloads for which aggressive superscalar designs were developed – large-scale floating-point computations.

More recently, CMPs (or multicore designs) have moved into the mainstream, with designs from Intel [6] and AMD [2] illustrating that consumer multitasking and multithreaded workloads are a reasonable target for this kind of design. While IBM has done multicore versions of the Power and PowerPC architectures starting from the POWER4 [24, 23, 10], they have failed to follow the recipe of less aggressive pipelines, and have not been able to compete with Intel on combined low power and high performance, losing Apple as a customer.

Some have advocated heterogeneous designs for multicore parts, with specialized processors for specific functionality [12]. An example of a processor in this style is the Cell [13], a design by IBM, Sony and Toshiba, with a single PowerPC core and 8 vector units. Each vector unit includes a fast local memory, the contents of which has to be explicitly managed under program control.

On the whole, issues which have resulted in success or failure of multiprocessor designs in the past are unlikely to be significantly different with multicore designs. The only practical difference is that the interprocessor interconnect is easier to scale with CPU speed, as it's part of the same logic fabric as the processor. Highly asymmetric or heterogeneous designs have tended to fail in the past on the difficulty of programming them.

Any variation on a simple symmetric multiprocessor (SMP) approach therefore has to have a plausible and demonstrably viable programming model to be credible. The tardiness of the launch of Sony's PlayStation 3 (first the hardware, then game titles), based on the Cell, is an indication of the difficulties in implementation of such designs.

2.3 Reconfigurable Designs

While a general completely reconfigurable part like an FPGA has little relationship to the vMP idea, some reconfigurable designs include hardwired logic. Designs with limited reconfiguration are closest to the general idea to vMP.

One example is DynaCORE, a dynamically reconfigurable combination of coprocessors for network processors [1]. DynaCORE contains several hardwired or FPGA implementations of algorithms which can apply in different combinations for different network protocols. A selection of these *hardware assists* can be combined in different ways through a *reconfiguration manager*, which sets up a *dispatcher* to distribute incoming data appropriately. DynaCORE is similar to vMP in that hardware resources can be reused in different ways by changing the datapath dynamically. However, it differs in that DynaCORE is oriented towards implementation on an

FPGA, i.e., aims to maximize flexibility at a cost to potential peak throughput. vMP has a much more limited model of reconfiguration.

The COBRA cryptographic engine uses the opposite strategy: the interconnect is fixed, but the processing elements can be varied [3].

The Amalgam architecture is a clustered architecture in which half the functional units form part of a conventional processor, and the other half are reconfigurable logic [16]. This approach differs from vMP in that vMP provides a way of switching between two alternative uses of conventional CPU logic, whereas Amalgam provides a mechanism for mixing conventional instructions with custom logic.

One of the more general approaches to the problem is a design with a combination of fixed and reconfigurable functional units, with logic to stall instructions for which the required functional unit is not available [25]. Unlike vMP, this approach aims to provide for varying the functionality of some of the functional units.

A promising new approach to reconfigurable technology is being developed by ChaoLogix: logic functions of a given gate can be rapidly reprogrammed by changing an input voltage [8]. A small amount of this technology to cover the parts of a vMP which need to be reconfigured to change modes would potentially avoid extra latency in choosing between logic paths for every instruction.

The closest approach to vMP is *core fusion* [11]. In the core fusion design, multiple cores can be dynamically reconfigured to vary from a multicore design with two instruction-wide pipelines to a single aggressive pipeline capable of issuing 6 instructions simultaneously, with multiple variations in-between, including asymmetric configurations. The overall effect is a complex design, requiring up to the equivalent of a core in extra wiring. The vMP proposal is much simpler; while some in-between cases may be optimal for some workloads, it is questionable that it will be feasible to tailor the architecture with this degree of exactitude. The much simpler problem of optimal use of multithreaded architectures proved intractable in real systems [21] – despite apparently useful gains in simulation studies.

The vMP approach is in some sense less general than most reconfigurable designs: it only aims to switch between two fixed configurations, sharing as much as possible while minimizing performance compromise versus a pure CMP or clustered design. There is a higher probability with such a simple design that the gains will actually be realizable than with a more complex design – in the worst case, if the sub-optimal mode is chosen, the performance loss should not be significantly worse than if the workload was run on a non-reconfigurable processor of the “wrong” type.

2.4 Summary

There are many approaches to clusters, a few of which have been reviewed here. There is a growing body of work in multicore or CMP designs, a subset of which is again represented here. The range of work on reconfiguration is so large as not to be possible to summarize briefly; the work

listed here is a representative sample of limited-scale reconfiguration, closest in style to that proposed for vMP.

3. COMBINED DESIGN

At any one time, the overall combination of functional units should operate in one of two modes: clustered or CMP.

In clustered mode, there would be a single instruction stream, with instructions scheduled in parallel across the clusters.

For purpose of example, assume each cluster can issue at most 2 instructions simultaneously, of which at most one can be each of an integer, floating point or memory reference instruction. Assume also that there are 4 clusters. In clustered mode then, this is an 8-wide machine. In CMP mode, it is instead a 4-core machine, with each core 2-wide.

Data paths within the functional unit are the same in either case. Differences are in instruction fetch and scheduling, which are done globally in the clustered design, and locally within one core in the CMP design. Branch prediction would also be different, and the clustered design needs a global bypass network to cover cases where a register value was needed by an instruction in a different cluster.

Figure 2 illustrates a combined design, without showing the additional logic required to select between the alternative organizations. Also not shown is the branch predictor. It is assumed in this illustration that any different logic is duplicated (e.g., L1 caches). It may turn out once the detail is worked through that there are alternatives to duplication (e.g., an L1 design that can work either as a multi-ported single L1, or as several independent L1 caches).

Working through these details and others is necessary to demonstrate viability of the design.

4. PROBLEMS TO SOLVE

The combined design would start from the functional units which would be the same in both cases. Instead of a global front end steering instructions to clusters, each CMP core would have its own fetch unit and L1 interface. Ideally, these differences should be accommodated by a small amount of logic, contributing as little as possible to the critical path.

Similarly, the bypass networks in the two cases should be possible to switch between global and local operation. L1 caches represent a significant problem, since the requirements for access differ significantly in the two modes. Another significant problem is register naming: in the clustered case, global register names are distributed, whereas the CMP cores each have their own register files. The steering logic in the clustered design is not needed in the CMP. The bypass networks have different functions in the two cases. Finally, branch prediction is unlikely to be easy to implement in a dual-purpose way, since the requirements of a simple core and an n -times wider cluster are so different.

The programming model is a separate issue but important to address, otherwise hardware viability will not equate to practicality.

Let us examine each of these problems in turn (grouping the

bypass network with registers), and itemise issues for further attention. In all cases, there are general design trade-offs to consider:

- access speed *vs.* space
- access speed *vs.* time to reconfigure
- maximum reuse of components *vs.* simplicity

4.1 First-Level Cache

One solution to the L1 problem would be to have a separate L1 cache for the clustered case, and individual L1 caches for each core in the CMP case. This approach would be simple and would make context switches from the one mode to the other relatively easy. The L1 contents could be frozen between context switches. However, it is not quite that simple if there is (as is true for most recent designs) an L2 cache. If multilevel inclusion was maintained, any replacement of a block in L2 for the inactive mode would require invalidating that block in the dormant L1 cache (possibly with a writeback).

Issues for further attention include then:

- *dual versus shared L1* – can we reconfigure L1 to work as either suitable for CMP mode or clustered mode, or would it be better to waste silicon on a separate L1 for clustered mode?
- *virtual vs. physical addressing* – a virtually addressed cache may make some of the other design trade-offs simpler

4.2 Registers

There are various approaches to register naming in clustered designs. Since the CMP cores will each need a full register file, the simplest approach would be to have a full register file at each core or cluster, and in the clustered case, mark registers as valid or invalid. If a register was invalid, its contents would be requested via the bypass network.

Problems to solve include how to handle context switches between modes, whether the global bypass network could be adapted to local bypass operations and whether aggressive features like register renaming would be implemented (more applicable to the clustered than CMP case).

Issues for further attention include:

- *virtual register naming* – would a general name translation scheme with automatic dumping of registers to secondary storage be an option? Implementation costs need to be considered against the general CMP argument of keeping things simple.
- *automatic register backup* – as a simplification of virtual register naming, would it be an option to back up architectural registers for the dormant mode?
- *bypass network design* – can we generalize the design so that the two modes are different cases, or would it be simpler to have completely separate bypass networks?

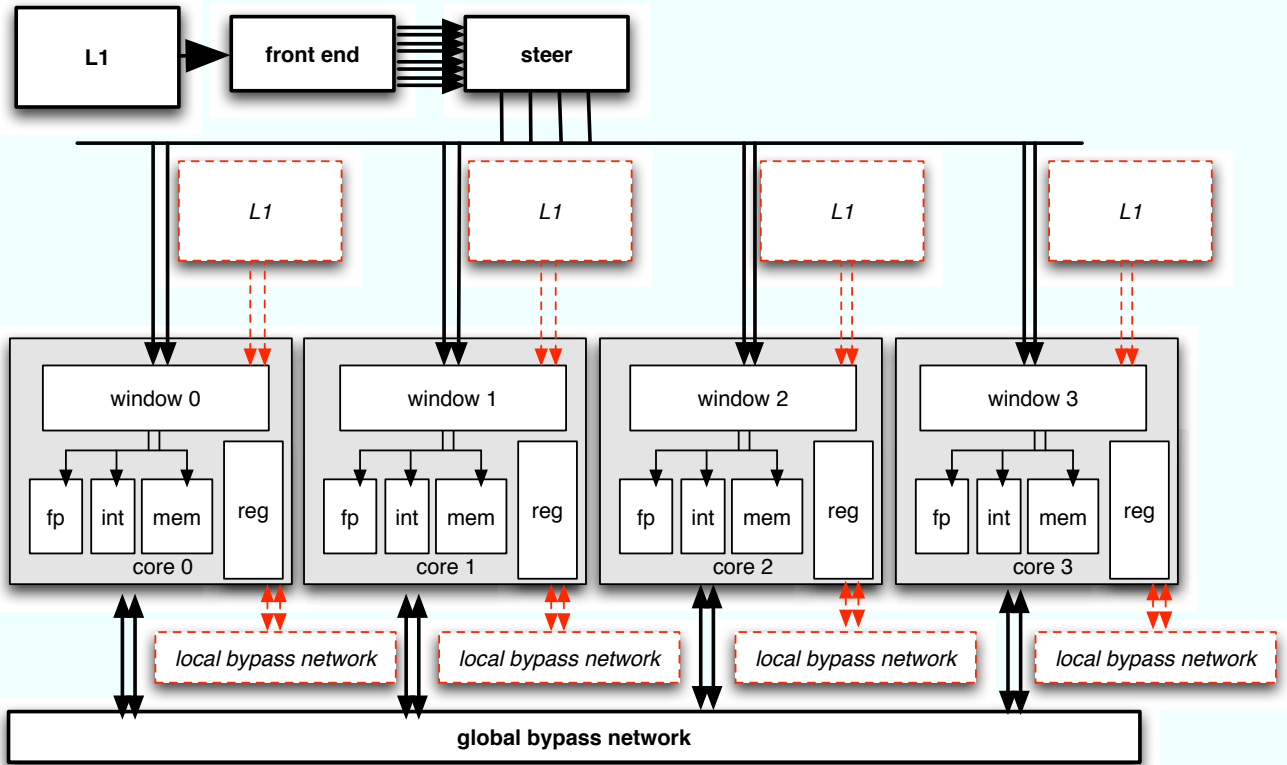


Figure 2: Combined architecture. The CMP differences are shown with red dashed lines and *italicized* text, and the cluster differences in heavy lines and **bold** text.

4.3 Steering Logic

The steering logic would not be necessary in CMP mode if the approach of a separate L1 cache for the clustered mode was adopted. However another option would be to use a single L1 for all cases, and use the steering logic for the CMP case to manage the shared L1 interface.

The main problem to be solved here is whether a sufficiently general mechanism could be designed to serve both purposes.

Issues for further attention include:

- *single versus duplicate L1* – in the single L1 case, the steering logic would route each L1 access at the cost of latency; the gain would be more total L1
- specific L1 design trade-offs include:
 - *L1 size vs. extra logic* – how much extra L1 do we in fact gain once we factor in the extra logic needed for using L1 in two modes?
 - *access speed vs. miss rate* – if L1 becomes slower to handle both modes, do we compensate by lowering the miss rate through having more L1?
 - *context switching costs* – how do the two cases for L1 organization compare when context switches are taken into account?

4.4 Branch Prediction

The most obvious approach for dealing with branch prediction is to use separate logic for the two cases, since many significant issues are different. CMP mode has multiple instruction streams, one per core, whereas the cluster doesn't. The CMP cores should be able to use a relatively simple branch predictor, whereas clustered mode could use a more sophisticated approach.

4.5 Programming Model

Logically, the vMP design will look like an $n+1$ -core design, in which at any one time, n simple or 1 complex processor is actually awake. Having an operating system automatically schedule these $n+1$ processors would be difficult as it would need to know which processes or threads were ILP-centric. However, user-specification of which mode a given process should run on would not be a very onerous model. "User-specification" could be automated by having a compiler determine whether a program was ILP-intensive (e.g., by measures such as the average basic block length).

What would happen on a mode switch depends on the degree of duplication of the memory hierarchy. If caches and registers were not shared across modes, the only issue would be maintaining consistency or inclusion between the "sleeping" mode and lower levels of the hierarchy (or other processors, in a multiprocessor configuration). Ideally, such consistency issues should be handled by the hardware as far as possible,

at worst by the operating system – not by user programs.

In the worst case, if the system is run in the wrong mode for a given workload or portion of workload, it is no worse than a system which did not have the more optimal mode available (e.g., if a process ran on a single core in CMP mode when it could have done better in clustered mode, you would be no worse off than if you had a simple CMP without the vMP feature).

4.6 Putting it All Together

Coming up with a strategy to reuse L1 cache in the two modes presents some interesting challenges; the fallback option of using different L1s in the two modes is the simplest strategy but has a cost in wasted silicon. Register naming also presents some interesting challenges. The simplest approach may again be the best, but design trade-offs will be worth investigating. Bypass, steering and branch prediction logic also present challenges in finding commonality.

The programming model does not present insurmountable problems.

Overall, even if only the functional units are common across the two modes, there is potential for an interesting hybrid design.

5. CONCLUSIONS

5.1 Key Ideas and Issues

The key idea in this proposed design is that there is enough overlap in components between a clustered design and a chip multiprocessor (or multicore) design to design a hybrid, which can operate in either mode.

This hybrid design would look to the software layer like $n+1$ processors, with only either n simple processors or 1 complex processor awake at a time.

The biggest challenge is in reconfiguring logic without adding unacceptable overheads in typical operations. The second biggest is in finding design compromises which work well in both modes.

The basic idea has been explored already in the form of core fusion; the argument in this paper is that the core fusion approach is too complex. The cost of reconfiguration should be kept low, so that suboptimal configurations will not perform significantly worse than a non-reconfigurable design.

Scheduling software on this model should present a few challenges, but should be significantly simpler than most heterogeneous designs, since the two modes run the same instruction set.

5.2 Way Ahead

The next step is to elaborate the design alternatives in more detail, and work out how best to evaluate them.

Strategies include implementing software support in an operating system, doing a complete logic design of selected alternatives, simulation of these alternatives, and evaluation of design trade-offs for performance and total logic.

Given that the core fusion approach has already been evaluated in detail and is considerably more complex than the vMP approach, there is good reason for certainty that vMP is feasible. The major contribution of future studies therefore will be to show that a vMP design does not lose significantly to core fusion despite being considerably simpler to implement.

5.3 Overall Conclusion

It is likely that the simplest approach – recycling the functional units and duplicating everything else – will be the best overall solution. The critical thing about this approach will be to end up with significantly less logic overall than simply implementing a complex processor and n simple processors.

However, other variations will be worth exploring to find the best overall compromise.

References

- C. Albrecht, J. Foag, R. Koch, and E. Maehle. DynaCORE—a dynamically reconfigurable coprocessor architecture for network processors. In *Proc. 14th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing (PDP'06)*, pages 101–108, Montbéliard, France, February 2006.
- AMD. Multi-core processors—the next evolution in computing. Technical report, AMD, 2005. http://multicore.amd.com/GLOBAL/WhitePapers/Multi-Core_Processors_WhitePaper.pdf.
- A. J. Elbirt and C. Paar. An instruction-level distributed processor for symmetric-key cryptography. *IEEE Trans. on Parallel and Distributed Systems*, 16(5):468–480, May 2005.
- K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. The multicluster architecture: reducing cycle time through partitioning. In *Proc. 30th Ann. ACM/IEEE Int. Symp. on Microarchitecture*, pages 149–159, Research Triangle Park, NC, 1997.
- E. Gibert, J. Sánchez, and A. González. An interleaved cache clustered VLIW processor. In *ICS '02: Proc. 16th Int. Conf. on Supercomputing*, pages 210–219, 2002.
- S. Gochman, A. Mendelson, A. Naveh, and E. Rotem. Introduction to intel core duo processor architecture. *Intel Technology J.*, 10(2), May 2006. ftp://download.intel.com/technology/itj/2006/volume10issue02/vol10_art01.pdf.
- R. González, A. Cristal, M. Pericas, M. Valero, and A. Veidenbaum. An asymmetric clustered processor based on value content. In *ICS '05: Proc. 19th Ann. Int. Conf. on Supercomputing*, pages 61–70, 2005.
- D. Graham-Rowe. Logic from chaos: New chips use chaos to produce potentially faster, more robust computing. *Technology Review*, June 2006. http://www.technologyreview.com/read_article.aspx?id=16989&ch=biztech.
- L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun. The Stanford Hydra CMP. *IEEE Micro*, 20(2):71–84, March/April 2000.

- IBM. IBM PowerPC 970MP RISC microprocessor user's manual. Technical report, IBM, 2006. [http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/55661B568F1FE69E87256F8C00686351/\\$file/970MP_um.V2_1.2006JUL31.pdf](http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/55661B568F1FE69E87256F8C00686351/$file/970MP_um.V2_1.2006JUL31.pdf).
- E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez. Core fusion: accommodating software diversity in chip multiprocessors. In *ISCA '07: Proc. 34th Ann. Int. Symp. on Computer architecture*, pages 186–197, 2007.
- A. Jerraya and W. Wolf. Hardware/software interface code-sign for embedded systems. *Computer*, 38(2):63–69, February 2005.
- J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM J. of Research and Development*, 49(4/5):589–604, July–September 2005.
- R. E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March–April 1999.
- T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, S. Reinhardt, K. Flautner, and T. Mudge. Picoserver: Using 3D stacking technology to enable a compact energy efficient chip multiprocessor. In *Proc. 12th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 117–128, San Jose, CA, October 2006.
- R. B. Kujoth, C.-W. Wang, D. B. Gottlieb, J. J. Cook, and N. P. Carter. A reconfigurable unit for a clustered programmable-reconfigurable processor. In *FPGA '04: Proc. 2004 ACM/SIGDA 12th Int. Symp. on Field Programmable Gate Arrays*, pages 200–209, 2004.
- R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Processor power reduction via single-ISA heterogeneous multi-core architectures. *Computer Architecture Letters*, 2(1):2–5, July 2003.
- D. Marculescu. Application adaptive energy efficient clustered architectures. In *ISLPED '04: Proc. 2004 Int. Symp. on Low Power Electronics and Design*, pages 344–349, 2004.
- K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proc. 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-7)*, pages 2–11, Cambridge, MA, October 1996.
- E. Özer, S. Banerjia, and T. M. Conte. Unified assign and schedule: a new approach to scheduling for clustered register file microarchitectures. In *MICRO 31: Proc. 31st Ann. ACM/IEEE Int. Symp. on Microarchitecture*, pages 308–315, 1998.
- Y. Ruan, V. S. Pai, E. Nahum, and J. M. Tracey. Evaluating the impact of simultaneous multithreading on network servers using real hardware. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 315–326, New York, NY, USA, 2005. ACM Press.
- P. Salverda and C. Zilles. A criticality analysis of clustering in superscalar processors. In *MICRO 38: Proc. 38th Ann. IEEE/ACM Int. Symp. on Microarchitecture*, pages 55–66, Barcelona, Spain, 2005.
- B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER5 system microarchitecture. *IBM J. of Research and Development*, 49(4/5):505–521, July/September 2005.
- J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM J. of Research and Development*, 46(1):5–25, January 2002.
- B. F. Veale, J. K. Antonio, and M. P. Tull. Configuration steering for a reconfigurable superscalar processor. In *Proc. 19th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS'05) – Workshop 3*, page 152b, Denver, Colorado, 2005.
- Z. Wang, X. S. Hu, and E. H.-M. Sha. Register aware scheduling for distributed cache clustered architecture. In *ASPDAC: Proc. 2003 Conf. on Asia South Pacific Design Automation*, pages 71–76, 2003.
- J. Zalamea, J. Llosa, E. Ayguadé, and M. Valero. Modulo scheduling with integrated register spilling for clustered VLIW architectures. In *MICRO 34: Proc. 34th Ann. ACM/IEEE Int. Symp. on Microarchitecture*, pages 160–169, 2001.