Shared Control – Supporting Control Parallelism Using a SIMD-like Architecture

Nael B. Abu-Ghazaleh[1] and Philip A. Wilsey[2]

[1] Computer Science Dept.,
State University of New York
Binghamton, NY 13902-6000
[2] Department of ECECS, PO Box 210030
University of Cincinnati,
Cincinnati, Ohio 45221–0030

Abstract. SIMD machines are considered special purpose architectures chiefly because of their inability to support control parallelism. This restriction exists because there is a single control unit that is shared at the thread level; concurrent control threads must time-share the control unit. We present an alternative model for building centralized control architectures that better supports control parallelism. This model, called *shared control*, shares the control unit(s) at the instruction level — in each cycle the control signals for the supported instructions are broadcast to the PEs. In turn, a PE receive its control by synchronizing with the control unit responsible for its current instruction. There are a number of architectural issues that must be resolved. This paper identifies some of these issues and suggests solutions to them. An integrated shared-control/SIMD architecture design (**SharC**) is presented and used to demonstrate the performance relative to a SIMD architecture.

1 Introduction

Parallel architectures are classified according to their control organization as Multiple Instruction streams Multiple Data streams (MIMD), or Single Instruction stream Multiple Data streams (SIMD) machines. MIMD machines have a *distributed control organization*: each Processing Element (PE) has a control unit and is able to sequence a control thread (program segment) locally. Conversely, SIMD machines have a *centralized control organization*: the PEs share one control unit. A single thread executes on the control unit, broadcasting instructions to the PEs for execution. Because the control is shared and the operation is synchronous, SIMD PEs are small and inexpensive.

In a centralized control organization (*e.g.*, SIMD [11], [14] and MSIMD [5], [18] machines), an arbitrary number of PEs share a fixed number of control units. Traditionally, sharing of control has been implemented at the *thread level*; the PEs following the same thread concurrently share a control unit. The presence of application-level control-parallelism causes the performance of this model to drop (proportionately to the degree of control parallelism). This drop occurs

because the control units are time-shared among the threads, with only the set of PEs requiring the currently executing control thread actively engaged in computation. General parallel applications contain control parallelism [3], [7] and, therefore, perform poorly on SIMD machines. Accordingly, SIMD machines have fallen out of favor as a platform for general-purpose parallel processing [10], [15].

This paper presents *Shared Control*: a model for constructing centralized control architectures that better supports control-parallelism. Under shared control, the control units are shared at the instruction (or atomic function) level. Each PE is assigned a local program and PEs executing the same instruction, but not necessarily the same thread, receive their control from the same control unit. A control unit is assigned to each instruction, or group of similar instructions, in the instruction set and, thus, broadcasts the microinstruction sequences to implement that instruction repeatedly to the PEs. Each PE receives its control by synchronizing with the control unit corresponding to its current instruction. Thus, all the PEs are able to advance their computation concurrently, regardless of the degree of control parallelism present in the application. The similarity of the hardware to the SIMD model allows the SIMD mode to be supported at little additional cost. With the ability to support control-parallelism efficiently, the major drawback of the SIMD model is overcome.

Shared control is a unique architectural paradigm; the classic association between control units and threads, present in all Von-Neumann based architectures, does not exist in this model. Therefore, it introduces several architectural issues that are unique to it. This identifies some of these issues and discusses solutions to them. The feasibility of the solutions is demonstrated using a SIMD/shared-control architecture design, **SharC**. Using a detailed simulator of **SharC**, the performance of the model is studied for some irregular problems. The remainder of this paper is organized as follows. Section 2 introduces the shared control model. Section 3 presents some architectural issues relating to a general shared control implementation. Section 4 presents a case study of a shared-control architecture. In Section 5, the performance of the architecture is studied. Finally, Section 6 presents some concluding remarks.

2 Shared Control

A shared control architecture is a centralized control architecture where the control units are shared at the operation level. PEs executing the same operation, but not necessarily the same thread, may share the use of the same control unit. An overview of a shared control machine is shown in Figure 1. The control program (microprogram) implementing the instruction set for the shared control mode is partitioned across a number of tightly coupled control units. This partitioning is static; it is carried out at architecture design time. Each control unit repeatedly broadcasts the microprogram sequence assigned to it to the PEs. A PE receives its control from the control unit associated with its current instruction. The PE synchronizes with the control unit by selecting the set

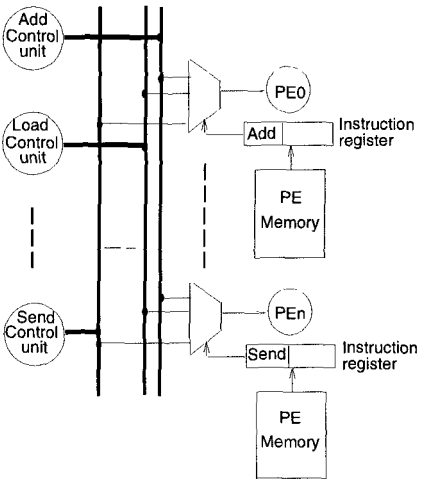

Fig. 1. Overview of a Shared Control Organization

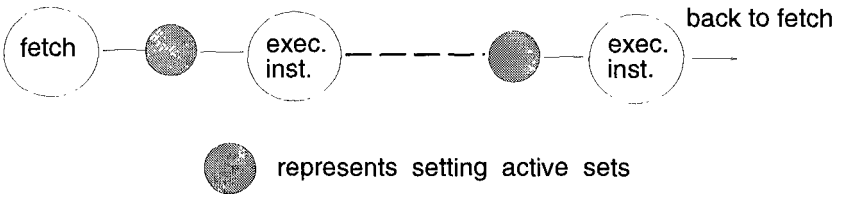

Fig. 2. Implementation of Several Instructions on a Single Control Unit

of control signals broadcast by that control unit. PEs are able to advance their computation concurrently; thus, MIMD execution is supported.

We first consider the problem of implementing and optimizing shared control using a single control unit; this is a special case that is needed for the general solution. Figure 2 shows the single control unit implementation. The control unit must supply control for all the instructions in every cycle. More precisely, the control unit sequentially issues *all* the paths through the microprogram, and the PEs conditionally participate in the path corresponding to their current instruction. In the remainder of this section, some of the architectural issues involved in constructing a single-control unit shared control multiprocessor are discussed.

Managing the Activity Status: Before every instruction execution stage, the activity bits for the PEs that are interested in this stage must be set (represented by the shaded circles in Figure 2). On SIMD machines setting the activity status before a conditional region requires the following operations on the PEs: (i) save current active set, (ii) evaluate the condition, and (iii) set active bit if condition is true. In addition, at the end of the conditional region the active set saved in

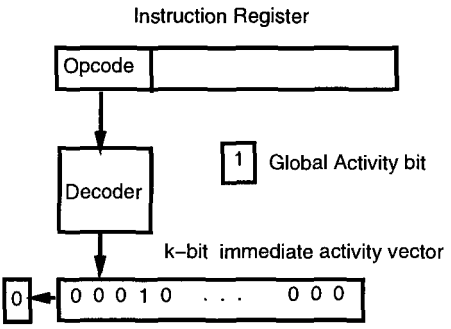

Fig. 3. Activity Set Management

step (i) is restored. The active bit can be tracked by saving the bit directly to an activity bit stack [13], or by using an activity counter that is incremented to indicate a deeper context [12]. Both schemes have a significant overhead (register space, as well as several execution steps). Implementing activity management using the SIMD model adds unacceptable overhead to the shared control cycle time since it is required several times per cycle in shared control.

The condition for the activity of a PE for instruction stage (called the *immediate* activity) is contained in the opcode field in the instruction register, allowing the following optimization to be made. The opcode field is decoded into a k-bit vector (as shown in Figure 3). At the beginning of every instruction segment, the bit vector is shifted into the immediate activity bit. Thus, instruction i is decoded into a bit vector consisting of 1 in the i^{th} position and 0 elsewhere, mirroring the order of the execution regions. Only PEs with a high immediate activity bit participate in the current instruction segment. The register shift can be performed concurrently with the execution of each region at no additional cost; activity management cost is eliminated.

A compositional instruction set: Examining Figure 2), it can be observed that each PE receives the control for all k instructions, but uses only one. Compositional instruction sets relax this restriction by allowing the PEs to use any number of the k instruction segments in each cycle [4]. The output of each execution region is deposited in a temporary register that serves as an input to the next one. The output of the last stage is stored back to the final destination. Thus, an instruction is represented by the subset of the instruction segments that compose it. Composition can be easily incorporated into the activity management scheme in Figure 3.

3 General Shared Control

A general implementation of the shared control model uses multiple control units to implement the microprogram for the instruction set. There are a number of architecture issues that are introduced by this model (in addition to the ones

present in the single control unit implementation). The discussion in this section will cover the range of possible solutions to each issue, rather than consider specific solutions in detail.

Control I/O pins: At first glance, this is the primary concern regarding the feasibility of shared control; because the model uses multiple control units, the width of the broadcast bus and the number of required pins at the PEs may become prohibitive. Fortunately, the increase in the number of pins is not linear with the number of control units because: (i) each control unit is responsible only for an instruction or a group of instructions; (ii) literal fields and register number fields are not broadcast; they are part of the instruction (they have to be broadcast in SIMD and MSIMD architectures); and (iii) pins that carry the same values on different control units throughout their lifetime are routed as a single physical pin.

The Control Broadcast Network: The control broadcast network is responsible for delivering the control signals from the control units to the PEs. Traditionally, control broadcast has been a bottleneck on centralized control machines; solutions to this problem include pipelining of the broadcast [3], and caching the control units closer to the PEs [16]. With advances in fabrication technology, there is a trend to move processing to memory [8]. For such systems, the control units may be replicated per chip, simplifying the control broadcast problem.

Control Unit Synchronization and Balance: The control stream required by each PE is not supplied *on demand* as per traditional computers. Rather, the control units have to be synchronized such that the control streams are available when a PE needs them (with minimal waiting time). A possible model to synchronize the control units is to force them to share a common cycle length, called *fundamental instruction cycle*, determined by the slowest control unit. However, the instruction cycle time may vary widely for the instructions in the instruction set, forcing long idle times for PEs with short instructions. Fortunately, there are a number of synchronization models that reduce PE idle time, including: (i) issuing the long instructions infrequently; (ii) allowing the short instructions to execute multiple times while the long instruction is executing; and (iii) breaking long instructions in a series of shorter instructions [2].

Support for Communication and I/O: Support of communication, I/O and other system functions poses the following problem: the system must provide support for both SIMD and MIMD operation at a cost that can be justified against the simple PEs. We focus this discussion on the communication subsystem. There are two options for the support communication. The first option restricts the support to the SIMD mode; the more difficult problem of supporting MIMD-mode communication is side stepped. Restricting communication to SIMD mode is inefficient because: (i) all the PEs have to synchronize and switch back to SIMD if any of them needs to communicate, (ii) because of SIMD semantics, all the PEs must wait for the PE with the worst path communication; a bottleneck of synchronous operation when irregular communication patterns are required [6]. Another alternative is to support MIMD operation using an inexpensive network [1].

4 A Case Study: SharC

In this section we present an overview of the architecture of *SharC*, a proof-of-concept shared control architecture [1]. We preface this discussion by noting that the *SharC* architecture was designed for possible fabrication within a very small budget (10,000 US dollars for a 64 PE prototype). The PE chip fits within the MOSIS small chip (a very small chip size with 100 I/O pads); much higher integration is possible with more aggressive technology. While *SharC* does not represent a realistic high-performance design using today's technology, it can still be used as an impartial model to investigate the shared-control performance relative to SIMD performance.

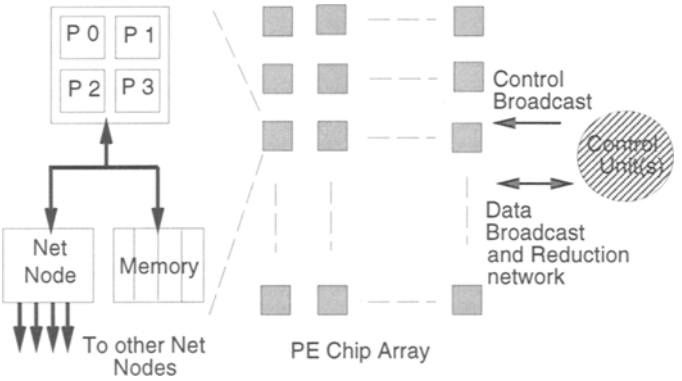

Fig. 4. System Overview

Figure 4 presents an overview of the system architecture. The shared control subsystem consists of 9 control units. The fundamental cycle for all the control units is 11 cycles, with the exception of the load/store control units which require 22 cycles. There are 4 PEs per chip sharing a single memory port. Communication occurs using the same memory port as well: 4 cycles of the 11-cycle fundamental cycle are reserved for the exchange of two message with the network processor (one each way). Most of the integer operations are mapped to the same control unit and implemented using composition. The fundamental cycle is 11 cycles long, constrained by the access time of the shared memory/communication port. If dedicated (non-shared) memory ports are supplied, this length can be reduced to 4 cycles for most instructions.

As was expected, the number of control pins for the shared control mode exceeded the number required by SIMD mode. However, the number of pins only increased from 112 pins for the SIMD mode to 118 pins for the shared control mode; the increase is small because of reasons discussed in Section 3. The average number of Clocks Per Instruction (CPI) for the shared control mode is higher than the CPI for SIMD mode; most SIMD instructions require less than

11 cycles to implement. The reason for the higher CPI is the time required for the additional fetch required for each instruction, as well as the idle cycles required to synchronize the control units. Thus, for a pure data-parallel applications, a SIMD implementation yields better performance than shared control. However, as the degree of control parallelism increases, the performance of shared control remains constant, while SIMD performance degrades.

The SHARC communication network is a packet-switched toroidal mesh network, with a chip (4 PEs) at each node sharing a network processor. The network processor is made simple by using an adaptive deflection routing strategy (no buffering is necessary) [9]. In many cases, deflection routing results in better performance than oblivious routing strategies (path taken by packet independent of other packets), because excess traffic is deflected away from congested areas; creating better network load balance.

5 Performance Analysis

A detailed simulator of a scalable configuration of the **SharC** design is used to study its performance. The model incorporates a structural description of the control subsystem at the microcode level; it also serves to verify the correctness of the microcode. The test programs were written using assembly language, and assembled using a macro assembler. The assembler supports SIMD/SPMD programming model; an algorithm is written as a SIMD/SPMD application, and any region can be executed in either mode (allowing mixed-mode programming). A transition from SIMD to MIMD operation is initiated explicitly by the SIMD program (using a special `c_mimd` instruction). A transition from MIMD back to SIMD occurs when all the PEs have halted (using a `halt` instruction); implementing a full barrier synchronization.

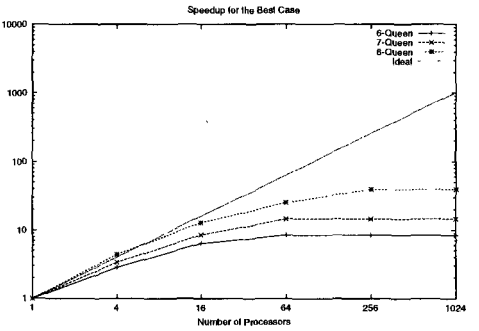

Fig. 5. Speedup for the N-Queen Problem

Our first example is the N-Queen problem: a classic combinatorial problem that is easy to express but difficult to solve. The N-Queen problem has recently

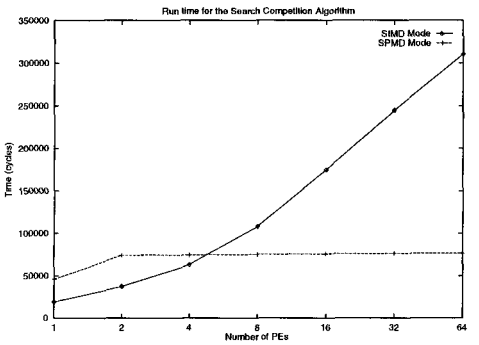

Fig. 6. Search Competition Algorithm Performance

found applications in optical computing and VLSI routing [17]. We considered the exhaustive case (find all solutions, rather than find a solution). The N-Queen problem is highly irregular and not suited to SIMD implementation. Figure 5 shows the performance of the shared control implementation normalized to that of the SIMD implementation. The SIMD solution was worse than a sequential solution because it failed to extract any parallelism, but incurred the activity management overhead. The speedup leveled because the of parallelism present at the small scales of the problem that were studied is limited.

The second application is a database search competition algorithm; an algorithm characteristic of massively parallel database operations. Each PE is initialized with a segment of a database (sorted by keys), and the PEs search for keys in their database segment. As the number of PEs is scaled, the size of the database is scaled (the size of the segment at each PE is kept the same). For a small number of PEs, there is little control parallelism, and SIMD performs better than shared control. As the number of PEs is increased, the paths through the database followed by each PE diverge according to their respective data. The performance of the SIMD implementation drops with the increased number of PEs while shared control performance remains constant.

Finally, we consider a parallel case statement with balanced cases (Figure 7). By varying the number of cases in the case statement, the degree of control parallelism is directly controlled. The instruction mix within the SIMD blocks generated randomly, with a typical RISC profile comprised of 30% load/store instructions with branches forced to the next address (allowing for the pipelined instruction). The blocks are chosen to of balanced lengths (10% variation). The program was simulated in three different ways: a SIMD implementation, a shared control (SPMD) implementation, and a mixed mode implementation. The mixed mode implementation executes the switch statement in the shared control mode, but executes block A and block B in SIMD.

Figure 8 shows execution times as a function of the number of cases. Not surprisingly, the performance of the SIMD mode degrades (linearly) with the increased control parallelism injected by increasing the number of cases. Both

```
plural p = p_random(1, n);

[SIMD BLOCK A]
switch(p) {
    case 1: [SIMD BLOCK 1];
            break;
          .  .
    case n: [SIMD BLOCK n];
    }
[SIMD BLOCK B]
```

Fig. 7. Parameterized Parallel Branching Region

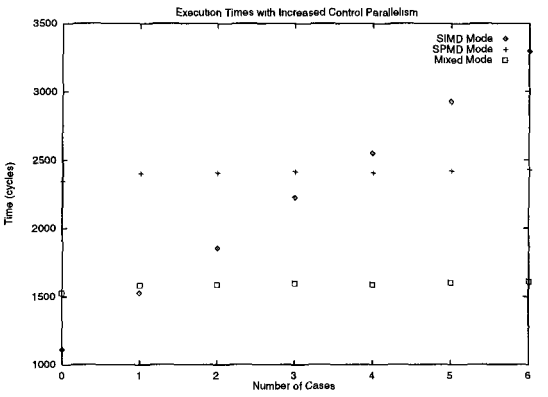

Fig. 8. Simulation results for the architecture on a parameterized branching region

the mixed mode and MIMD execution times did not change significantly with the added control parallelism. The mixed mode implementation is more efficient than the full shared control implementation because of its superior performance on the leading and trailing SIMD blocks.

6 Conclusions

SIMD machines offer an excellent price to performance alternative for applications that fit their restricted control organization. Commercial SIMD machines with superior price to performance ratio continue to be built for specific applications. Unfortunately, centralized control architectures (like the SIMD model) cannot support control parallelism in applications; the control unit has to sequentially broadcast the different control sequences required by each of the control threads. In this paper, we present a model for building centralized control architectures that is capable of supporting control parallelism efficiently. The shared control model is less efficient than the SIMD model on regular code sequences (it requires an additional instruction fetch, and memory space to hold the task

instructions). When used in conjunction with the SIMD model, irregular regions of applications can be executed in shared control, extending the utility of the SIMD model to a wider range of applications.

The shared control model is fundamentally different from the SIMD model. Therefore, there are a set of architectural and implementation issues that must be addressed before an efficient shared control implementation can be realized. The performance of **SharC** was studied using a detailed RTL simulator. Applications were implemented in the SIMD mode, and in the shared control mode. The performance of the shared control implementation was compared to a pure SIMD implementation for several applications (the highly irregular N-queen, and massively parallel database search algorithms were used). Even at the small scales of the problems considered, shared control resulted in significant improvement in performance over the pure SIMD implementation. Unfortunately, larger applications (or problem sizes of the studied applications) could not be studied because: (i) simulating a massively parallel machine on a uniprocessor is inefficient: the simulation run time for the 1024 PE 8-queen problem was in excess of 20 hours on a SunSparc 100 workstation, and (ii) we do not have a compiler for the machine; all the examples were written and coordinated in assembly language. Finally, we used a parameterized conditional region benchmark to demonstrated that shared control is beneficial for applications where even a small degree of control parallelism is present.

References

1. Abu-Ghazaleh, N. B. *Shared Control: A Paradigm for Supporting Control Parallelism on SIMD-like Architectures.* PhD thesis, University of Cincinnati, July 1997. (in press).
2. Abu-Ghazaleh, N. B., and Wilsey, P. A. Models for the synchronization of control units on shared control architectures. *Journal of Parallel and Distributed Computing* (1998). (in press).
3. Allen, J. D., and Schimmel, D. E. The impact of pipelining on SIMD architectures. In *Proc. of the 9th International Parallel Processing Symp.* (April 1995), IEEE Computer Society Press, pp. 380–387.
4. Bagley, R. A., Wilsey, P. A., and Abu-Ghazaleh, N. B. Composing functional unit blocks for efficient interpretation of MIMD code sequences on SIMD processors. In *Parallel Processing: CONPAR 94 - VAPP VI* (September 1994), B. Buchberger and J. Volkert, Eds., vol. 854 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 616–627.
5. Bridges, T. The GPA machine: A generally partitionable MSIMD architecture. In *Proceedings of the 3rd Symposium on the Frontiers of Massively Parallel Architecutres* (1990), pp. 196–203.
6. Felderman, R., and Kleinrock, L. An upper bound on the improvement of asynchronous versus synchronous distributed processing. In *Proceedings of the SCS Multiconference on Distributed Simulation* (January 1990), vol. 22, pp. 131–136.
7. Fox, G. What have we learnt from using real parallel machines to solve real problems? Tech. Rep. C3P–522, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA 91125, March 1988.

8. Gokhale, M., Holmes, B., and Iobst, K. Processing in memory: The Terasys massively parallel PIM array. *IEEE Computer* (April 1995), 23–31.
9. Greenberg, A., and Goodman, J. Sharp approximate models of deflection routing in mesh networks. *IEEE Transactions on Communications 41* (Jan. 1993).
10. Hennesy, J. L., and Patterson, D. A. *Computer Architecture a Quantitave Approach, Second Edition.* Morgan Kaufman Publishers Inc., San Mateo, CA, 1995.
11. Hillis, W. D. *The Connection Machine.* The MIT Press, Cambridge, MA, 1985.
12. Keryell, R., and Paris, N. Activity counter: New optimization for the dynamic scheduling of SIMD control flow. In *1993 International Conference on Parallel Processing* (Aug. 1993), vol. 2, pp. 184–187.
13. MasPar Computer Corporation. *MasPar Assembly Language (MPAS) Reference Manual.* Sunnyvale CA, July 1991.
14. Nickolls, J. The design of the MasPar MP-1. In *Proceedings of the 35th IEEE Computer Society International Conference* (1990), pp. 25–28.
15. Parhami, B. SIMD machines: Do they have a significant future? *Computer Architecture News* (September 1995), 19–22.
16. Rockoff, T. SIMD instruction caches. In *Proceedings of the Symposium on Parallel Architectures and Algorithms '94* (May 1994), pp. 67–75.
17. Sosic, R., and Gu, J. Fast search algorithms for the N-queens problem. *IEEE Transactions on Systems, Man, and Cybernatics* (Nov. 1991), 1572–1576.
18. Weems, C., Riseman, E., and Hanson, A. Image understanding architecture: Exploiting potential parallelism in machine vision. *Computer* (Feb 1992), 65–68.