Buffer Allocation for Exposed Datapath Architectures

Anoop Bhagyanath and Klaus Schneider

Embedded Systems Group
University of Kaiserslautern

MCSoC 2022

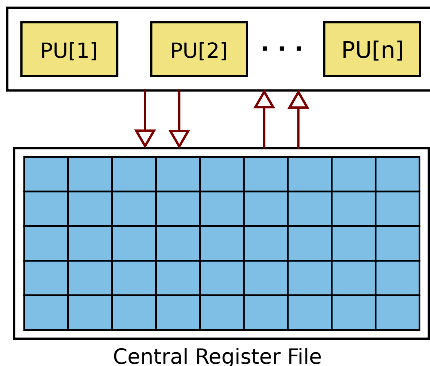
Outline

1. Buffered Exposed Datapath (BED) Architectures
2. BED Code Generation
3. Preliminary Experimental Results
4. Future Work

Instruction-Level Parallelism (ILP)

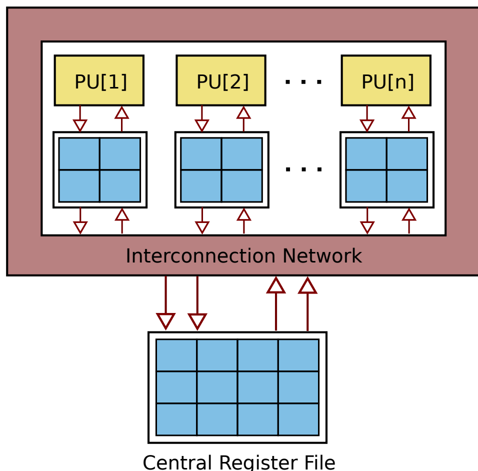
- increasingly compute-and-data-intensive embedded applications
- general trend is to utilize multiple processor cores, however,
  - ⇝ difficulty in multithreaded programming
  - ⇝ difficulty in program execution time analysis
- improve *instruction-level parallelism (ILP)*

Conventional Processor Architectures



- instructions read operands from and write result to registers in a central register file
- ILP limited due to poorly scaling central register file
  - near-quadratic growth of area, power, and access time with increase in registers and register file ports
- led to the notion of *exposing processor datapaths to compiler*

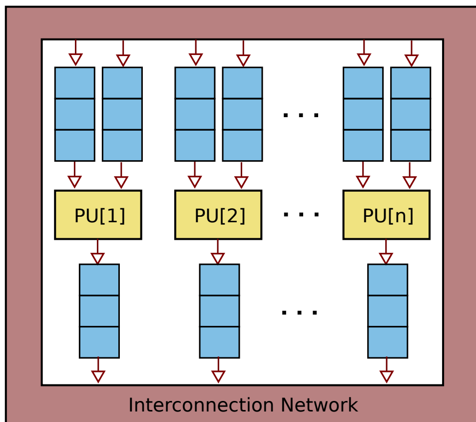
Exposed Datapath Architectures



- an interconnected set of small PU-local register files
- compiler can move values directly between PUs
- reduces the need for registers and ports in the central register file
- still difficult to provide scalable ILP
  - increasing number of registers still difficult for a fixed PU count
  - compilers rely on static scheduling to avoid race conditions, limiting ILP

Buffered Exposed Datapath (BED) Architecture

Abstract Execution Model



- FIFO buffers at PU input and output ports
- a PU fires when sufficient values available at input buffer heads and empty space available in the output buffer
- a fully decentralized architecture that scale (quasi-)linearly
- no static scheduling, rather the compiler must order instructions on PUs appropriately

Code Generation Basics

$x \longleftarrow \ldots$
$y \longleftarrow \ldots$
$z \longleftarrow \ldots$
$\ldots \longleftarrow y$
$\ldots \longleftarrow x$
$\ldots \longleftarrow z$
$\ldots \longleftarrow x$

- *produce values in each output buffer in the same order as they should be consumed*
  - $\rightsquigarrow$ always possible given enough PUs
- executing the example program on a 3-PU BED machine
  - $\rightsquigarrow$ values x, y, z may be produced in different output buffers which may then be accessed in any order
- *minimal number of output buffers (equivalently PUs) required to execute the program?*

Code Generation Basics

$$
\begin{array}{lcl}
x & \longleftarrow & \ldots \\
y & \longleftarrow & \ldots \\
z & \longleftarrow & \ldots \\
\ldots & \longleftarrow & y \\
\ldots & \longleftarrow & x \\
\ldots & \longleftarrow & z \\
\ldots & \longleftarrow & x
\end{array}
$$

- *produce values in each output buffer in the same order as they should be consumed*
  - ⇝ always possible given enough PUs
- executing the example program on a 3-PU BED machine
  - ⇝ values x, y, z may be produced in different output buffers which may then be accessed in any order
- *minimal number of output buffers (equivalently PUs) required to execute the program?*

Code Generation Basics

$$
\begin{array}{lcl}
x & \longleftarrow & \ldots \\
y & \longleftarrow & \ldots \\
z & \longleftarrow & \ldots \\
\ldots & \longleftarrow & y \\
\ldots & \longleftarrow & x \\
\ldots & \longleftarrow & z \\
\ldots & \longleftarrow & x
\end{array}
$$

- *produce values in each output buffer in the same order as they should be consumed*
  - ⇝ always possible given enough PUs
- executing the example program on a 3-PU BED machine
  - ⇝ values x, y, z may be produced in different output buffers which may then be accessed in any order
- *minimal number of output buffers (equivalently PUs) required to execute the program?*

Buffer Allocation - Idea

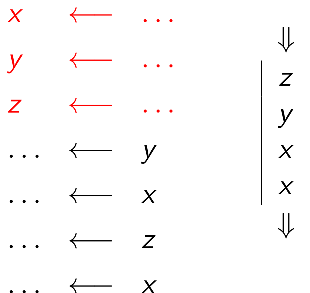
- *handle instruction ordering and buffer allocation separately*
  - ⇝ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 1-PU BED machine
  - ⇝ x blocks y
  - ⇝ x blocks z

Buffer Allocation - Idea

$$
\begin{aligned}
x &\longleftarrow \dots \\
y &\longleftarrow \dots \\
z &\longleftarrow \dots \\
\dots &\longleftarrow y \\
\dots &\longleftarrow x \\
\dots &\longleftarrow z \\
\dots &\longleftarrow x
\end{aligned}
$$

- *handle instruction ordering and buffer allocation separately*
  - $\rightsquigarrow$ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 1-PU BED machine
  - $\rightsquigarrow$ x blocks y
  - $\rightsquigarrow$ x blocks z

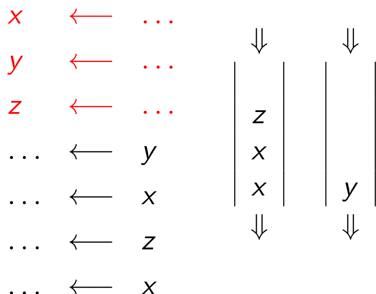
Buffer Allocation - Idea



$\Downarrow$ | z | y | x | x | $\Downarrow$

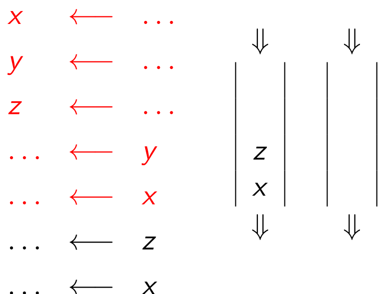
- *handle instruction ordering and buffer allocation separately*
  - $\rightsquigarrow$ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 1-PU BED machine
  - $\rightsquigarrow$ x blocks y
  - $\rightsquigarrow$ x blocks z

Buffer Allocation - Idea



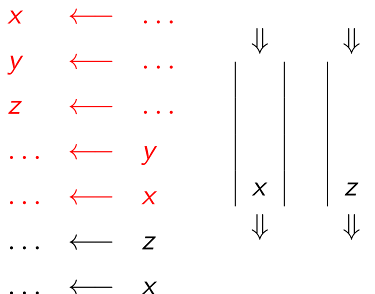
- *handle instruction ordering and buffer allocation separately*
  - $\rightsquigarrow$ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 2-PU BED machine
  - $\rightsquigarrow$ x blocks y
  - $\rightsquigarrow$ x blocks z

Buffer Allocation - Idea



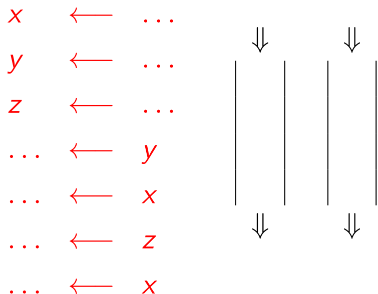
- *handle instruction ordering and buffer allocation separately*
  - ⇝ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 2-PU BED machine
  - ⇝ x blocks y
  - ⇝ x blocks z

Buffer Allocation - Idea



- *handle instruction ordering and buffer allocation separately*
  - ⇝ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 2-PU BED machine
  - ⇝ x blocks y
  - ⇝ x blocks z

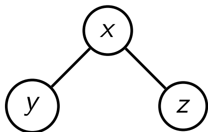
Buffer Allocation - Idea



- *handle instruction ordering and buffer allocation separately*
  - ⇝ *perform buffer allocation for a given instruction order*
- consider execution of the example program on a 2-PU BED machine
  - ⇝ x blocks y
  - ⇝ x blocks z

Buffer Allocation - Computation

$$
\begin{array}{lcl}
x & \longleftarrow & \ldots \\
y & \longleftarrow & \ldots \\
z & \longleftarrow & \ldots \\
\ldots & \longleftarrow & y \\
\ldots & \longleftarrow & x \\
\ldots & \longleftarrow & z \\
\ldots & \longleftarrow & x
\end{array}
$$

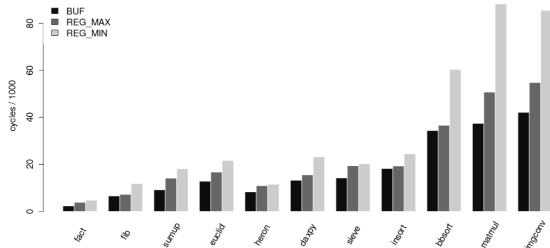


- For any pair of variables x and y, we say x blocks y if there is a program point p such that the following path exists:

$$\text{Def}(x) \rightarrow \text{Def}(y) \overbrace{\rightarrow}^{p} \text{Use}(y) \rightarrow \text{Use}(x)$$

- determine existence of p using dataflow analyses
- variables x and y interfere if x blocks y or y blocks x
- construct interference graph whose coloring will yield valid buffer allocation

Preliminary Experimental Results



buffer-based and register-based (single-ported and multi-ported) execution times of benchmarks on a minimal number of PUs

- reasonable buffer sizes
- due to saved register writes, buffer-based execution is faster than ideal multi-ported register file based execution
- due to contention of concurrent register access on single port, single-ported register file based execution is slowest

Future Work

- compiler
  - develop techniques to maximize ILP
  - take into account buffer depths
- hardware prototyping
  - compare on-chip storage capacity of FIFO buffers with registers
  - performance comparison with conventional processors

Thank You!

Questions?