論文93-30B-11-2

병렬구조 컴퓨터에서 Branch penalty를 감소시키기 위한 소프트웨어와 하드웨어 방법

(A Software And Hardware Scheme For Reducing The Branch Penalty In Parallel Computers)

咸 瀣 淑*, 趙 鍾 顯*, 趙 榮 一*

(Chan Sook Ham, Jong Hyun Cho and Young Il Cho)

要 約

한 사이클내에 다수의 조건을 검사할 수 있는 VLIW 구조는 다중분기를 위한 효과적인 메카니즘이 제공되야한다. 본 논문에서는 다중분기문의 수행시간을 개선시키는 메카니즘과 낭비되는 메모리 공간을 감소시키는 효과적인 명령어 배치 방법을 제안한다. 또한, 다중분기가 적용될 수 없는 프로그램 세그먼트를 다중분기가 적용가능도록 변환하는 새로운 컴파일러 기술을 제안한다. 그런 변환에 의해 branch penalty를 감소시키고 명령어 수준 병렬성을 증가시킬 수 있다는 잇점을 얻을 수 있다.

Abstract

VLIW architecture capable of testing multiple conditions in a cycle must support an efficient mechanism for multi-way branches. This paper proposes a mechanism to speed up the execution of multi-way branches and an efficient memory packing method of instructions, which reduces the wasted memory space. Also, we develops a new compiler technique which can transform program segments that are not applied to multi-way branches into ones that are applied to multi-way branches. The benefits gained by the transformation are to reduce branch penalty and to increase instruction-level parallelism.

Ⅰ. 서론

VLIW(Very Long Instruction Word) 구조 컴퓨터는 여러 명령들이 동일 사이클에 수행되고 파이프라인이 가능한 다수의 functional units을 갖으며 이것들이 1개의 긴 명령어로 제어되는 명령어 수준 병렬 구조 컴퓨터이다.[2] VLIW 구조 컴퓨터는 과학 계산 프로그램에 대해서는 병렬 수행성에 의해 상당한 속도 향상을 얻을 수 있다.[3,4] 그러나 이러한 구조의 컴퓨터는 사용자가 병렬성을 고려하여 기계어로 프로그램하는 것은 불가능하고, 또한 종래의 컴파일러 기술들은 기본단위 블럭을 대상으로 병렬성을 생성하기 때문에 기본 블럭에 있는 명령어 수의 제한으로 인해 병렬성을 충분히 찾을수 없기 때문에 컴퓨터의 functional units을 최대로 이용할수 없다. 따라서 VLIW 구조에 적합한 새로운 컴파일러 기술이 요구된다. Fisher가 제안한 Trace scheduling을 이용한 Ellis의 BULLDOG 컴파일러[4]는 기본 블럭의 범위를 벗어나 병렬성을 찾기 때문에 상당한 속도 개

*正會員, 水原大學校 電子計算學科
(Dept. of Computer Science, Suwon Univ.)

이 논문은 1991년도 교육부지원 학국학술진흥재단의 자유공모과제 학술연구조성비에 의하여 연구되었음 (과제명:"병렬구조 컴퓨터의 컴파일러에 관한 연구")

接受日字 : 1992年 7月 2日

선을 이룩하였다. 그러나 BULLDOG 컴파일러는 수치 과학 계산 프로그램인 경우 만족할 만한 결과를 얻을 수 있으나, 비 수치프로그램인 경우 많은 분기 명령이 존재하고, 분기 명령들의 분기확률이 비슷할 경우 나중에 선택되는 경로에 대해서는 많은 copy 명령으로 인해 수행 시간이 더 증가될수 있다는 문제점을 갖고 있다. 또한 분기 명령들은 제어 종속 관계를 야기시키기 때문에 명령어 수준 병렬성을 제약하고, 파이프라인시 branch penalty를 발생시킨다. 이러한 branch penalty를 감소시키기 위한 방안으로 하드웨어적인 방법과 소프트웨어적인 방법이 제안 되었다. 하드웨어적인 방법들은 대부분이 분기 예상의 정확성에 따라 조건 분기 명령에 의한 branch penalty 정도가 크게 영향을 받게 되고, 분기 예상이 잘못됐을 경우 파이프 라인을 flush 시키기 위한 하드웨어를 제공해야 한다.[12] 소프트웨어적인 방법으로 delayed branch 방법[11]은 컴파일시 분기 방향에 관계없이 모두 수행되는 명령어를 지연 스롯에 삽입함으로써 branch penalty를 감소시키나 지연 스롯의 수가 많은 경우 삽입 시킬수 있는 명령어 수의 제약 때문에 많은 NOP(Nooperation)가 삽입 된다.

본 논문에서는 컴파일시 다수의 조건 분기문을 다중 분기 명령으로 변환 시킴으로써 다수의 조건 분기문에 의한 branch penalty를 감소시키고 명령어 수준 병렬성을 향상시키는 방법과 그러한 다중 분기 명령을 효과적으로 처리할수 있는 다중 분기 하드웨어 메카니즘을 제안한다. 또한 다중 분기 명령이 적용될 수 없는 프로그램 세그먼트를 적용 가능하도록 변환하는 컴파일러 기술을 제안한다.

Ⅱ. 다중 분기 명령으로의 변환

이 장에서는 다중 분기가 적용 불가능한 프로그램 세그먼트를 적용 가능하도록 변환하는 방법에 대해 서술한다. 분기 명령은 VLIW에서 2개의 부정적인 요인을 갖는다. 첫번째는 프로그램에서 제어 종속 관계를 야기시킴으로써 명령어 수준 병렬성을 상당히 제약 시키는 원인이 된다는 것이고, 두번째는 branch penalty이다. 분기 명령은 파이프 라인 프로세서에서 지연 스롯(delay slot)을 발생시킨다. 클럭 사이클당 N개의 명령어를 수행하는 파이프라인 VLIW 머신에서 N개의 지연 스롯이 삽입 되는데 종전의 연구 조사에 의하면 컴파일 중 명령어 재배치 에 의해 1개의 지연 스롯을 채울 확률은 약 70%인 반면 2개의 지연 스롯을 채울 확률은 25% 정도이다.[7] 따라서 분기 명령의 수를 감소 시키기 위해 다수의 분기 명령을 다중 분기 명령으로 변환 시키는 것은 많은 지연 스롯을 제거 시킴으로 인해 성능 향상의 상당한 개선을 줄 수 있다. 그러나 실제 프로그램의 목적 코드에서 보면 직접 다중 분기 명령으로 변환이 가능한 조건 분기 명령이 연속적으로 이어지는 경우는 적으므로 다중 분기 명령으로 변환 시킬 수 있는 부분은 극히 제한적이다. 따라서 분기명령이 연속적으로 이어지지 않는, 즉, 분기 명령 사이에 비분기 명령이 있는 프로그램 세그먼트를 임시 기호 변수를 사용하여 분기 명령이 연속적으로 이어지도록 하여, 다중 분기 명령으로의 변환이 필요하며 그 변환 과정은 다음과 같다.

단계 1. 다른 분기 명령을 dominate하는 분기 명령 Bd와 Bd 바로 다음에 위치한 기본블럭에 있는 분기 명령 B1,B2,....,Bn를 변환 대상으로 설정한다.

단계 2. Bd와 Bi사이에 있는 명령어들 중에서 Bi와 데이타 종속 관계가 없는 명령어들은 Bi에 의해 분기되는 모든 기본 블럭에 복사하고 해당 명령을 제거한다.

단계 3. Bd와 Bi사이에 Bi와 데이타 종속 관계가 있는 명령어가 있다면 종속관계가 있는 명령어의 destination 기호 변수를 임시 기호 변수로 대치 시켜 그 명령을 Bd위로 이동시키고 Bi의 기호변수를 임시 기호 변수로 대치 시킨다.

단계 4. Bi로 부터 분기되는 경로상에서 단계 3에서 destination 기호 변수를 사용하는 명령 직전에 임시 기호 변수를 destination 기호 변수로 이동 시키는 명령어를 삽입한다

```
label_0: load value(ptr),v
         cmp v,x,t1
         bre t1,label_1
         brp t1,label_2
         load lchild(ptr),ptr
         cmp ptr,null,t2
         bre t2,label_3
         br label_0
label_1: rtn ptr
label_2: load rchild(ptr),ptr
         cmp ptr,null,t3
         bre t3,label_3
         br label_0
label_3: rtn null
```

그림 1. 탐색 프로그램에 대한 기호 코드

Fig. 1. symbolic code for a binary search program.

단계 2에서 종속관계가 없는 명령을 Bi의 이후 모든 경로에 복사 시키는 것은 프로그램의 의미 보존을 위한 것이고, 단계 3에서 임시 기호 변수로 대치된 명령어는 다음 명령어들에 어떤 영향도 주지 않기 때문에 Bd위로 이동시키므로써 병렬성을 증가 시킬수 있다. 이러한 변환 과정을 이진 탐색 트리에서 임의의 요소를 찾는 탐색 프로그램에 대해 적용시켜보자.

그림 1은 2진 트리에서 탐색 프로그램을 가상머신의 기호 코드로 나타낸 것이고, 그림2는 탐색 프로그램의 기호 코드에 대한 흐름도이다.

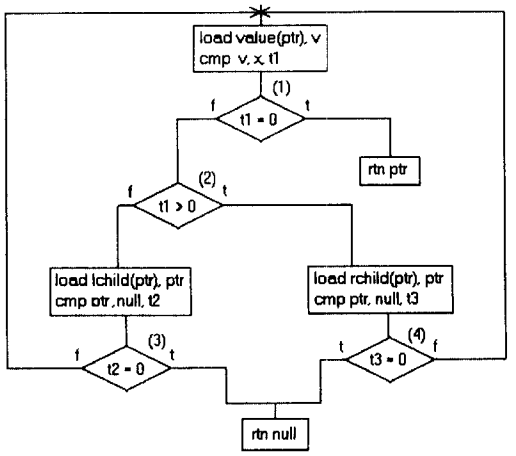

그림 2. 2진 탐색 프로그램에 대한 흐름도
Fig. 2. Flow diagram for the binary search program.

그림 2에서 분기 명령(1)과 (3)사이의 destination 오퍼랜드 ptr은 절차 3에 의해 임시 변수 temp-lchild로 대치시켜 분기 명령 (1)위로 이동시키고, 절차 4에 의해 MOVE 명령을 삽입한다. 분기 명령 (2)와 (4)사이의 destination 오퍼랜드 ptr에 대해서도 절차 3과 4를 적용시켜 변환시킨 결과는 그림 3과 같다.

그림 3에서 load 명령들과 cmp 명령들은 하드웨어 리소스(resources)가 충분하다면 동시에 수행 시킬수 있으므로 명령어 수준 병렬성이 증가되는 효과를 얻을수 있으며, 4개의 분기 명령들을 1개의 다중 분기 명령으로 수행시키기 때문에 분기지연과 branch penalty를 감소 시킬수 있다.

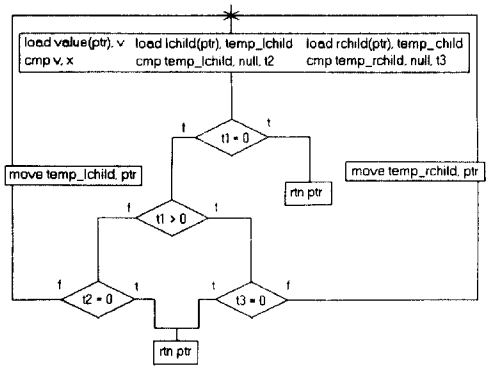

그림 3. 변환후의 2진 탐색 프로그램의 흐름도
Fig. 3. Flow diagram of the binary search program after the transformation.

Ⅲ. 다중 분기 명령의 구현

이 장에서는 다중분기 명령을 효과적으로 처리할 수 있는 하드웨어 메카니즘에 대해 기술한다. 즉, 한 사이클내에 다수의 조건 코드를 조사하여 다음에 수행될 명령어를 선택하는 다중 분기 메카니즘에 대해 알아보고, 다중분기 구현 하드웨어를 간단하게 하기 위해 흐름 그래프상의 연속적인 분기 명령들에 의한 분기 트리를 기준 분기 트리로 변환시키는 방법을 서술한다.

1. 머신 모델

Fisher는 다중분기 명령의 다음 명령어의 주소를 선택하기 위해 n개의 조건 코드가 있을때 2^n개의 가능한 다음 명령들을 2^n개의 메모리 뱅크의 동일 주소에 위치시켜, n개의 조건 코드에 의해 그 중 한개의 뱅크를 enable시켜 명령어를 인출하는 방법을 제안하였다.[8] 그러나 조건 코드에 의한 분기명령들을 트리 형태로 표현시키면 실제 n개의 조건 코드에 대해

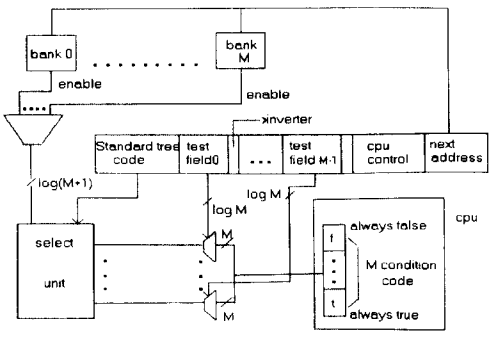

그림 4. 다중 분기 구현 하드웨어에 대한 블럭도
Fig. 4. Block diagram of a multi-way branch hardware implemented.

가능한 다음 명령어는 n+1개로 줄일 수 있으므로 메모리 뱅크수를 대폭 감소시킬수 있다. 또한 n개의 조건 코드에 대한 서로 다른 분기트리 수는 n가 증가함에 따라 지수함수적으로 증가하므로 모든 분기트리에 대해 다음 명령어를 선택하도록 하기위해서는 많은 하드웨어가 필요하기 때문에 3절 에서 제안한 변환에 의해 서로다른 분기트리수를 감소시켜, 다음 명령어를 선택하는 select unit을 간단하게 구현시킬수 있다. 다중분기를 처리할수 있는 하드웨어에 대한 블럭도 는 그림 4와 같다.

그림 4에서 모든 뱅크의 출력은 함께 연결되어 prefetch상태에 있고, select unit은 기준분기코드와 각 테스트 필드에 의해 대응하는 조건코드를 입력받아 조건을 만족하는 enable 신호가 명령어 사이클초기에 생성되어 다음 명령어가 지연없이 인출될수 있게 한다.

2. 분기 트리

분기명령들은 DAG로 구성하고 그러한 DAG를 분기 트리로 변환한다. 이때 2개의 분기 명령의 타겟이 되는 분기 명령에 대한 노드는 분할시켜 트리형태로 변환한다. 동일 노드수를 갖는 많은 분기트리는 조건문을 변형시키면 동일한 트리형태로 변환될 수 있다. 예를 들면 3개의 분기 명령에 대한 분기 트리는 그림 5와 같이 5개가 가능한데 그림 5의 (b),(d),(e)는 각 노드의 조건문을 반전시키면 (a)로 변환될수 있기 때문에 2개의 분기 트리(a),(c)로 나타낼수 있다.

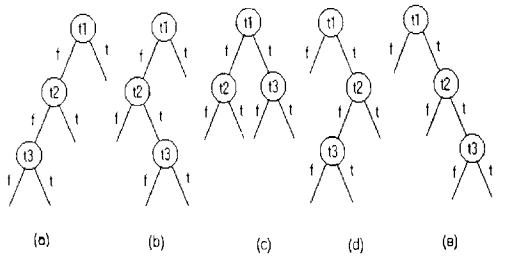

그림 5. 3개의 분기 명령에 대한 가능한 분기트리
Fig. 5. Possible branch trees with three branches.

표 1은 분기 명령수에 따른 가능한 분기 트리수와 변환에 의한 서로 다른 분기 트리의 수를 나타내고 있다.

따라서 select unit의 입력된 분기트리 코드는 기준 분기트리에 대한 인코딩 코드만을 입력시키면 되므로 select unit에 대한 하드웨어를 간단화 시킬수 있다.

표 1. 분기 명령수에 대한 가능한 분기트리와 기준 분기 트리수
Table 1. The number of branch trees and the number of standard branch trees for each number of branches.

분기명령수	가능한 분기 트리수	서로 다른 분기 트리(기준 분기 트리)수
1	1	1
2	2	1
3	5	2
4	14	3
5	42	6
6	132	11
7	429	23

3. 기준 분기 트리로의 변환

기준 분기 트리는 n개의 분기 명령에 의해 구성될 수 있는 분기 트리중 임의의 노드에 대해 좌서브트리의 노드수가 우서브트리의 노드수 보다 같거나 많은 성질을 만족하는 분기 트리를 나타내는데 예를들면 그림 5에서 (a),(c)가 3개의 분기 명령에 대한 기준 분기 트리가 된다. 만약 어떤 프로그램에 대한 분기 트리가 기준 분기 트리와 같지 않다면 기준 분기 트리가 되도록 임의의 노드의 좌서브트리의 노드수가 우서브트리의 노드수보다 작다면 좌 서브트리와 우서브트리를 교환시킨다.이 때 노드의 태그 비트가 기준 분기 트리와 다르다면 노드의 테스트 비트를 반전시켜 기준 분기 트리와 태그 비트가 같도록 한다. 그림 6은 변환의 예로서 그림 4의 분기 트리 (e)를 기준 분기 트리 (a)로 변환하는 과정을 나타낸다.

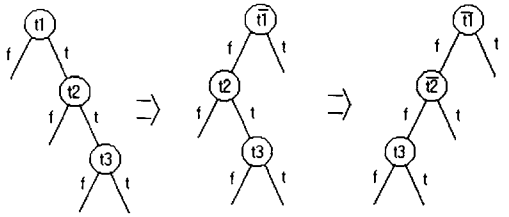

그림 6. 기준 분기 트리로의 변환 과정
Fig. 6. Steps to transform a branch tree into the corresponding standard branch tree

Ⅱ장의 예인 2진 트리에서의 탐색 프로그램에 대한 분기 트리를 기준 분기 트리로 변환한 결과를 그림 7에 나타내었다.

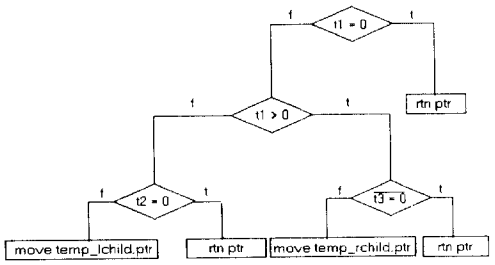

그림 7. 2진 트리의 탐색에 대한 분기트리를 기준 분기 트리로 변환시킨 결과

Fig. 7. Result to transform a branch tree into the corresponding standard branch tree in the binary search program.

Ⅳ. 명령어 배치

다중 분기문의 명령어 구조는 M개의 분기 명령을 가질때 다음에 수행될 M+1개의 명령어를 동일 타겟 주소를 갖는 M+1개의 메모리 뱅크에 효과적으로 배치 시킬수 있다. 그러나 대부분의 경우 M보다 작은 분기 명령을 갖기 때문에 메모리를 비 효율적으로 사용하게 된다. VLIW 구조 컴퓨터의 명령어의 길이는 매우 길기 때문에 그러한 비 효율적인 명령어 배치는 많은 메모리 낭비를 초래하게 된다. 만약 다중 분기문의 분기명령이 M보다 클 경우는 M개로 구성되는 다중 분기문과 나머지로 구성되는 다중 분기문으로 분할시킨다. 이장에서는 명령어들을 메모리 뱅크에 효과적으로 배치 시키므로써 메모리 낭비를 없애는 명령어 배치 방법에 대해 알아 본다. 명령어에 분기 명령수가 M보다 작은 N개를 갖는다면 그 명령어는 M개의 분기 명령를 갖는 기준 트리의 부 트리(subtree)가 된다. 이러한 성질을 이용하여 분기 명령이 M이하인 명령어들을 더미 노드(dummy node)를 첨가하여 분기 명령이 M이고 M+1개의 타겟 명령어를 갖는 기준 트리로 변환하는 과정은 다음과 같다.

단계 1. 모든 명령들을 분기 명령을 노드로 하고 제어 관계를 에지로 한 트리로 구성하고 트리의 좌서브트리의 노드수가 우서브트리의 노드 수보다 크거나 같게 되도록 분기 명령의 조건문을 반전시켜 트리를 재구성한다.

단계 2. 노드수가 M보다 큰 트리는 분할하여 모든 트리의 노드수가 M이하인 트리로 만들고 트리들을 노드수에 따라 정렬한다

단계 3. 노드수가 M이하의 트리에 대해 트리의 노드합이 M-1이 되는 트리 쌍을 선택하여 루트에 더미 노드를 첨가하여 노드가 M인 트리로 머지한다. 이때 노드가 많은 트리는 더미노드의 좌서브트리로 적은 트리는 우서브트리가 되게 하고 태그 비트는 각각 "f"와 "t"로 세트시킨다.

단계 4. 나머지 트리 중에서 2개의 트리의 노드합이 M-3 이하가 되는 트리쌍을 단계 3과 같은 과정을 수행 머지 시킨다.

단계 5. 마지막 1개의 트리를 제외하고 모든 트리의 노드수가 M이 될때까지 단계 3,4를 반복한다.

단계 6. 노드합이 M인 모든 트리에 대한 명령어 생성시 고유의 기준 트리의 코드값을 부여하고, 더미 노드를 갖는 트리에 대해서는 더미 노드에 대응하는 테스트 필드는 태그 비트가 "f"일 경우는 조건코드 "f"가 선택되게, "t"일 경우는 조건코드가 "t"가 선택되도록 하며, 조건문이 반전된 경우는 인버터 필드를 세트시킨 다음 시작 명령어를 갖는 트리의 주소를 시작으로 하여 각 트리에 대해 메모리 뱅크 주소를 할당한다.

Ⅴ. 결론

본 논문에서는 multi-way branch를 적용할수 없는 프로그램 세그먼트를 적용 가능하도록 하는 변환 방법을 제안 하였다. 이러한 변환은 많은 분기 명령에 의한 지연 스롯을 제거할뿐 아니라 제어종속관계에 의해 병렬 수행할 수 없던 명령어를 분기 명령위로 이동시킴으로써 병렬성을 확장할수 있었다. 또한 다중 분기 명령에서 다음에 수행될 명령어를 한 명령어 사이클내에 fetch하게 하는 하드웨어 메카니즘을 제안하였다. 일반적인 프로그램의 경우 조건 분기 명령이 전체 명령어의 15-30%를 차지하는데 다수의 조건 분기 명령들을 한개의 다중 분기 명령으로 결합시키므로써 분기문에 의한 지연을 감소시켜 프로그램의 수행속도에 상당한 개선을 얻을수 있다.

參 考 文 獻

[1] F.Gasperoni,"Compilation techniques for VLIW architectures",Research Report RC 14915,IBM Research Division,T.J. Watson Research Center, Sep. 1989.

[2] J. Fisher,"VLIW architecture and the ELI-512", In proceedings of the 10th

Annual Symposium on Computer Architecture, pp 140-150, 1983.
[3] J. Fisher, "The VLIW machine: a multiprocessor for compiling scientific code", *IEEE Computer*, pp 45-53, Jul. 1984.
[4] J. Ellis, "Bulldog: A compiler for VLIW architecture", PhD thesis, Yale Univ. 1985.
[5] K. Karplus and A. Nicolau, "Efficient hardware for multi-way jumps and prefetches", In Proceedings of the 18th Annual Microprogramming Workshop, pp 11-18, 1985.
[6] S. Moon and A. K. Agrawala, "Hardware implementation of a general multi-way jump mechanism", In Proceedings of the 23th Annual Workshop on Microprogramming and Microarchitecture, pp 38-45, Nov. 1990.
[7] Yen-Jen Oyang, "Exploiting multi-way branching to boost superscalar processor performance", ACM SIGPLAN vol.26, no.3, Mar. 1991.
[8] J. Fisher, "2n-way jump microinstruction hardware and an effective instruction binding method", In Proceedings of the 13th Annual Micro-programming Workshop, pp 64-75, Nov. 1980.
[9] R. Colwell et al., "A VLIW architecture for a trace scheduling compiler", *IEEE Trans. computers*, 37(8) pp 967-979, Aug. 1988.
[10] K. Ebcioglu, "Some design ideas for a VLIW architecture for sequential natured software", Proceedings of IFIP Conference on Parallel Processing, pp 3-17, Apr. 1988.
[11] McFarling and John Hennessy, "Reducing the cost of branches", Proc. 14th Symposium computer Architecture, pp 396-403, Jun. 1986.
[12] Smith A. and J. LEE, "Branch prediction strategies and branch target buffer design", Computer 17:1, pp 6-22, Jan. 1984.

著者紹介

咸 濼 淑(正會員)
1967年 1月 29日生. 1989年 2月 수원대학교 전자계산학과 졸업(이학사). 1992年~현재 수원대학교 전자계산학과 석사과정. 주관심 분야는 컴파일러, branch prediction, 병렬처리 시스템 등임.

趙 鍾 顯(正會員)
1962年 4月 4日生. 1986年 2月 한양대학교 전자공학과 졸업(공학사). 1992年~현재 수원대학교 전자계산학과 석사과정. 주관심 분야는 컴파일러, static 및 dynamic instruction scheduling, 컴퓨터구조 등임.

趙 榮 一(正會員)
1957年 4月 28日生. 1980年 2月 한양대학교 전자공학과 졸업(공학사). 1982年 2月 한양대학교 전자공학과 대학원 졸업(공학석사). 1985年 8月 한양대학교 전자공학과 공학박사 취득. 1986年 3月~현재 수원대학교 전자계산학과 근무. 부교수. 주관심 분야는 컴파일러, static 및 dynamic instruction scheduling, 컴퓨터구조 등임.