Seamless Integration of Hardware and Software in Reconfigurable Computing Systems

Miljan Vuletić, Laura Pozzi, and Paolo Ienne
{Miljan.Vuletic, Laura.Pozzi, Paolo.Ienne}@epfl.ch
http://lap.epfl.ch
Ecole Polytechnique Fédéral de Lausanne
The School of Computer and Communication Sciences
Processor Architecture Laboratory
Potentials of Reconfigurable HW

- Prototyping, glue logic, acceleration
- Mask costs direct users toward FPGAs
- Reconfigurable computing mixes SW and HW: Exploit HW parallelism to obtain speedup!
- Major obstacles:
  - Lack of portability
  - Nontransparent HW/SW interfacing
  - Nonstandardised programming paradigms
Microprocessors: Power of programmability

- Temporal computing engines
- Different kinds of problems
- Programming instead of logic design
- Memory abstraction
- Standardised programming paradigms

HW/SW Interfacing? Portability?
Compilers: Power of Portability

- Hide machine details
- Improve productivity
- Allow code portability
- Achieve good efficiency

for(i = 0; i < n; i++)
a[i] += b[i]*c[i];

ld r2, 0(r5)
ld r3, 0(r6)
mul r2, r2, r3
Compilers (Synthesizers) for Reconfigurable Hardware

- Hide technology details
- Improve productivity
- Achieve good efficiency
- But interfacing/portability?

process(a, b) begin
c <= a + b;
end process;

Front End

Back End

(cell addsub_1
  (cellType generic)
  (interface ...
Reconfigurable toward General-purpose computing

- Ease of programming: transparent software code
- Ease of hardware design: platform-agnostic accelerators
- Application portability: recompilation and resynthesis suffice
- Abstraction: the price to pay should not be too high!
Related Work

- Component-based design and IP-reuse: [AMBA, VCI, Gharsalli02]
- Virtualisation of Reconfigurable Resources: [Caspi00, Dales03, Walder03]
- OS support for interfacing reconfigurable HW: [Leong01, Nollet03]
- Programming paradigms for RC: [Hauser97, Mencer01, Vassiliadis04, Brebner04]
Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives
Standard Multithreading
Multithreading: Memory Point of View

- Same memory address space
- Separate stacks

for(I=0;i<size;i++)
C[i] = A[i] + B[i];
Adding Vectors in Software

/* Typical SW version */

int *A, *B, *C;
int add_vectors(int *, int *, int *, int);
int thrd1;

read(A, SIZE); read(B, SIZE);

thrd1 = thread_create(add_vectors, A, B, C, SIZE);
do_some_work();
thread_join(thrd1);
Extended Multithreading
Extended Multithreading

- No standard support when critical threads are to be mapped onto hardware
Extended Multithreading: Memory Point of View

- Disjoint SW and HW memory address spaces
- Wrapper Thread

```c
for(I=0;i<size;i++)
    C[i] = A[i] + B[i];
```
Typical Coprocessor System

- CPU
- MMU
- Virtual Memory
- Scratch Memory
- FPGA
- CoProc
- Main Memory
- Virtual Memory Manager
- Operating System

Control
Memory Copy

March 2005, Vuletić et al.
Burdensome Programming...

```c
/* Typical HW accelerator version */
int *A, *B, *C; ... read(A, SIZE); read(B, SIZE);
d_chunk = BUF_SIZE/3; d_ptr = 0;
write(HWACC_CTRL, INIT);
while (d_ptr < SIZE) {
    copy(A + d_ptr, BUF_BASE, d_chunk);
    copy(B + d_ptr, BUF_BASE + d_chunk, d_chunk);
    write(HWACC_CTRL, ADD_VECTORS);
    while() {
        if (read(HWACC_STATUS) == FINISHED) {
            copy(BUF_BASE + 2*d_chunk, C + d_ptr, d_chunk);
            break;
        } else {
            do_some_work();
        }
    }
    d_ptr += d_chunk;
}...
```
Extended Multithreading: Memory Point of View

- Same virtual memory address space
- Virtualisation layer

```c
for (i = 0; i < size; i++)
    C[i] = A[i] + B[i];
```

```
thread_create
add_vectors_hw
A, B, C, size

return and join
```

Virtual Memory

- \( A[] \)  
- \( B[] \)  
- \( C[] \)  

\( \text{size} \)
Transparent Programming...

/* Transparent version */

int *A, *B, *C;
int hw_add_vectors(int *, int *, int *, int);
int hwacc_thrd;

read(A, SIZE); read(B, SIZE);

hwacc_thrd = thread_create(hw_add_vectors, A, B, C, SIZE);
do_some_work();
thread_join(hwacc_thrd);

Exactly Same as Software!!!
Extended Multithreading

- Abstraction layer extended to support hardware accelerators
Virtual Memory Window System

- CPU
- MMU
- Main Memory
- Virtual Memory Manager
- Operating System

Virtual Address

SW

HW

CoProc

FPGA

Virtual Memory

WMU

Window Memory

Physical Address

March 2005, Vuletić et al.
Accelerator Initiated Data Transfers

OS (SW) → μP Exec → CP Exec → Idle → μP Exec → CP Exec → Wait → CP Exec → Window Memory

Control → Memory Copy

Window Memory Accesses
Platform-dependent and Portable VHDL-like coding

-- Platform dependent
-- Initialisation
ptr_a <= BUF_ADDR;
ptr_b <= BUF_ADDR + BUF_SIZE/3;
ptr_c <= BUF_ADDR + 2*BUF_SIZE;

-- Computation
cycle 1:
   -- partition of A[]
   BUF_ADDR <= ptr_a;
   BUF_ACCESS <= '1';
   BUF_WR <= '0';

-- Portable
-- Initialisation
ptr_a <= A;
ptr_b <= B;
ptr_c <= C;

-- Computation
cycle 1:
   -- object A[]
   VIRTMEM_ADDR <= ptr_a;
   VIRTMEM_ACCESS <= '1';
   VIRTMEM_WR <= '0';
Platform-dependent and Portable VHDL-like coding (II)

-- Platform dependent

 cycle 2:
    reg_a <= BUF_DATAIN;
    -- partition of B[]
    BUF_ADDR <= ptr_b;
    BUF_ACCESS <= '1';
    BUF_WR <= '0';

-- Portable

 cycle 2:
    reg_a <= VIRTMEM_DATAIN;
    -- object B[]
    VIRTMEM_ADDR <= ptr_b;
    VIRTMEM_ACCESS <= '1';
    VIRTMEM_WR <= '0';
cycle 3:
  reg_b <= BUF_DATAIN;
  reg_c <= reg_a + reg_b;
  -- partition of C[]
  BUF_ADDR <= ptr_c;
  BUF_ACCESS <= '1';
  BUF_WR <= '1';
  ptr_{a, b, c} <= ptr_{a, b, c} + 1;
  if (ptr_c = BUF_SIZE) then
    -- finished for a data chunk
    partial_finish();
  else cycle 1;
  end if;

cycle 3:
  reg_b <= VIRTMEM_DATAIN;
  reg_c <= reg_a + reg_b;
  -- object C[]
  VIRTMEM_ADDR <= ptr_c;
  VIRTMEM_ACCESS <= '1';
  VIRTMEM_WR <= '1';
  ptr_{a, b, c} <= ptr_{a, b, c} + 1;
  if (ptr_c = SIZE) then
    -- finished for the entire vectors
    finish();
  else cycle 1;
  end if;
Experimental Setup

- Board based on Altera Excalibur EPXA1: ARM (133MHz), FPGA, SDRAM (64MB), Linux
- On-chip DP RAM (16KB) directly accessible by FPGA
- WMU in synthesizable VHDL
- VMW manager as Linux kernel module
- IDEA (encryption/decryption)
- ADPCM Decoder (MediaBench) coprocessor
Experimental Setup: ROKEPXA Board
Programming Example: IDEA Cryptography

/* main function */
void main() {
    int *A, *B, n64;
    ...
    read(A, n64);
    idea_encrypt(A, B, n64);
    ...
}

/* library function */
int idea_encrypt(int *A, int *B, int n64) {
    struct cp_param param;
    ...
    param.u.nparam = 3;
    param.p[0] = A;
    param.p[1] = B;
    param.p[2] = n64;
    FPGA_EXECUTE(HW_IDEA, &param);
    return param.u.retry;
}

March 2005, Vuletić et al.
Experimental Results

![Bar charts showing execution time for IDEA and ADPCM with different input data sizes (8KB and 2KB) and coprocessor versions (Typical and VMW).](image)

- **IDEA**
  - Input data size: 8KB
  - Execution time: 52.8 ms
  - Typical Coprocessor: 2.3 ms, 22.9x faster
  - VMW Coprocessor: 3.5 ms, 15.1x faster

- **ADPCM**
  - Input data size: 2KB
  - Execution time: 4.8 ms
  - Typical Coprocessor: 1.0 ms, 4.8x faster
  - VMW Coprocessor: 1.7 ms, 2.8x faster

**Legend**
- **Pure SW version:**
  - Red: SW
  - Blue: HW Time
  - Pink: Manage Time
  - Green: Copy Time

---

*EPFL*  
École Polytechnique Fédérale de Lausanne

March 2005, Vuletić et al.
# WMU Area Overhead

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Number of Units</th>
<th>WMU Area (%)</th>
<th>WMU Area, ADPCM (%)</th>
<th>WMU Area, IDEA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Cell</td>
<td>576</td>
<td>14</td>
<td>48</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>5</td>
<td>19</td>
<td>83</td>
<td>45</td>
</tr>
</tbody>
</table>
Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives
Virtual Memory Window System

- CPU
- MMU
- Virtual Memory
- Main Memory
- FPGA
- VMW Manager
- CoProc
- WMU
- Window Memory
- Physical Address
- Virtual Address
- SW
- HW
- RSoC

March 2005, Vuletić et al.
VMW with No Prefetching

Legend:
- **MT** – Management Time
- **CT** – Copy Time
- **ST** – Sleep Time
- **RT** – Response Time
- **HT** – Hardware Time
VMW with Prefetching

Legend:
- **MT** – Management Time
- **CT** – Copy Time
- **ST** – Sleep Time
- **RT** – Response Time
- **HT** – Hardware Time
Window Management Unit (WMU): Page Access Detection

TLB

VirtPageNo → CAM → RAM → Hit/Miss

Invalid

PhyPageNo

Dirty

“00000100”

1-hot

Valid

Hit

AIR – Access Indicator Register

AMR – Access Mask Register

Page Access
ADPCM Decoder
Prefetching Improvements

![Graph showing execution time for different input data sizes and prefetching conditions.]

- No Prefetching: 1.2 miss/page
- Prefetching: ~0 miss/page
IDEA Encryption for 64KB of Input Data

Execution Time (ms)

<table>
<thead>
<tr>
<th>Number of Window Memory Pages</th>
<th>No Prefetching</th>
<th>Prefetching</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10 (Manage)</td>
<td>20 (Copy)</td>
</tr>
<tr>
<td>8</td>
<td>20 (Manage)</td>
<td>30 (Copy)</td>
</tr>
<tr>
<td>16</td>
<td>30 (Manage)</td>
<td>40 (Copy)</td>
</tr>
<tr>
<td>32</td>
<td>40 (Manage)</td>
<td>50 (Copy)</td>
</tr>
</tbody>
</table>

Legend:
- Purple: Manage
- Yellow: Sleep
- Red: Copy
Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives
Malloc Example

![Diagram showing the interaction between SW (Software) and HW (Hardware) with calls and returns involving memory allocation using malloc function.]

Support Library

Accelerator Thread

retval = malloc(size)

Virtual Memory
Unrestricted Automated Synthesis: Function Calls; Malloc Example

ioctl(fd, VMW_IOC_START, &params);
...
while(! params.hwret) {
    switch(params.cback) {
        case 1: … break;
        case 2: … break;
        case 3: params.retval = malloc(params.p[0]);
            ioctl(fd, VMW_IOC_RESUME, &params); break;
        ...
        case n:
            }
    }
}
Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives
Portable Hardware Accelerators for Java Virtual Machines

Dubach et al., February 2004
Java Accelerator Results

Execution time for IDEA

<table>
<thead>
<tr>
<th># blocks</th>
<th>256 (2KB)</th>
<th>512 (4KB)</th>
<th>1024 (8KB)</th>
<th>2048 (16KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Java</td>
<td>33 ms</td>
<td>65 ms</td>
<td>130 ms</td>
<td>260 ms</td>
</tr>
<tr>
<td>Java + CP</td>
<td>19 x 48</td>
<td>26 x 30</td>
<td>31 x 41</td>
<td>30 x 49</td>
</tr>
<tr>
<td>Java + CP (PF)</td>
<td>19 x 48</td>
<td>26 x 30</td>
<td>31 x 41</td>
<td>30 x 49</td>
</tr>
</tbody>
</table>

Execution time for AES

<table>
<thead>
<tr>
<th># blocks</th>
<th>128 (2KB)</th>
<th>256 (4KB)</th>
<th>512 (8KB)</th>
<th>1024 (16KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Java</td>
<td>46 ms</td>
<td>95 ms</td>
<td>190 ms</td>
<td>380 ms</td>
</tr>
<tr>
<td>Java + CP</td>
<td>36 x 44</td>
<td>38 x 44</td>
<td>47 x 62</td>
<td>53 x 77</td>
</tr>
<tr>
<td>Java + CP (PF)</td>
<td>36 x 44</td>
<td>38 x 44</td>
<td>47 x 62</td>
<td>53 x 77</td>
</tr>
</tbody>
</table>
Unrestricted Java Bytecode Synthesis

Dubach, January 2005, Masters thesis
IDEA Synthesis Results

[Bar chart showing comparison of different programming languages and hardware implementations]
Virtualisation Layer: Portability of SW/HW Applications

CPU

Virtual Memory

Operating System

Virtual Memory Manager

Virtualisation Manager

Main Memory

ARM

MMU

WMU

Dual Port Memory

FPGA

APEX20KE

ALTERA EXCALIBUR

AMBA

March 2005, Vuletić et al.
Virtualisation Layer: Portability of SW/HW Applications (II)
Portability for MPEG4 HW Reference
(Supported by MPEG4 group and Xilinx)

CPU

FPGA Virtex II

No Change in C Code!

No Change in HDL Code!

Virtual Memory

Operating System

Virtual Memory Manager

Virtualisation Manager

Main Memory

Pentium II

MMU

Cardbus Controller

WMU

DRAM

CardBus

PLB

PC

Annapolis Wildcard

EPFL

ECOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

March 2005, Vuletić et al.
VMW for Virtual Sockets (MPEG4 Standardisation Group)
Virtualisation Layer for Transparent HW/SW Multithreading

Application Software (executing on µProc)

HW/SW Thread Library

System Software Support

Communication Assistant

Application Hardware (executing on FPGA)
Reconfigurable Parallel Architecture

Interconnection Network

- \(\mu \text{Proc}\)
- Main Memory
- Local Memory
- HW Acc
- \(\text{CA}\)
- Communication Assistant

March 2005, Vuletić et al.
Conclusions

- Virtualisation layer for seamless hardware and software interfacing
- Portable reconfigurable applications through recompilation and resynthesis
- Compiler, synthesizer and OS suffice!
- Parallel programming paradigm for reconfigurable computing systems
Conclusions (II)

- Significant speedup on a real platform with limited overhead!
- Translation should go to VLSI!
- Runtime memory optimisation: Benefits without any change in user SW/HW code
- Dynamic prefetching almost hides memory latency!
- Support for unrestricted automated synthesis
- Portable prototyping framework for SW/HW partitioning
Seamless Integration of Hardware and Software in Reconfigurable Computing Systems

Miljan Vuletić, Laura Pozzi, and Paolo Ienne
{Miljan.Vuletic, Laura.Pozzi, Paolo.Ienne}@epfl.ch
http://lap.epfl.ch
Ecole Polytechnique Fédéral de Lausanne
The School of Computer and Communication Sciences
Processor Architecture Laboratory