Rapid: A Configurable Architecture for Compute-Intensive Applications

Carl Ebeling
Dept. of Computer Science and Engineering
University of Washington
Alternatives for High-Performance Systems

✧ **ASIC**
  ➢ Use application-specific architecture that matches the computation
  ➢ Large speedup from fine-grained parallel processing
  ➢ Smaller chip because hardware is tuned to one problem
  ➢ Lower power since no extra work is done
  ➢ Little or no flexibility: problem changes slightly, design a new chip
    ✦ No economy of scale
    ✦ Long design cycle

✧ **Digital Signal Processors**
  ➢ Optimized to signal processing operations
    ✦ Simple, streamlined processor architecture
    ✦ Cheaper, lower power than GP processors
  ➢ Very flexible: just change the program
  ➢ Lower performance: small scale parallelism
Motivation for Rapid

✦ Many applications require programmability
  ➤ Old standards evolve
  ➤ Multiple standards, protocols, technology
    ✦ Similar but different computation
    ✦ Reprogram for different context
  ➤ New algorithms give competitive advantage

✦ We need a “configurable ASIC”
  ➤ Application-specific architecture
  ➤ Reprogrammable
What is a Configurable ASIC?

✦ Like an ASIC: Architecture tuned to application
  ➔ High performance/low cost

✦ But configurable:
  ➔ Datapath *structure* can be rewired via static configuration
  ➔ Datapath *control* can be reprogrammed

✦ Rapid approach
  ➔ Domain-specific architecture model
    ‣ Reconfigurable Pipelined Datapaths
  ➔ Well-suited to many compute-intensive applications
Example: Programmable Downconverter

Reconfigurable Pipelined Datapaths
Example: Programmable Downconverter

Reconfigurable Pipelined Datapaths
Example: Programmable Downconverter
Generating a Rapid Array

Diagram:

1. Rapid-C Program
2. Rapid Compiler
3. Netlist
4. Rapid Array Synthesis
5. Rapid Array Netlist
6. Hardware Synthesis
7. Rapid Array

Rapid Architecture Model
Programmable Downconverter ASIC

Diagram showing the components:
- Analog
- NCO
- Embedded DSP
- Program
- Rapid Array
- Memory for programmable control
- Datapath Control Program
- Configuration

Reconfigurable Pipelined Datapaths
Reconfiguring the Rapid Array

Rapid-C Program → Rapid Compiler → Netlist → Map Place/Route → Configuration → Control Program

Configurable ASIC Description

Rapid Array

Reconfigurable Pipelined Datapaths
Overview of Using Rapid

Generating a Rapid Array

1. Rapid-C Programs
2. Rapid Compiler
3. Rapid Array Netlist
4. Hardware Synthesis
5. Rapid Array

Programming a Rapid Array

1. New Rapid-C Program
2. Rapid Compiler
3. Rapid Program Netlist
4. Map Place/Route
5. Configuration
6. Control Program
7. Rapid Array
How Configurable is Rapid?

✦ Experimental Rapid Array:
  ✦ 16-bit data, configurable long addition
  ✦ 16 Multipliers, 48 ALUs, 48 Memories, Registers
  ✦ Extensive routing resources
  ✦ Configurable control logic
  ✦ 100 Mhz
  ✦ \(\sim 100 \text{ mm}^2\) in .6u technology
    ✦ Layout done for major components
How Configurable is Rapid?

✦ Applications mapped to Experimental Array
  ❯ FIR filters
    ✦ 16 tap, 100MHz sample rate
    ✦ 1024 tap, 1.5 MHz sample rate
    ✦ 16-bit multiply, 32-bit accumulate
    ✦ Decimating filters
  ❯ IIR filters
  ❯ Matrix multiply
    ✦ Unlimited size matrices
  ❯ 2-D DCT
  ❯ Motion Estimation
  ❯ 2-D Convolution
  ❯ FFT
  ❯ 3-D spline generation

✦ Performance:
  ❯ > 3 BOPS (data multiplies and adds)
Questions to be Answered

✧ How do you add configurability to an application-specific architecture?
  ➷ Use a domain-specific meta-architecture: Rapid
    ✧ Fine-grained parallelism
    ✧ Deep computational pipelines
    ✧ High performance

✧ How do you program a Rapid array?
  ➷ Use a programming model tuned to the meta-architecture
  ➷ Concise descriptions of pipelined computations

✧ How do you compile a Rapid array?
  ➷ Generating a Rapid array from multiple source programs
  ➷ Reconfiguring Rapid from a source program
RaPiD: Reconfigurable Pipelined Datapath

- Linear array of function units
- Function type determined by application
- Function units are connected together as needed using segmented buses
- Data enters the pipeline via input streams and exits via output streams
Section of a Sample Rapid Datapath

- Programmable registers
- Word-based data busses
- Input multiplexers
- Tri-state output drivers
- Bus connector: Open, connected, or up to three register delays
FIR Filter

- Given a fixed set of coefficient weights and an input vector
- Compute the dot product of the coefficient vector and a window of the input vector
- Easily mapped to a linear pipeline

\[ y_6 = \sum \]

\[ \ldots x_9 \ldots x_8 \ldots x_7 \ldots x_6 \ldots x_5 \ldots x_4 \ldots x_3 \ldots x_2 \ldots x_1 \ldots x_0 \]
**FIR Filter**

Each number refers to the index of the stream: e.g. $w_0$, $w_1$, $w_2$.
FIR Filter

Each number refers to the index of the stream: e.g. $w_0$, $w_1$, $w_2$
FIR Filter

Each number refers to the index of the stream: e.g. $w_0$, $w_1$, $w_2$
FIR Filter

Each number refers to the index of the stream: e.g. $w_0$, $w_1$, $w_2$
Configuring the FIR Filter

- Systolic pipeline implements overlapped vector products

- The array is configured to implement this computational pipeline
  - Stages of the FIR pipeline are configured onto the datapath
Configuring Different Filters

✦ Time-multiplexing: trade number of taps vs. sampling rate
  ❍ M taps assigned per multiplier
  ❍ Requires memory in datapath

✦ Symmetric filter coefficients
  ❍ Doubles the number of taps that can be implemented
  ❍ Requires increased control

✦ Followed by downsampling by M
  ❍ Number of taps increased by factor of M
Dynamic Control

- A completely static pipeline is very restricted
  - Virtually all applications need to change computation dynamically
  - Dynamic changes are relatively small
    - Initialize registers, e.g. filter coefficients
    - Memory read/write
    - Stream read/write
    - Data dependent operations, e.g. MIN/MAX
    - Time-multiplexing stage computation

- Solution: Make some of the configuration signals dynamic

- Problem: Where do these signals come from?
Alternatives for Dynamic Control

✦ Per-stage programmed control

- Similar to programmable systolic array/iWARP
- Very expensive, requires synchronization

✦ VLIW/Microprogrammed control

- Very expensive, high instruction bandwidth
The RaPiD Approach

✦ Factor out the computation that does not change
  ➤ Statically configure the underlying datapath
  ➤ Datapath is temporarily hardwired

✦ Remaining control is dynamic
  ➤ Configure an instruction set for the application
  ➤ A programmed controller generates instructions
  ➤ Instruction is pipelined alongside the datapath
  ➤ Each pipeline stage “decodes” the instruction
  ➤ Instruction size is small: typically <16 bits
FIR Filter Control

Control stream contains dynamic control bits.
FIR Filter Control

DATA

CONTROL

Reconfigurable Pipelined Datapaths
FIR Filter Control

DATA

CONTROL

Reconfigurable Pipelined Datapaths
FIR Filter Control
Summary of Control

✧ Hard control:
  - Configures underlying pipelined datapath
  - Changed only when application changes
  - Like FPGA configuration

✧ Soft control:
  - Signals that can change every clock cycle
    - ALU function, multiplexer inputs, etc.
  - Configurably connected to instruction decoder

✧ Only part of soft control may used by an application
  ★ Static control
    Soft control that is constant
  ★ Dynamic control
    Soft control generated by instruction
Soft Control Signals

✧ Static control:
  ➔ Connect control signal to 0/1

✧ Dynamic control:
  ➔ Connect control signal to instruction
Rapid Programming Model

- Pipelined computations are complicated
  - Use a Broadcast model
  - Assumes data can propagate entire length of datapath in one cycle
  - A “datapath instruction”:
    - specifies computation executed by entire datapath in one cycle
    - execution proceeds sequentially in one direction
Rapid-C Datapath Instruction

- Described using a loop

```c
for (s=0; s < 3; s++) {
    if (s==0) in[0] = streamX;
    if (s==0) out = in[0] * W[0];
    else out = out + in[s] * W[s];
    if (s==3) streamY = out;
}
```
Pipelining RaPiD

- The Rapid datapath is pipelined/retimed by the compiler
  - One “datapath instruction” is really executed over multiple cycles
  - Programmer always thinks in broadcast time
Instructions are Broadcast

- Datapath instructions are executed in one cycle
  - Control is broadcast
  - Compiler pipelines control along with data
Instructions are Broadcast

- Datapath instructions are executed in one cycle
  - Control is broadcast
  - Compiler pipelines control along with data
Data Communication in Rapid

- Predominately nearest-neighbor communication
  - To the next stage (broadcast within an instruction)
  - To the next instruction (within a stage)
  - Across N instructions (through local memory)
Rapid-C Process

✧ Each nested loop is a *loop process*
  ➤ The inner loop is a datapath instruction

```c
for (k=0; k < NX-NW; k++)
  for (s=0; s < NW-1; s++) {
    if (s==0) in[0] = streamX;
    if (s==0) out = in[0] * W[0];
    else out = out + in[s] * W[s];
    if (s==NW-1) streamY = out;
  }
```

Reconfigurable Pipelined Datapaths
Rapid-C Programs

✦ Programs are comprised of several loop processes

✦ Processes can run sequentially
  ➢ One process starts when the previous process completes

✦ Processes can run concurrently
  ➢ Computations overlap
  ➢ Processes run in lock-step
    ✦ One inner loop execution per cycle
  ➢ Concurrent processes are synchronized via signal/wait
    ✦ One process waits for the other to send a signal
FIR Filter: Three Sequential Loop Processes

Pipe in[NW];
Reg W[NW];

for (i=0; i < NW; i++)
    for (s=0; s < NW; s++) {
        if (s==0) in[0] = streamW;
        if (i==NW-1) W[s] = in[s];
    }

for (j=0; j < NW-1; j++)
    for (s=0; s < NW; s++) {
        if (s==0) in[0] = streamX;
    }

for (k=0; k < NX-NW; k++)
    for (s=0; s < NW-1; s++) {
        if (s==0) in[0] = streamX;
        if (s==0) out = in[0] * W[0];
        else out = out + in[s] * W[s];
        if (s==NW-1) streamY = out;
    }
FIR Filter Using Concurrent Loop Processes

```plaintext
for (i=0; i < NW; i++)
    for (s=0; s < NW; s++) {
        if (s==0) in[0] = streamW;
        if (i==NW-1) W[s] = in[s];
    }

PAR {
    for (j=0; j < NX; j++)
        if (j==NW) signal(goAhead);
        for (s=0; s < NW; s++) {
            if (s==0) in[0] = streamX;
        }
    wait(goAhead); // Wait for enough inputs
    for (k=0; k < NX-NW; k++)
        for (s=0; s < NW-1; s++) {
            if (s==0) out = in[0] * W[0];
            else out = out + in[s] * W[s];
            if (s==NW-1) streamY = out;
        }
}
```
Rapid-C Programs

✧ Processes are composed to make other processes
  ➩ Sequential composition
  ➩ Parallel composition

✧ The process structure is represented by a Control Tree
  ➩ Each node is a SEQ or PAR

```c
for (i=0; i < NW; i++)
    for (s=0; s < NW; s++)
        if (s==0) in[0] = streamW;
        if (i==NW-1) W[s] = in[s];

for (j=0; j < NX; j++)
    if (j==NW)
        signal(goAhead);
    for (s=0; s < NW; s++)
        if (s==0) in[0] = streamX;

wait(goAhead);
for (k=0; k < NX-NW; k++)
    for (s=0; s < NW-1; s++)
        if (s==0) out = in[0] * W[0];
        else out = out + in[s] * W[s];
    if (s==NW-1) streamY = out;
```
Executing a Rapid Control Tree

Reconfigurable Pipelined Datapaths
Matrix Multiply

A

\[ \begin{bmatrix}
00 & 01 & 02 & 03 \\
10 & 11 & 12 & 13 \\
20 & 21 & 22 & 23 \\
30 & 31 & 32 & 33 \\
\end{bmatrix} \]

\[ \begin{bmatrix}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\end{bmatrix} \]

X

\[ \begin{bmatrix}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\end{bmatrix} \]

\[ \begin{bmatrix}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
\ldots & \ldots & \ldots \\
\end{bmatrix} \]

C

Reconfigurable Pipelined Datapaths
Matrix Multiply

\[
\begin{bmatrix}
00 & 01 & 02 & 03 \\
10 & 11 & 12 & 13 \\
20 & 21 & 22 & 23 \\
30 & 31 & 32 & 33 \\
\vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\times
\begin{bmatrix}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\vdots & \vdots & \vdots \\
\end{bmatrix}
=
\begin{bmatrix}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\vdots & \vdots & \vdots \\
\end{bmatrix}
\]
Matrix Multiply

\[
\begin{array}{cccc}
00 & 01 & 02 & 03 \\
10 & 11 & 12 & 13 \\
20 & 21 & 22 & 23 \\
30 & 31 & 32 & 33 \\
\end{array}
\times
\begin{array}{cccc}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\end{array}
= \begin{array}{cccc}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\end{array}
\]
Matrix Multiply

\[
\begin{align*}
\begin{bmatrix}
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 \\
\end{bmatrix}
& \times
\begin{bmatrix}
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 \\
\end{bmatrix}
= \\
\begin{bmatrix}
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 \\
\end{bmatrix}
\end{align*}
\]
Matrix Multiply

A

00 01 02 03
10 11 12 13
20 21 22 23
30 31 32 33
...

B

00 01 02
10 11 12
20 21 22
30 31 32
...

C

00 01 02
10 11 12
20 21 22
30 31 32
...

00 02
10 12
20 22
30 32

X
+}

00 01 02 03
10 11 12 13
20 21 22 23
30 31 32 33
...

X

00 01 02
10 11 12
20 21 22
30 31 32
...

X

00 01 02
10 11 12
20 21 22
30 31 32
...

X

00 01 02
10 11 12
20 21 22
30 31 32
...

X

00 01 02
10 11 12
20 21 22
30 31 32
...

X
Matrix Multiply

\[
\begin{array}{ccc}
A & X & B \\
\begin{array}{cccc}
00 & 01 & 02 & 03 \\
10 & 11 & 12 & 13 \\
20 & 21 & 22 & 23 \\
30 & 31 & 32 & 33 \\
\ldots & \ldots & \ldots & \ldots \\
\end{array} & \times & \begin{array}{cccc}
00 & 01 & 02 \\
10 & 11 & 12 \\
20 & 21 & 22 \\
30 & 31 & 32 \\
\end{array} \\
\end{array}
\]

\[
\begin{array}{c}
C \\
= \begin{array}{cccc}
00 & 01 & 02 & 02 \\
10 & 11 & 12 & 12 \\
20 & 21 & 22 & 22 \\
30 & 31 & 32 & 32 \\
\ldots & \ldots & \ldots & \ldots \\
\end{array}
\end{array}
\]
Matrix Multiply Program

✧ Three process
  ➔ Load B matrix
  ➔ Compute C values
  ➔ Output C values

```plaintext
for e=0 to M-1
  for f=0 to N-1
    for s=0 to N-1
      ...loading B matrix...
      PAR {
        for i=0 to L-1
          for k=0 to M-1
            if (k==M-1) signal(go);
            for s=0 to N-1
              ...calculation...
            for g=0 to L-1
              wait (go);
              for h=0 to N-1
                ...retire results...
        }
```
Blocked Matrix Multiply

\[ A \times B = C \]
Program for Blocked Matrix Multiply

✦ Load initial B submatrix

✦ Concurrently
  ✐ Load next B submatrix
  ✐ Compute/Accumulate C submatrix
  ✐ Output completed C submatrix

✦ Concurrent processes synchronize
  ✐ Swap double buffered B memories
  ✐ Output C submatrix when completed
Compiling Rapid-C

✦ Balance between programmer and compiler
  ➤ Programmer
    ✦ Specifies basic computation
    ✦ Specifies parallelism using RaPiD model of computation
    ✦ Partitions/schedules sub-computations
    ✦ Optimizes data movement

➤ Compiler
  ✦ Translates inner loops into a datapath circuit
  ✦ Pipelines/retimes computation to meet performance goal
  ✦ Extracts dynamic control information
    ✦ conditionals that use run-time information
  ✦ Constructs instruction format and decoding logic
  ✦ Builds datapath control program
Compiling Rapid-C

for (i=0; i < NW; i++)
    for (s=0; s < NW; s++) {
        if (s==0) in[0] = streamW;
        if (i==NW-1) W[s] = in[s];
    }

wait(goAhead);  // Wait for enough inputs
PAR {
    for (j=0; j < NK; j++)
        if (j==NW) signal(goAhead);
    for (s=0; s < NW; s++) {
        if (s==0) in[0] = streamX;
    }
}

for (j=0; j < NK; j++)
    if (j==NW) signal(goAhead);
for (s=0; s < NW; s++) {
    if (s==0) in[0] = streamX;
}

wait(goAhead);  // Wait for enough inputs
PAR {
    for (k=0; k < NK-NW; k++)
        for (s=0; s < NW-1; s++) {
            if (s==0) out = in[0] * W[0];
            else out = out + in[s] * W[s];
            if (s==NW-1) streamY = out;
        }
}

for (j=0; j < NK; j++)
    if (j==NW) signal(goAhead);
Synthesizing a Rapid Array

 Newtown Generate Rapid datapath that “covers” all spec netlists
   ➢ Union of all function units
   ➢ Sufficient routing
     - Busses with different bit widths
   ➢ Configurable soft control signals
   ➢ Wide enough instruction
   ➢ Enough instruction generators
     - Max parallel processes

 Newtown Provide some “elbow-room”
Current Status

✦ Rapid meta-architecture well-understood

✦ Rapid-C programming language
  ➢ Programs for many applications

✦ Rapid-C compiler
  ➢ Verilog structural netlist
  ➢ Program for datapath controller

✦ Rapid Simulator
  ➢ Executes Verilog netlist and control program
  ➢ Visualization of datapath execution

✦ Place & Route
  ➢ Uses an instance of a Rapid datapath
  ➢ Places and routes datapath and control
  ➢ Pipelines/Retimes to target clock cycle
Future Work

✦ Synthesis of Rapid array netlist
  ➔ What is the best way to cover a set of netlists?
  ➔ How to provide additional elbow-room?

✦ Synthesis of Rapid layout
  ➔ How much can industry tools do?
  ➔ Datapath generator
    ➔ Use standard blocks for functional units
      ➔ Library cells
      ➔ Synthesized cells
    ➔ Generate segmented bus structure from template
    ➔ Generate control structure from template
  ➔ Use standard block for datapath controller
    ➔ Parameterized
Future Work (cont)

✦ Improvements
  ✗ Language features
    ✗ Custom functions
      ✗ Allow arbitrary operations, synthesize the hardware
    ✗ Escapes
    ✗ Pragmas
    ✗ Spatially sequential processes
  ✗ Compiler features
    ✗ Automatic time-multiplexing
    ✗ Optimized control

✦ Multiple configuration contexts
  ✗ e.g. switch between data compression/decompression
  ✗ Use program scope to determine context

✦ System interface issues
Using Multiple Contexts

✦ Sometimes computation phases are quite different
  ➢ Produces lots of dynamic control
  ➢ e.g. switch between motion estimation and 2-D DCT

✦ Solution: provide fast “context switch”
  ➢ Multiple configurations (hard and soft control)
  ➢ Datapath control selects the context
  ➢ Rapid context switch
  ➢ Context switch may be pipelined

✦ Programming multiple contexts
  ➢ Scope in program determines context
  ➢ Compiler compiles different scopes independently
  ➢ Instruction now contains context pointer
The Rapid Research Team

✦ Students
  ➤ Darren Cronquist - architecture and compiler
  ➤ Paul Franklin - architecture and simulator
  ➤ Stefan Berg - simulation, memory interface
  ➤ Miguel Figueroa - applications

✦ Staff
  ➤ Larry McMurchie - place/route
  ➤ Chris Fisher - circuit design and layout

✦ Funding: DARPA and NSF
Overview of Using Rapid

Generating a Rapid Array

- Rapid-C Programs
- Rapid Compiler
- Rapid Array Netlist
- Hardware Synthesis
- Rapid Array

Programming a Rapid Array

- New Rapid-C Program
- Rapid Compiler
- Rapid Program Netlist
- Map Place/Route
- Configuration
- Control Program
- Rapid Array