Efficient Compilation of Complex Data-Dependent Code to Coarse-Grained Reconfigurable Architectures

Stephen Friedman, Robin Panda, Benjamin Ylvisaker
Scott Hauck, Carl Ebeling
University of Washington Mosaic Project
Overview

- Course-Grained Reconfigurable Architecture (CGRA)
  - What/Why
- The Problem
  - Supporting resource sharing
- Compiler Abstractions
  - Control Model and Distribution
- Mapping Flow Modifications
  - Scheduling
  - Placement
  - Routing
Coarse-Grained Reconfigurable Architecture

- Lots of ALUs
- Time-multiplexed
- Embedded memory
- Streaming I/O
Coarse-Grained Reconfigurable Arrays

ASIC (Efficiency)

FPGA (Flexibility)

Multi-core CPU (Programmability)
Accelerating DSP & Scientific Computing

- Large, repetitive, computation
- Want flexibility

- Digital signal processing & scientific computing
  - Video
  - Networking
  - Encryption
  - DNA sequencing
  - Protein Folding
  - ...

Exploiting Many Types of Parallelism

- Data level
- Loop level
- Instruction level

```cpp
for(c=0;c<3;c++){
    for(p=0;p<10;p++){
        a[c][p]=a[c][p-1]+b[p]-(e[p]+d[p]);
        x[c][p]=a[c][p]*norm[c];
    }
}
```
Exploiting Many Types of Parallelism

- **Data level**
  
  ```
  for(c=0;c<3;c++){
    for(p=0;p<10;p++){
      a[c][p]=a[c][p-1]+b[p]-(e[p]+d[p]);
      x[c][p]=a[c][p]*norm[c];
    }
  }
  ```

- **Loop level**

- **Instruction level**

```text
for(p=0;p<10;p++){
  a[0][p]=...
  x[0][p]=...
}
```

```text
for(p=0;p<10;p++){
  a[1][p]=...
  x[1][p]=...
}
```

```text
for(p=0;p<10;p++){
  a[2][p]=...
  x[2][p]=...
}
```
Exploiting Many Types of Parallelism

- Data level
- Loop level
- Instruction level

```cpp
for(c=0; c<3; c++){
    for(p=0; p<10; p++){
        a[c][p] = a[c][p-1] + b[p] - (e[p] + d[p]);
        x[c][p] = a[c][p]*norm[c];
    }
}
```

- c=0; p=1; a[0][1] = a[0][0]...
- x[0][1] = a[0][1]...

- c=0; p=2; a[0][2] = a[0][1]...
- x[0][2] = a[0][2]...

- c=0; p=3; a[0][3] = a[0][2]...
- x[0][3] = a[0][3]...
Exploiting Many Types of Parallelism

- Data level
- Loop level
- **Instruction level**

```cpp
for(c=0;c<3;c++){
    for(p=0;p<10;p++){
        a[c][p]=a[c][p-1]+b[p]-(e[p]+d[p]);
        x[c][p]=a[c][p]*norm[c];
    }
}
```
Overview

- Course-Grained Reconfigurable Architecture (CGRA)
  - What/Why
- The Problem
  - Supporting resource sharing
- Compiler Abstractions
  - Control Model and Distribution
- Mapping Flow Modifications
  - Scheduling
  - Placement
  - Routing
Control Flow in Software Pipelining

- If-conversion – Compute both sides and predicate the unneeded computation

```c
for(;;){
    b = c+d;      [uncond]
    if(i<5)       [uncond]
        a = a xor b;  [i<5]
    else
        a = a and b;  [i>=5]
}
```
Control Flow in Software Pipelining

- If-conversion – Compute both sides and predicate the unneeded computation

```c
for (;;) {
    b = c + d;  [uncond]
    if (i < 5) [uncond]
        a = a xor b;  [i < 5]
    else
        a = a and b;  [i >= 5]
}
```

- Waste of resources!!!
Control Flow in Software Pipelining

- If-conversion – Compute both sides and predicate the unneeded computation

```c
for(;;){
    b = c+d; [uncond]
    if(i<5) [uncond]
    a = a xor b; [i<5]
    else
    a = a and b; [i>=5]
}
```

- Waste of resources!!!!
- Should share resources between “compatible” operations and signals – mutually exclusive
Goals

- Map mutually exclusive operations and signals to the same resources
- Ensure that the necessary predicates are available to select the operation to perform and routes to use
- Remain as architecture agnostic as possible
Overview

• Course-Grained Reconfigurable Architecture (CGRA)
  - What/Why
• The Problem
  - Supporting resource sharing
• Compiler Abstractions
  - Control Model and Distribution
• Mapping Flow Modifications
  - Scheduling
  - Placement
  - Routing
Configuration Hardware

- Configuration word selected by modulo counter
- Only covers a sub-portion of the chip
- One configuration memory for multiple computation units
Configuration Hardware - Modification

- Sharing is limited by Combiner and Config. Memory Size
- Need to route Predicate Signals to Config HW
Combiner Options

- Mux-Based
  - Mux in predicates as high-order counter bits to select alternate configurations

- CAM-Based
  - Match against predicate and counter bits to select wordline
Tracking Distribution of Control

- Regions:
  - Each color represents the area controlled by the same set of predicate bits
  - Each region could use a different subset of the predicate bits available
Overview

- Course-Grained Reconfigurable Architecture (CGRA)
  - What/Why
- The Problem
  - Supporting resource sharing
- Compiler Abstractions
  - Control Model and Distribution
- Mapping Flow Modifications
  - Scheduling
  - Placement
  - Routing
Compiler Modifications

- **SPR – Our Starting Point** [Friedman et al. FPGA'09]
  - **Schedule** -- Iterative Modulo Scheduling
  - **Place** -- Simulated Annealing
  - **Route** -- PathFinder, QuickRoute

- **Predicate Awareness Additions**
  - Allow sharing of mutually exclusive operations and signals
  - Route predicate signals to the configuration hardware
  - Respect limitations (regions/config depth) of config hw
Predicate Awareness in VLIWs (Scheduling)

Figure 2: Data dependence graph for code segment

Figure 4: IMS kernel (a) versus PAMS kernel (b) schedule

Placement Aware of New Flex/Constraint

- Based on Simulated Annealing
  - Allow multi-placement
  - Track Op compatibility
  - Track Pred availability
Predicate Availability Eases Routing

- The flexibility routing depends on the availability of predicates in the region routed through.

With predicates available, routes can split and join.

Without predicates available, routes can share, but only if they take the same path.
Making Routes Agree

- Without predicates available, routes can share, but only if they take the same path -> negotiate.
- Extend prior work on sharing multiplexed signals on non-multiplexed interconnect.
Making Routes Agree

- Without predicates available, routes can share, but only if they take the same path -> negotiate.
- Extend prior work on sharing multiplexed signals on non-multiplexed interconnect.
- Agree at different times --> Agree under different run-time conditions.
So Much To Do, So Little Time

- Topics covered here just scratch the surface
- Come ask at the poster
  - Can sharing reduce performance, and what can you do about it?
  - What if the start of a route is in a block controlled by one predicate, and there are many ends controlled by different predicates?
  - Some predicates are subsets of others – can we leverage this?
Thanks!