Zbornik radova 49. Konferencije za ETRAN, Budva, 5-10. juna 2005, tom III
Proc. 49th ETRAN Conference, Budva, June 5-10, 2005, Vol. III

A PROPOSAL FOR REGISTER-LEVEL COMMUNICATION IN A SPECULATIVE CHIP MULTIPROCESSOR

Milan B. Radulović, Milo V. Tomašević[1]
[1]School of Electrical Engineering, University of Belgrade

Abstract – *The advantage of support for register-level communication in speculative Chip Multiprocessors (CMP) has already been clearly recognized in the literature. This paper presents Snoopy Inter-register Communication (SIC) protocol as a hardware mechanism to support the communication of registers values and synchronization between processor cores in a CMP architecture over a shared bus. First, the needed software support that enables the identification of threads from a sequential binary code in loop-intensive applications is described. Then, the states of loop-live and other registers in the protocol are identified. After that, protocol actions during producer-initiated and consumer-initiated communication between threads are defined. Finally, a brief qualitative comparison to IACOMA solution is given.*

1. INTRODUCTION

There are two major approaches in the speculative CMP design. The first approach is related to the CMP architecture that is fully geared towards exploiting speculative parallelism, e.g., Multiscalar [1], Trace processor [2], and SM processor [3] The second approach is related to a generic enough CMP design with only minimal support for speculative execution, e.g., Hydra [4] or STAMPede [5]. The application threads can communicate between themselves through registers or through shared memory. The first approach in the CMP design supports inter-thread communication both through registers and shared memory, while the second approach supports inter-thread communication via shared memory only. Speculative CMP architectures mostly have sufficient hardware support for efficient speculative execution, but this large amount of hardware remains unutilized when running a fully parallel application or a multiprogrammed workload.

The advantage of register-level communication over memory-level communication is in faster execution, since memory-level communication needs the instructions to explicitly store and load the communicated values to and from memory, as well as for synchronization of two communicating threads (unless special hardware support is provided). Earlier work has shown that it is worth the effort to include the support for inter-thread communication in a speculative CMP [6]. Therefore, this proposal considers the possibility to make a simpler and more cost-effective solution for register-level communication in CMP architecture based on the snoopy protocol [7]

2. RELATED WORK

The inter-register communication can be found in some representative CMP architectures. The processing units in Multiscalar [1] share a common register namespace, considering it as a logically centralized register file no matter it consists of physically decentralized register files, queues and control logic. This adds a significant hardware overhead to each Multiscalar processor core, such as duplicate registers along with a set of register masks. In Trace processor [2] a centralized global register file takes care about inter-thread register communications in addition to the local register set in each processor. Unlike Trace processor, the SM processor [3] has all the register files completely distributed. In addition to, each local register file has both a register map table and a register write table to resolve dependences through registers.

However, these CMP architectures have a large amount of hardware support dedicated to speculative execution that remains unutilized when running a fully parallel application or a multiprogrammed workload. It is not the case in IACOMA CMP solution equipped with support for inter-register communication over shared bus, but based on a distributed directory protocol [6]. An entry for a register in a local directory of 4-processor CMP is 12 bit-wide. Our proposal, based on a snoopy protocol, is aimed to decrease the directory overhead notified in IACOMA, as well as the extra logic added to check the register availability and last-copy status for each register in the processor.

3. SIC PROTOCOL

The speculative CMP architecture in this proposal tries to combine the best of two above-mentioned major approaches for configuring multiple processing units on a chip. It is generic enough and has a minimal support for speculative execution, while the inter-processor communication is both through registers and memory.

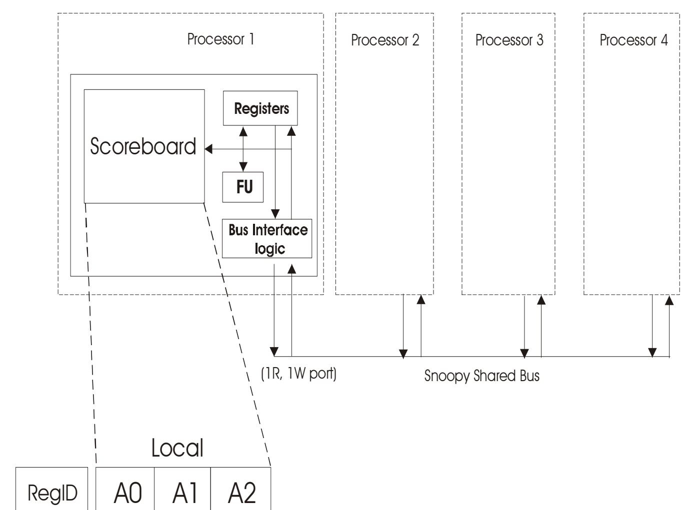

Fig. 1. *Hardware support for register-level communication in proposed CMP architecture*

The proposed CMP architecture consists of four processor cores with private L1 caches connected via shared bus, and shared L2 cache on chip. The hardware support for register communication is presented in Fig.1. It consists of a snoopy shared bus for transferring register values between cores and local scoreboards for keeping the status of registers.

3.1 Binary annotator

To run an application on a speculative CMP architecture the threads have to be identified first. This is achieved in software using the modified binary annotator proposed in IACOMA. The main advantage of using a modified IACOMA binary annotator is in identification of threads without the need for source recompilation. As in IACOMA, the speculative threads are limited to loop iterations. Beside thread identification, the binary annotator also identifies the inter-thread register-level dependences that can be found accurately in the binary code.

The steps involved in annotation process for sequential binary code are illustrated in Fig.2. First, the inner-loop iterations are identified, as well as their initiation and termination points. Then, the inter-thread register dependences are identified. This includes identification of the *loop-live* registers, which are those that are live at loop entry/exits and may also be redefined in the loop, and *other* registers, which are those that cannot be redefined in the loop. From these loop-live reaching definitions IACOMA binary annotator identifies the safe definitions at the points where the register value will never be overwritten by another definition, and, the release points for register values at the exit of the thread (Fig. 3).

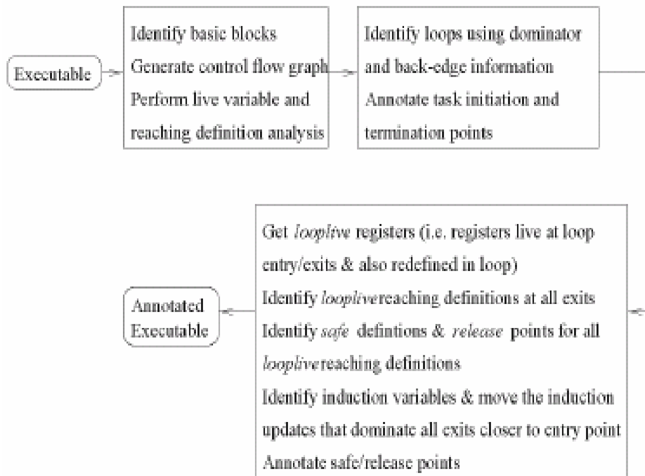

Fig.2. *Binary annotation process [6]*

The SIC requires some modifications of the binary annotator related to the classification of writes to the loop-live registers. Those writes can be divided into non-final writes and final writes (Fig.3). Non-final writes are those that correspond to register values that might be overwritten during execution of loop iteration. Final writes are those that produce the register values that will not be overwritten in any case during the execution of current loop iteration, and, also, correspond to safe definitions in IACOMA binary annotator. All other writes are considered as non-final.

3.2. The SIC infrastructure

The hardware support for register-level communication maintains the status of each thread in the form of a bit mask in a special register. Since four threads can run in parallel in the proposed CMP architecture, a thread status can have one of four masks: 00, 01, 10, and 11. The non-speculative thread (mask-00) is the one that executes the current iteration of the loop, while the speculative successors execute the first (mask-01), second (mask-10) and third (mask-11) successive speculative iteration respectively. The status of non-speculative thread will move from one thread to its immediate successor and so on after a thread completes. It is presumed that threads strictly commit in order to keep the sequential semantics. The thread has to wait to reach the non-speculative status before it can be retired and a new thread to be initiated on the same processor. The register communication between threads in this proposal can be producer-initiated and consumer-initiated.

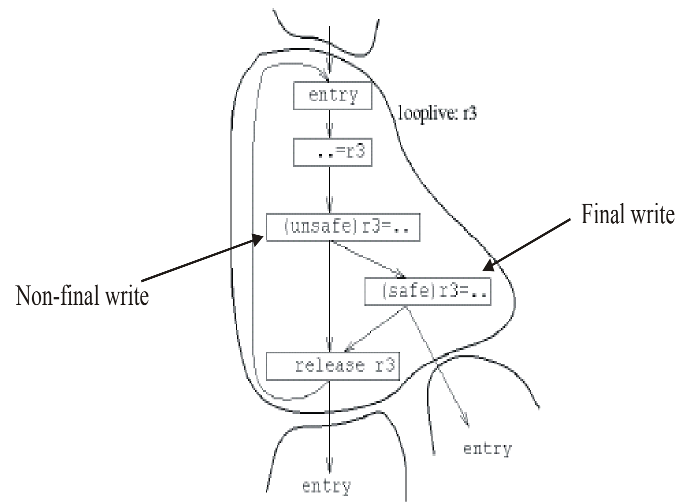

Fig.3. *An example for safe definitions and release points, non-final and final writes*

A loop-live register value in the SIC protocol can be found in one of the following states during the execution: *Invalid (INV)*, *Valid-Unsafe (VU)*, *Valid-Safe (VS)*, and *Last Copy (LC)*. Other registers can be either in *Invalid (INV)* or *Valid-Safe (VS)* states during the execution.

The *Invalid* state indicates that the register's contents is not longer valid, and cannot be used by a local processor or others. The *Valid-Unsafe* state indicates that a given register value is valid for current thread, but it is not a final, safe value than can be forwarded to successor threads. The *Valid-Safe* state indicates that a given register value is valid for current thread, and it is also safe to be forwarded to successor threads. The *Last-Copy* state indicates that a given register value is the only copy of valid register value among all processor cores. Three bits (A0, A1 and A2) per register are provided in local scoreboard (Fig.1), one for distinguishing between loop-live and other registers and two for coding four states for loop-live registers.

The processor core issues two types of requests: read (*R*) and write. Writes can be final (FW) and non-final (NFW). The control part of shared bus consists of *BusR*, *BusW*, *Mask* and *Shared* signals.

Bus-Read (BusR) transaction is a consequence of a processor read that misses in a local scoreboard. The read request is issued on the bus along with the mask code in order to get the requested register value.

Bus-Write (BusW) transaction is a consequence of a processor write request when a speculative thread either reaches Valid-Safe state for a given register, or when a register value in Last-Copy state is sent on the bus. The register value is put on the bus, so the consumer threads can load it. It is important to emphasize that in both cases the register value is transferred on the bus without previous request from the consumer threads.

Mask signals are issued on the bus on read miss when a thread requests the appropriate register value from its closest predecessor. Each predecessor thread with requested register value in Valid-Safe state in its scoreboard replies by posting its mask on bus mask lines, which are realized as wired-OR lines. Consequently, the bus interface logic in each processor core can sense whether any other predecessor posted its mask on the bus. The mechanism of distributed arbitration chooses a closest predecessor thread, which is allowed to put the valid requested register value on the bus.

Shared signal, which is also sent over a wired-OR bus line, is issued by a consumer thread when it responds to a non-demand producer-initiated request for sending a register value, which has been found in Last Copy state. By raising this line, the producer is informed that the register value issued on the bus is loaded elsewhere and it ceases to be the last copy.

3.3. SIC protocol state transitions

The register-level communication in this proposal is based on SIC protocol states and transitions between them. The processor-induced and bus-induced state transitions for loop-live registers in SIC protocol are presented in Fig.4. Since the registers are divided in two groups, loop-live and other registers, the SIC protocol considers them differently. The protocol mechanism for loop-live registers works as follows:

(1) **Read hit.** If a thread, either speculative or non-speculative, issues a read request (R) for a register in VU, VS or LC state, the request is satisfied locally and that register remains in the same state.

(2) **Read miss.** A read request for a register in INV state causes the read miss and initiates *BusR* transaction. Read request will be issued on the bus along with the mask code of speculative thread. Consequently, read miss incurs a consumer-initiated inter-thread communication. All possible suppliers, i.e., predecessors (non-speculative thread or/and earlier speculative threads) that have a requested register either in VS or LC state, reply with issuing their mask codes on the bus and the mechanism of distributed arbitration chooses the latest predecessor. Then, the chosen predecessor thread supplies the requested register value over the bus. The requested register is loaded with VU state in the local scoreboard of the speculative thread that issued a read request. If the requested register was in LC state at the supplier's side, it goes from LC state to VS state. If there is no supplier available at the moment when a consumer thread issued the read request on the bus, the consumer thread blocks.

(3) **Write Hit.** If a register value is in VU state when a thread (either non-speculative or speculative) issues the write request and performs the *non-final write* (NFW), the register is updated locally and it remains in VU state.

However, if a register value is in VU state when a processor issues the write request and performs the *final write* (FW), the register is updated and state is changed from VU state to VS state. Also, the *final write* incurs the producer-initiated communication by sending the register value on the bus for the immediate successor thread. If the value is loaded into the register of immediate successor, it raises *Shared* line informing the producer that the value is received. Also, if the successor was blocked waiting for this particular register value, it becomes running and continues the execution. If the shared line remains inactive (low), the producer changes the state for this register from VS to LC in the local scoreboard.

(4) **Write Miss.** A thread's *non-final write* (NFW) to a register value in INV state in a local scoreboard causes the write miss. However, the write miss does not cause the *BusW* transaction in this case, but only update the register value. Also, the register state is modified from INV state to VU state.

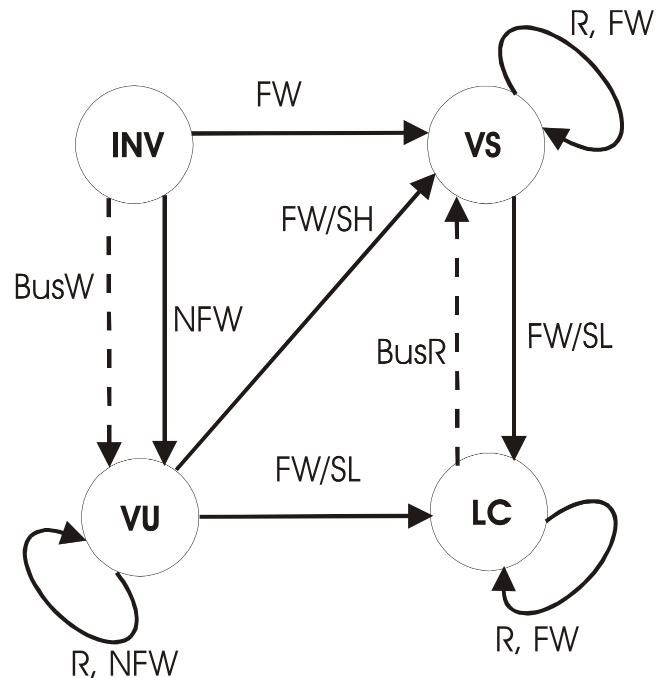

Fig.4. *Producer-initiated and bus-induced state transitions for loop-live registers in SIC protocol. Notation A/B means that processor writes (final write - FW and non-final write - NFW) are observed in relation to Shared signal either high or low (SH, SL). Processor reads are denoted with R. Dashed line style indicates bus-induced state transitions.*

A write miss occurs also in case when a thread performs *final write* (FW) to a register in INV state. This causes the update of the register value and the change of its state from INV state to VS state. Also, write miss incurs a producer-initiated inter-thread communication in case of final write by issuing a *BusW* transaction on the bus. The producer thread puts the register value on the bus, so successor thread can read it. The actions are the same as in the case of write hit.

The protocol mechanism for other registers works as follows:

(1) **Read hit.** If a register value is in VS state when a thread, either speculative or non-speculative, issues a read request, this hit is satisfied locally and the register remains in VS state.

(2) **Read miss.** A thread's read to a register that is in INV state causes the read miss and *BusR* transaction. Read request is issued on the bus along with a mask code of speculative thread. Also, read miss incurs a consumer-initiated inter-thread communication. This case can happen only when a processor runs a speculative thread for the first time after some sequential section of the application. Then, non-speculative or some predecessor thread supplies the requested register value. The requested register value is loaded in VS state with the consumer thread. Also, the other successor threads that have the same register in their local scoreboards in INV state snoop the bus and on this occasion load the sent value in VS state by means of *Read Snarfing* technique [8].

3.4. Thread initiation and completion

When a processor initiates a new speculative thread (the most speculative thread) it is necessary to invalidate any loop-live register by setting their state to INV. However, the other registers remain in their current state during the new thread initiation. When a thread is about to complete, a search for last copies (LC state) in local scoreboard is carried out. The thread is allowed to complete only if there are no registers in this state. Otherwise, for each register in LC state its value must be saved by sending it over bus to its successor, in the same way as in the processing of final write in producer-initiated communication.

4. COMPARISON

Quantitative evaluation of the SIC proposal is not carried out yet, but some qualitative comparison to IACOMA [7] (as an existing solution of the same kind) can be made. As for the hardware support for register-level communication, the SIC scheme significantly reduces the directory overhead as well as the overhead in extra logic for checking the register availability and last-copy status for each register. Besides, by means of mechanism of distributed arbitration, SIC protocol finds the closest predecessor in an easier and faster, non-sequential way than in IACOMA. The proposed software support involves only minimal changes in the binary annotator tool for thread identification

Also, the SIC protocol employs the Read-Snarfing technique in communication for non-looplive registers that decrease a number of misses in comparison to IACOMA and helps to validate the other registers more quickly.

5. CONCLUSION

The SIC proposal for register-level communication in CMP architecture is a solution for register-level communication based on the snoopy protocol which tries to improve over the existing solutions. It has been shown that hardware and software support for inter-register communication is quite acceptable and effective to provide correct speculative execution. The future research directions will be oriented towards qualitative evaluation of the proposed hardware support for register-level communication by simulation means. Also, some ideas has already appeared about modifying the protocol into the direction of more aggressive speculation by forwarding even possibly final write values in order to obtain further performance gain. Finally, an extension of the protocol functionality and making the binary annotator tool possible to deal with other kinds of speculations (e.g. subprogram calls, etc.) would be very desirable.

REFERENCES

[1] G.S. Sohi, S. Breach, and T.N. Vijaykumar, "Multiscalar Processors", *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 414-425, May 1992.

[2] E.Rotenberg, Q. Jacobson, Y. Sazeides, J.E. Smith, et al, "Trace Processors", *Micro-30*, pp. 68-74, December 1997.

[3] P.Marcuello, A. Gonzalez, "Control and data dependence speculation in multithreaded processors", *HPCA-4*, February 1998.

[4] Lance Hammond et all., "The Stanford Hydra CMP", *IEEE MICRO Magazine*, pp., 71-84, March-April 2000.

[5] J. G. Steffan, T. Mowry, "The potential for using thread-level data speculation to facilitate automatic parallelization", *Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, Las Vegas, NV, February 1998.

[6] V. Krishnan, J.Torrellas, "A Chip-Multiprocessor Architecture with Specualtive Multithreading", *IEEE Transactions on Computers*, Vol. 48, No. 9, September. 1999.

[7] M. Tomašević and V. Milutinović, *Tutorial on the Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions*, IEEE Computer Society Press, Los Alamitos, Calif., 1993.

[8] S.J. Eggers, R.H. Katz, "Evaluating the performance of four snooping cache coherency protocols", *Proceedings of the 16th International Symposium On Computer Architecture*, pages 2-15, 1989.

Sadržaj – Prednost postojanja podrške u spekulativnom multiprocesorskom sistemu na čipu (CMP) za komunikaciju na nivou registara je jasno pokazana u ranijim studijama. U ovom radu se predlaže SIC (Snoopy Inter-register Communication) protocol. On predstavlja hardverski mehanizam koji omogućava razmenu podataka na nivou registara i sinhronizaciju procesora u CMP sistemu preko zajedničke magistrale. Prvo je opisana neophodna softverska podrška za identifikaciju niti iz izvršnog koda sekvencijalne aplikacije koja sadrži dosta ciklusa. Onda su identifikovana stanja koja mogu imati registri koji se menjaju ili ne menjaju u nekoj niti kao i ostala infrastruktura potrebna za protokol. Zatim su detaljno opisane akcije protokola i prelazi između stanja pri komunikaciji iniciranoj od strane proizvođača i korisnika. Na kraju je dato grubo kvalitativno poređenje sa sistemom IACOMA koji pripada istoj klasi.

PREDLOG REGISTARSKE KOMUNIKACIJE U SPEKULATIVNOM MULTIPROCESORU NA ČIPU

Milan B. Radulović, Milo V. Tomašević