네트워크 플로우에 기반한 아키텍쳐 수준에서의 전력 최적화

여준기 김태환
한국과학기술원 전산학과
cglyuh@vlsisyn.kaist.ac.kr tkim@cs.kaist.ac.kr

Network-Flow Based Architecture-Level Power Optimization

Chun-Gi Lyuh Taewhan Kim
Dept. of EECS, Korea Advanced Institute of Science and Technology

요 약

이 논문은 행위 합성(behavioral synthesis)에서 전력 소모 최적화를 위한 효율적인 알고리즘을 제안한다. 이전의 논문에서 전력 최적화를 위한 여러가지의 하드웨어 할당/바인딩(allocation/binding) 문제들이 네트워크 플로우 문제로 나타내어 질 수 있고, 최적으로 풀릴 수 있음이 보여졌다. 그러나, 그 연구에서는 고정된 스케쥴이 가정되고 있다. 이와 관련해서 주어진 스케쥴에 대한 하드웨어 할당/바인딩 문제를 위한 최적의 네트워크 플로우 결과가 주어졌을 때, 주어진 스케쥴을 일부분 바꾼 것에 대한 새로운 최적의 네트워크 플로우 결과를 얻는 것이 주요 문제이다. 이 때문에 우리는 네트워크 구조와 플로우 계산간의 관계에 대한 분석으로 부터 *최대 플로우 계산 단계*와 *최소 비용 계산 단계*의 2 단계 과정을 고안하였다. 실험결과를 통해 우리의 설계가 스케쥴의 영향과 각 스케쥴에 대한 최적의 바인딩을 이용함으로써 전력 소모와 계산 시간에서 매우 향상된 결과를 얻을 수 있음을 볼 수 있다.

1 서론

랩탑 컴퓨터와 무선통신기기와 같은 이동이 가능하고 고밀도 마이크로 전자장치의 발전과 함께 VLSI 회로의 전력 소모는 결정적인 관심사가 되었다. 전지의 수명, 패키징/냉각 비용, 신뢰도 등은 다양한 응용에서 성능과 면적보다 전력 소모가 설계에서 더 결정적인 관심사가 되도록 했다. 이렇게 전력 모델링, 계산, 최적화가 시스템과 행위 수준(behavioral level)에서부터 게이트, 레이아웃 수준까지의 모든 설계 단계에 수행되어야 한다.

행위 합성은 주어진 설계 제약 조건하에서 디지탈 시스템의 행위 명세를 동일한 기능의 레지스터 전달 수준(register-transfer level)의 기술로 바꾸는 자동화된 방법을 제공한다. 행위 합성에서 주요 단계는 오퍼레이션 스케쥴링(scheduling)과 하드웨어 할당(allocation), 바인딩(binding)이다. 대부분의 행위 합성 시스템은 오퍼레이션의 스케쥴링이 오퍼레이션의 할당과 바인딩을 위한 각 주기마다의 타이밍 정보를 제공하기 때문에 할당과 바인딩 이전에 스케쥴링을 수행한다. 스케쥴링과 저전력을 위한 하드웨어 할당 및 바인딩에 관해 여러 논문들이 발표되었다. [1, 2, 3, 4, 5] 논문들은 할당/바인딩 단계에서 전력 최소화의 결과에 스케쥴링이 큰 비중을 차지함을 보여 주었다. 이는 스케쥴링, 할당, 바인딩 각각의 영향을 전부 이용하기 위해서는 이들이 결합된 형태로 수행되어야 함을 강조하고 있다.

반면에 스케쥴 결과가 주어졌을 때 저전력을 위해 하드웨어를 할당/바인딩하는 알고리즘들이 있다. Chang과 Pedram [6]은 스위칭 활동을 최소화하기 위한 레지스터 할당과 바인딩 방법을 제안하였다. 그들[7]은 또한 기능 모듈에서의 스위칭 활동을 최소화 하기 위한 바인딩 방법을 제안하였다. 비록 [6, 7]에서의 시도는 많은 특정 저전력 설계 문제들에 대해 최적의 결과를 구하여 준다 할 지라도 스케쥴링 결과의 변화에 관련한 최적의 결과를 효율적으로 찾는 문제에 대해서는 다루지 않았다. 이는 우리의 네트워크 플로우 방법에 기반을 둔 새로운 최적화 방법의 개발에 동기가 되었다.

본 논문에서 제안한 알고리즘은 [4, 5]에서와 같이 재스케쥴링에 의해 이전의 바인딩 결과를 반복적으로 향상시킨다. 여기에서 주요 논점은 어떻게 새 스케쥴에 대한 바인딩 결과를 빠르고 정확하게 구할 수 있느냐 하는 것이다. 우리는 이 문제를 네트워크 플로우 계산의 포괄적인 2 단계 방법을 고안함으로써 해결하였다. 첫 단계는 이전 단계에서 얻은 플로우를 가능한 많이 유지하면서 최대 플로우 결과를 찾는 최대 플로우 계산 단계이고, 두 번째 단계는 최소 비용을 가지도록 반복적으로 갱신하는 최소 비용 계산 단계이다.

이 논문은 다음과 같이 구성되어 있다. 2장에서는 저전력을 위한 버스 최적화 문제와 네트워크 플로우 공식화를 정의한다. 3장에서는 제안하는 알고리즘의 개요와 스케쥴의 변화에 대한 플로우 계산의 핵심 방법을 설명한다. 4장에서는 벤치마크 설계에 대한 실험을 다루고, 마지막으로 5장에서는 결론을 내린다.

2 저전력을 위한 버스 최적화

2.1 문제 정의

CMOS 설계에서 버스에서의 전력 소모는 버스에서의 스위칭 활동에 비례한다.[8] 이 스위칭 활동은 버스의 비트 라인에서 신호의 천이(transition)를 나타내는 것이다. 결국, 버스에서의 신호 천이의 수를 줄이는 것이 전체 전력 소모를 줄이는 것과 같게 된다. 버스에서 각 비트 라인에서의 신호 스위칭 활동은 버스에서의 데이타 전송의 종류와 데이타 전송의 순서에 따라서 바뀐다. 스케쥴링은 각 클락 사이클에서 전송되는 데이타들을 결정한다. 그러나 각 데이타 전송이 어느 버스에서 일어날지에 대해서 결정하지는 않는다.

버스에서 데이터 전송 x와 y가 연속적으로 일어날 때 $SW(x,y)$를 평균 비트 천이 수, 즉 해밍 거리라고 하자. 스케쥴 되지 않은 데이타-흐름 그래프(data-flow graph)에서 각 데이타 전송 쌍에 대한 $SW(x,y)$ 값은 데이타-흐름 그래프를 반복하여 모의 실험 함으로써 얻을 수 있다. $SW(x,y)$ 값은 x와 y 사이의 해밍 거리의 평균 값을 가지게 된다.

$SW^k(x,y)$를 데이타 전송 x와 y가 연속적으로 버스 k에서 일어났을 때 천이 수의 기대값이라고 하자. 그리고 SW^k를 버스 k에서 연속적인 데이타 전송의 모든 쌍에 대한 $SW^k(\cdot)$ 값의 합이라고 하자. 그러면, 우리가 풀고자 하는 문제는 각 오퍼레이션을 스케쥴함으로써 데이타 전송을 스케쥴하고 데이타 전송을 버스에 바인딩함으로써 다음 수식의 값을 최소화하는 것이다.

$$SW_{tot} = \sum_{\forall k \ of \ buses} SW^k \quad (1)$$

주어진 스케줄링 결과에 대해 버스에서의 데이타 전송은 여러가지 방법으로 바인딩 될 수 있다. 그리고, 스케줄링은 바인딩의 결과와 버스에서의 스위칭 활동에 상당히 영향을 끼친다. 그러므로 저전력을 위한 버스 최적화를 위해서는 버스 바인딩 뿐만 아니라 스케줄링 또한 고려되어야 한다.

2.2 네트워크 플로우 공식: 개요

주어진 스케줄로부터 데이타 전송의 최적 바인딩을 찾는 문제는 네트워크에서 최소 비용을 가지는 최대 플로우를 찾는 문제로 공식화 할 수 있다.

클락 스텝 i에서의 데이타 전송 집합을 $DT(i)$라고 하자. 네트워크 $G = (N, A)$는 노드 집합 N과 아크 집합 A를 가지는 방향있는 그래프이다. 전체 n 개의 데이타 전송에 대해서, N은 각 데이타 전송에 대해 두 개씩 전체 $2n$ 개의 노드를 가지며, 네트워크에서 소스(s)와 싱크(r)의 두 노드를 추가적으로 가지게 된다. 소스와 싱크를 제외한 다른 노드들은 데이타 전송이 수행되는 클락 스텝에 따라서 수직으로 나열된다. 그림 1(a)의 네트워크는 G의 구조의 한 예를 보여 준다.

데이타 전송에서 *외부 천이*(데이타-흐름 그래프들 간에서의 천이)를 고려하면 그림 1(a)의 네트워크는 그림 1(b)에서 보여지는 것 처럼 추가적인 왼쪽의 노드 열을 포함하도록 확장된다.

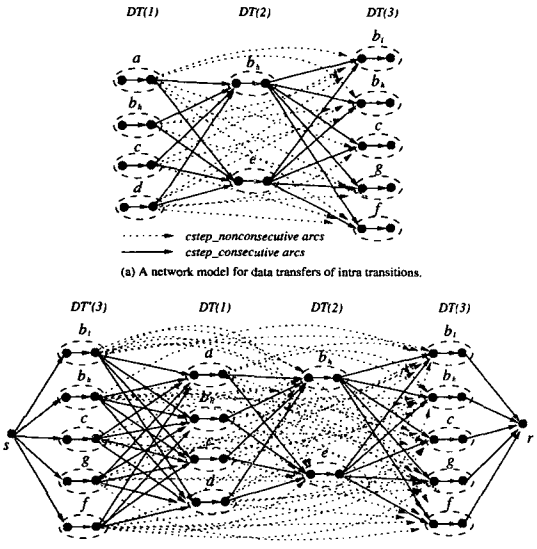

그림 1: 스케줄링된 데이타 전송 바인딩에 대한 네트워크 플로우 모델의 예.

아크 집합 A의 각 아크의 용량은 1이다. 우리는 각 아크에 비용을 할당하여서 G에서 최소 비용의 최대 플로우 결과가 다음의 조건들을 만족하도록 해야 한다. (*조건 1*) 플로우는 G의 모든 노드를 포함하고 흘러야 한다. (조건 2) *조건 1*을 만족하는 모든 가능한 플로우 결과 중에서 플로우에 속하는 아크에 대한 $SW(\cdot)$ 값의 합이 최소이어야 한다.

$SW(\cdot)$는 두 데이타 전송 사이의 비트 천이의 평균 숫자이므로, $SW(x,y) + SW(y,z) \geq SW(x,y)$가 항상 만족되지는 않는다. 그래서 우리는 각 아크의 비용을 다음과 같이 공식화하였다. SW^{max}와 SW^{min}을 각각 모든 $SW(\cdot)$ 중에서의 최대값과 최소값이라고 할 때, 열 $t1$의 데이타 전송에 해당하는 노드 x에서 열 $t2$의 데이타 전송에 해당하는 노드 y로의 아크에 주어지는 비용 $C(x,y)$는 다음과 같다.

$$C(x,y) = SW(x,y) - (2 \cdot SW^{max} - SW^{min}) \quad (2)$$

3 스케줄링과 버스 바인딩의 통합 알고리즘

버스 바인딩을 위한 우리의 알고리즘은 반복적으로 수행된다. 주어진 초기 스케줄링과 바인딩 결과로부터 재스케줄링과 재바인딩을 반복적으로 수행함으로써 바인딩 결과를 향상시킨다. 각 스케줄에 대한 최적의 바인딩이 결정되는데, 이 때 주요 논점은 어떻게 최적의 바인딩을 빠르게 생성해 내느냐 하는 것이다. 우리의 알고리즘의 전체 흐름은 그림 2로 요약된다. 이에 대한 상세한 설명은 지면 관계상 생략하도록 하겠다.

Flow_LP: 저전력을 위한 네트워크 플로우 기반 버스 합성
(CDFG, cstep_limit, resource_limit) {
- 데이타 흐름 그래프를 모의 실행 하여 $SW(\cdot)$ 표를 만든다. /* 2.1절 */
- 임의의 스케줄링 알고리즘을 이용하여 데이타-흐름 그래프에 대한 초기 스케줄을 만든다.
- 최소 비용 증가 방법을 이용하여 초기 스케줄에 대한 버스 바인딩을 얻고, 플로우 충돌이 있을 경우 이를 해결한다. /* 3.3 절 */
- $COST_{min}$ 값을 초기 바인딩의 SW^{opt}_{tot} 값으로 둔다.
- $BIND_{best}$ 을 초기 스케줄과 바인딩으로 둔다.

while ($BIND_{best}$가 수정되었으면)
- $COST_{current}$을 $COST_{min}$으로 둔다.
- $BIND_{current}$을 $BIND_{best}$으로 둔다.
- **while** (재스케줄링이 가능한 오퍼레이션이 있을 경우) {
  - /* 클락 스텝 제한과 자원 제한에 의해 제약된다 */
  - **foreach** 클락 스텝 j로의 재스케줄 가능한 오퍼레이션들에 대해
    - 재스케줄링을 하고 네트워크 G를 수정한다.
    - G의 플로우를 수행가능한 결과가 되도록 최소한으로 고친다. /* 3.2 절 */
    - G의 플로우가 최소 비용 플로우가 되도록 고친다. /* 3.2 절 */
    - 충돌되는 플로우가 있으면 이를 해결한다. /* 3.3 절 */
    - 이 스케줄링에 대해 SW^{opt}_{tot}을 계산하고 스케줄을 원상대로 돌린다.
  - **endforeach**
  - 최소 SW^{opt}_{tot} 값을 가지는 오퍼레이션에 대해서 재스케줄 한다.
  - **if** (바뀌어진 SW^{opt}_{tot} 값이 $COST_{min}$ 보다 작을 경우) {
    - $COST_{min}$ 값을 현재 SW^{opt}_{tot} 값으로 바꾸고, $BIND_{best}$도 수정한다.
  - **endif**
  - 그 오퍼레이션을 해당 클락 스텝에 고정시킨다.
- **endwhile**
- 모든 오퍼레이션에 대해 고정시킨 것을 해제시킨다.

endwhile
- $BIND_{best}$와 $COST_{min}$ 결과를 돌려준다.

그림 2: 스케줄링과 결합된 버스 바인딩을 위해 제안된 알고리즘.

우리 알고리즘의 핵심은 하나의 오퍼레이션이 클락 스텝 i에서 j로 재스케줄 될 때 최적의 SW_{tot} 값을 효율적으로 계산하는 것이다. 이것을 위해 우리는 다음과 같은 두 단계로 이루어진 포괄적인 네트워크 플로우 계산 방법을 고안하였다.

단계 1 (*최대 플로우 계산 단계*) : 이 단계에서는 이전 스케줄에 대한 플로우 경로를 현재 스케줄에서 유효한 플로우가 되도록 고친다. 우리는 이것을 이전의 네트워크 플로우에서 수정되어야 하는 영역을 찾아내어 그 영역에 대해서만 최적의 플로우를 만들어냄으로 해결하였다.

이러한 플로우 수정 아이디어를 예를 통해서 설명하고자 한다. 그림 3(a)는 이전 스케줄에 대한 플로우 결과의 한 부분을 보여준다. 여기에서 클락 스텝 $i+1$에서 수행되었던 데이타 전송 $y1, y2, y3$가 재스케줄에 의해 클락 스텝 i에서 수행되도록 바뀌었다고 가정하자. 그림 3(b)는 이전 플로우 결과에서 바뀌어진 네트워크를 보여준다. 결과적으로, 우리는 그림에서 굵은 선으로 보여진 플로우 아크 $x1 \rightarrow y3$와 $x2 \rightarrow y2$가 유효하지 않은 플로우가 되었음을 알 수 있다.

이렇게 유효하지 않게 된 플로우로부터 부분 네트워크를 추출하고, 이렇게 추출한 부분 네트워크에서 최대 플로우를 찾은 후에, 이에 따라 전체 네트워크를 수정해 줌으로써 유효한 최대 플로우로 만들어

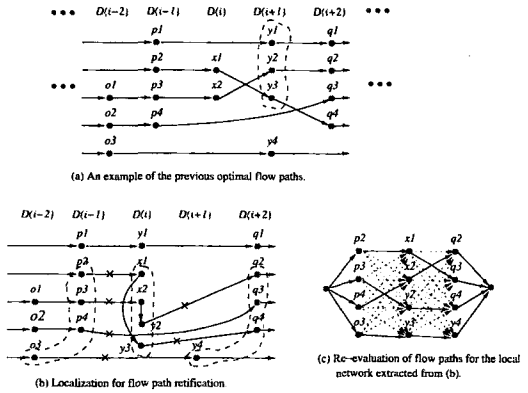

그림 3: 단계 1의 방법을 보여주는 예.

주게 된다. 그림 3(c)는 그림 3(b)로 부터 추출된 부분 네트워크와 그 플로우 결과를 보여준다.

단계 2 (*최소 비용 계산 단계*) : 단계 1에서 구해진 플로우는 최대 플로우이다. 그러나 대부분의 경우에 이 플로우가 최적의 비용 플로우에 매우 가깝다고 하여도 최적의 비용 플로우는 아니다. 따라서, 단계 1에서 구해진 플로우를 최소 비용의 최대 플로우로 빠르게 고치는 것이 다음의 문제가 된다. 그리고, 이 문제는 다음의 *비용 감소 방법*을 통해 해결할 수 있다. 즉, 최대 플로우에서 시작하여 residual 그래프에서 음의 비용 사이클을 따라 가능한한 많은 플로우를 흘려보내고 residual 그래프에서 더이상 음의 사이클이 없을 때 까지 반복하는 알고리즘을 사용하여 이 문제를 해결한다.

```
/* 하나의 오퍼레이션이 클락 스텝 j에서 i로(i = j + 1 or j − 1)
재스케줄 되었을 때 최적의 재바인딩*/
(단계 1) 최대 흐름 계산: /* 재스케줄 위해 G를 유효한 최대 흐름이 되도록 만듦. */
• 그래프 G에서 클락 단계 i와 j사이의 무효화된 플로우 아크의 집합 S를 찾는다.;
• SRC_i(S)와 DEST_i(S)를 찾는다.;
• 그래프 G에서 클락 단계 i를 건너 뛰는 플로우 아크의 집합 Q를 찾는다.;
• SRC_i(Q)와 DEST_i(Q)를 찾는다.;
• G로 부터 세 개의 노드 열, SRC_i(S) ∪ SRC_i(Q), S의 터미널 노드,
  DEST_i(S) ∪ DEST_i(Q)로 구성된 G_partial을 얻는다.;
• G_partial에 최소 비용 증가 방법을 수행한다.;
(단계 2) 최소 비용 계산: /* G의 플로우 비용을 최소화 한다. */ {
• 단계 1에서 얻어진 Gdml 최대 흐름에 대한 residual 그래프 R을 만든다.;
• R에서 음의 비용 사이클을 찾는다.;
while (음의 비용 사이클이 있을 경우)
  • 플로우 비용을 줄이기 위해 G의 플로우를 수정하고, R을 수정한다.;
  • R에서 음의 비용 사이클을 찾는다.;
endwhile
```

그림 4: 재스케줄에 대한 최적 플로우 계산의 2단계 알고리즘.

4 실험 결과

알고리즘 *Flow_LP*는 C++로 구현되었고, Sun Sparc20 웍스테이션에서 수행되었다. 우리는 실험에서 여러가지의 상위 수준 합성 벤치마크 설계들을 가지고 실험하였다. 실험은 두 가지 관점에서 수행되었다. 하나는 전체 스위칭 천이 수(즉, 식 (1)에서의 SW_{tot}의 양)의 면에서의 결과가 얼마나 좋은 가를 살펴보았고, 다른 하나는 알고리즘의 수행 속도를 살펴보았다.

표 1은 [5]에서 제안된 *임의 이동*에 기초한 방법으로 만들어진 설계와 우리의 최적 알고리즘 *Flow_LP*에 의해 만들어진 설계에 대해서 식 (1)의 SW_{tot}의 관점에서 측정된 버스 스위칭 활동의 비교를 보여준다. 표를 통해서 우리의 알고리즘이 *임의 이동*에 비해서 17.4%만큼 버스 스위칭 활동을 감소 시켰음을 알 수 있다.

	전체 천이 수		
설계	*임의 이동*[5]	*Flow_LP*	[5]에 대한 감소(%)
DIFF	22.48	15.45	31.3
DIFF.2	37.44	32.11	14.2
IDCT	73.82	67.07	15.9
IDCT.2	130.02	105.85	18.6
KALMAN	20.19	18.11	10.3
KALMAN.2	35.18	31.92	9.2
EWF	16.32	13.05	20.0
COMPLX	9.76	7.88	19.3
평균			17.4

표 1: 상위 수준 합성 표준 설계에 대한 버스 스위칭 활동의 결과.

표 2는 바인딩을 위한 경로 증가 방법(path augmentation method)의 전체 수행을 이용한 알고리즘(*old_flow*)와 *Flow_LP*와의 수행 시간 비교를 보여준다. 이 표를 통해서 우리의 최적 바인딩 방법이 최적의 *old_flow*보다 2.8배 빠름을 알 수 있다.

	수행 시간		비율
설계	*old_flow*	*Flow_LP*	*old_flow*/*Flow_LP*
DIFF	362 sec	78 sec	4.36
DIFF.2	451 sec	192 sec	2.35
IDCT	3.0 hr	0.5 hr	6.00
IDCT.2	4.2 hr	1.7 hr	2.47
KALMAN	28 sec	19 sec	1.47
KALMAN.2	62 sec	38 sec	1.63
EWF	1.6 hr	0.6 hr	2.67
COMPLX	46 sec	30 sec	1.53
평균			2.81

표 2: 각 설계에 대한 수행 시간 결과.

5 결론

우리는 [4, 5, 6, 7]의 이전 방법들의 제약점을 극복하기 위해 상위 수준 전력 최적화를 위한 포괄적인 네트워크 플로우 알고리즘을 보였다. 본 논문에서 제안하는 알고리즘은 (a) *스케쥴링 효과를 이용*하고, (b) *매 스케쥴 결과에 대해 최적의 바인딩*을 얻음으로써 이제까지의 알고리즘 보다 전력면에서 17.4% 효율적인 결과를 얻을 수 있었고, 중복된 플로우 계산을 최소화 하기 위해 (c) *이전 플로우 결과를 최대로 이용하는 2 단계 방법*으로 수행시간 면에서 이전의 네트워크 플로우 기반 최적 버스 합성 알고리즘 보다 2.8배 빠르게 결과를 얻을 수 있었다.

참조 서적

[1] E. Musoll and J. Cortadella, "Scheduling and Resource Binding for Low Power," *International Symposium on System Synthesis*, pp. 104-109, 1995.
[2] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar, "Scheduling Technique to Enable Power Management," *Design Automation Conference*, pp. 349-352, 1996.
[3] A. Raghunathan and N. K. Jha, "SCALP: An Iterative Improvement Based Low-Power Data Path Synthesis System," *IEEE Transactions on Computer-Aided Design of Intergated Circuits and Systems*, Vol. 16, No. 11, pp. 1260-1277, 1997.
[4] A. Dasgupta and R. Karri, "Simultaneous Scheduling and Binding for Power Minimization During Microarchitecture Synthesis," *International Symposium on Low Power Electronics and Design*, pp. 69-74, 1995.
[5] A. Dasgupta and R. Karri, "High-Reliability, Low-Energy Microarchitecture Synthesis," *IEEE Transactions on Computer-Aided Design of Intergated Circuits and Systems*, Vol. 17, No. 12, pp. 1273-1280, 1998.
[6] J.-M. Chang and M. Pedram, "Register Allocation and Binding for Low Power," *Design Automation Conference*, pp. 29-35, 1995.
[7] J.-M. Chang and M. Pedram, "Module Assignment for Low Power," *European Design Automation Conference*, pp. 376-381, 1996.
[8] F. N. Najm, "Transition Density, A Stochastic Measure of Activity in Digital Circuits," *Design Automation Conference*, pp. 644-649, 1991.