# Partitioned Register File for TTAs

Authors: Johan Janssen and Henk Corporaal

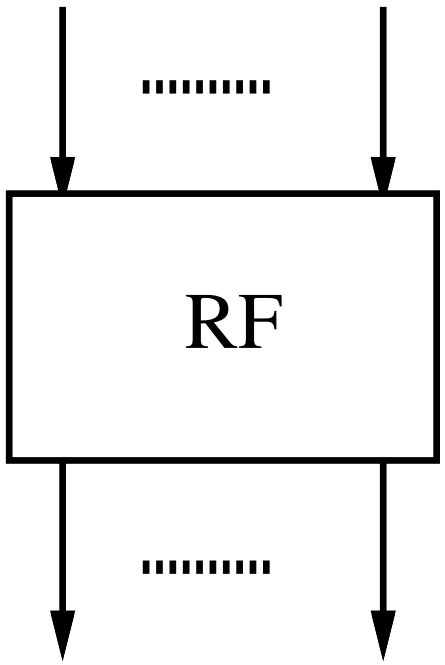
Department of Electrical Engineering
Delft University of Technology
P.O.Box 5031
2600 GA Delft
The Netherlands

# Overview

- Introduction
- Transport Triggered Architectures
- The Move-compiler environment
- Register allocation
  - For a monolithic RF
  - For a partitioned RF
- Equal area compiling
- Conclusions

## Motivation

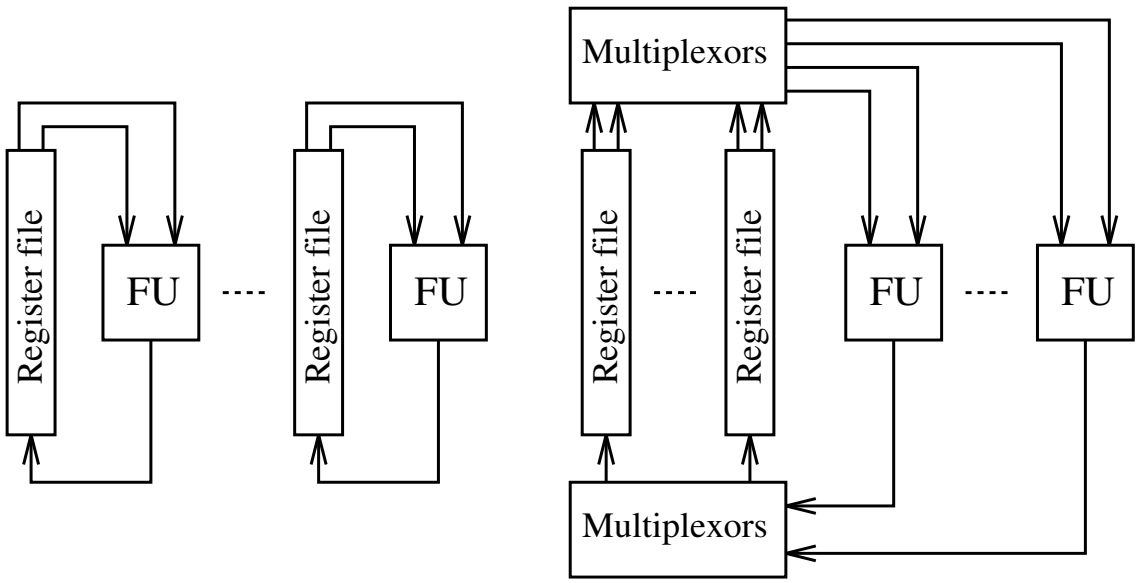
An architecture that exploits ILP needs a large multi-ported register file (RF).




**Disadvantages**

- Difficult to realize
- Increases chip-area
- Increases access time

## Distributed versus Partitioned RF




Distributed register file

Partitioned register file

- Distributed RF
  - Clusters of FUs
  - Full connectivity between FUs and RFs.
  - Requires inter-cluster copy operations
- Partitioned RF
  - Less connectivity between FUs and RFs.
  - Every FU has direct access to any register.
  - No extra copies required
  - Introduction of access conflicts.

# Transport Triggered Architectures (TTAs)

## Characteristics of TTAs

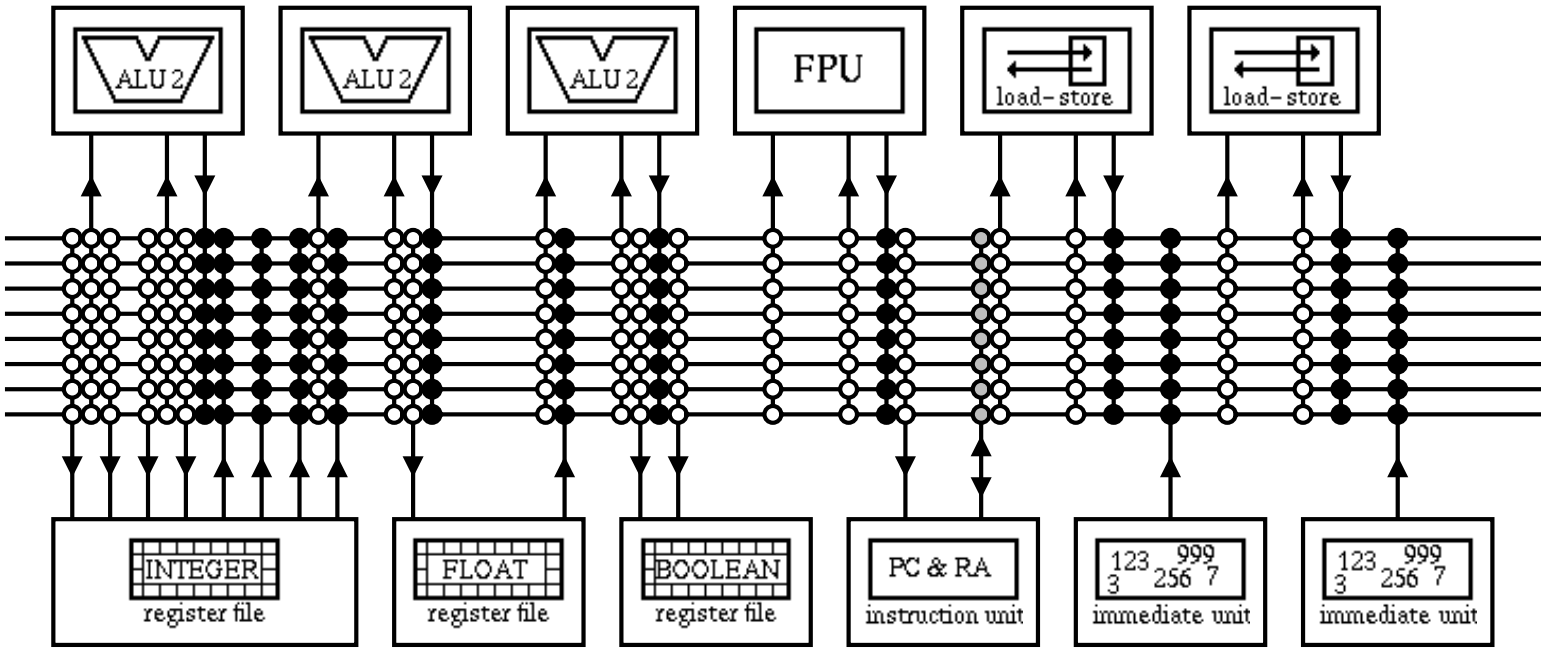
- Exploits Instruction level parallelism
- Instead of specifying operations, transports are specified.

  *Example:*

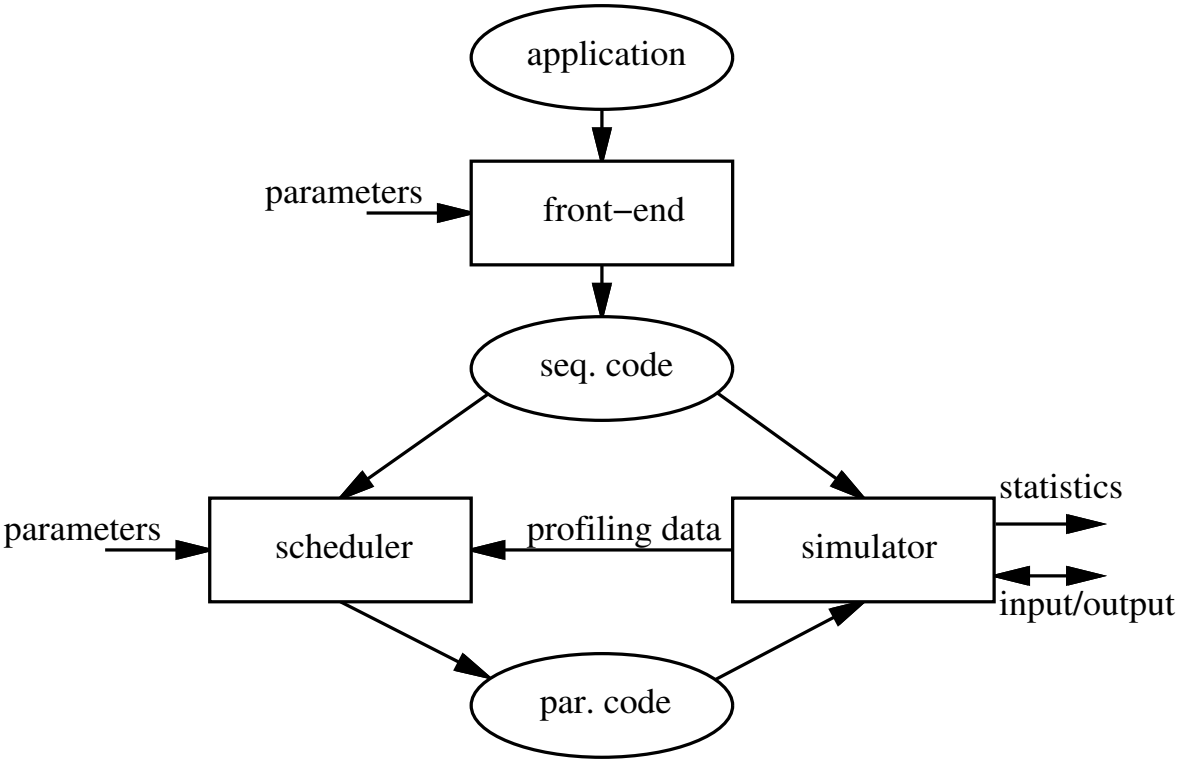
  `ADD r3,r2,r1` $\Rightarrow$ `r1` $\rightarrow$ $\texttt{O1}_{ADD}$; `r2` $\rightarrow$ $\texttt{O2}_{ADD}$
  $\texttt{R}_{ADD}$ $\rightarrow$ `r3`

- More scheduling freedom
- Modular structure
- Flexible and scalable

# Transport Triggered Architectures (TTAs)



# The Move-compiler environment




## Scheduler Characteristics

- Uses profiling information
- Parameterized
- Region scheduling scope
- Applies predicated execution
- Applies smart register allocation *before* scheduling
- Applies TTA specific optimizations

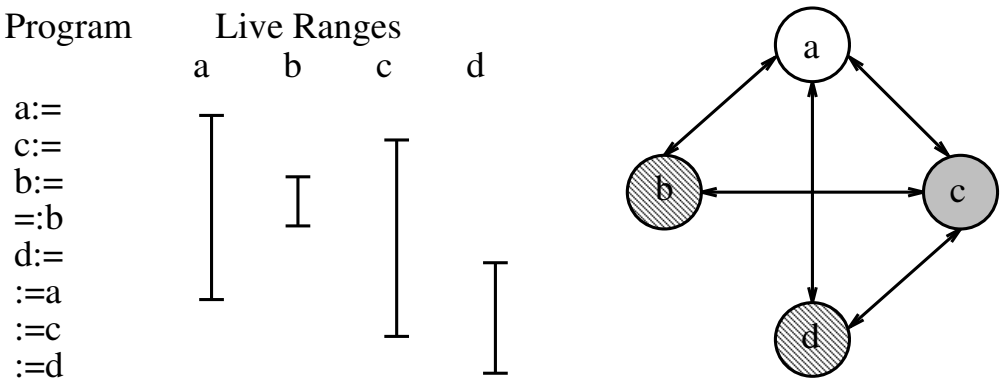
# Register allocation

- Introduction to register allocation
- Allocation for a monolithic RF
- Allocation for a partitioned RF
  - Simple distribution methods
  - Advanced distribution methods
- Evaluation

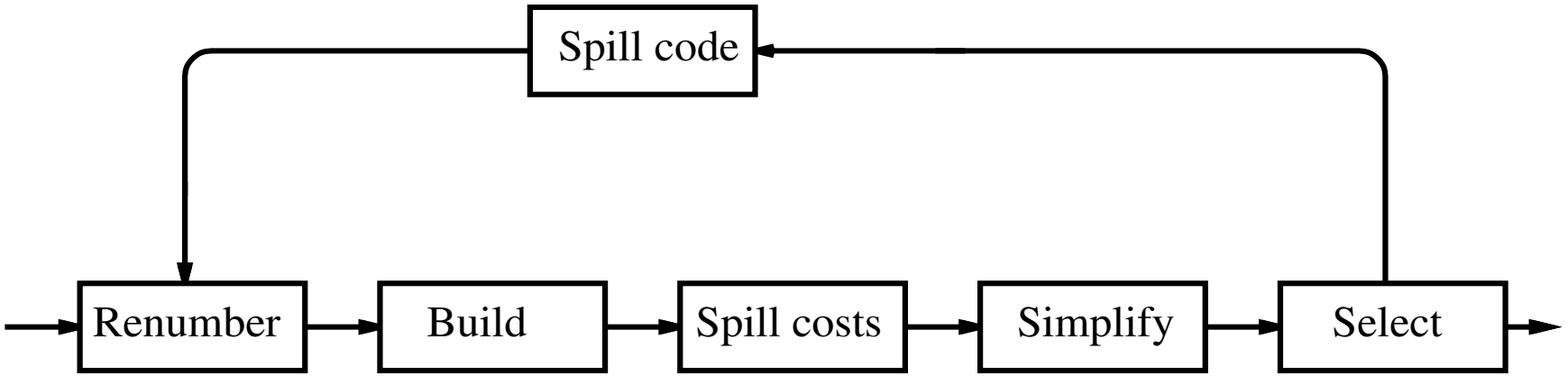
## Register allocation

The problem of finding the most efficient mapping of program variables to the registers of the target machine

- A variable is *defined* at a point in a program when a value is assigned to it.
- A variable is *used* at a point in a program when its value is referenced in an expression.
- A variable is said to be *live* at a point if it has been defined earlier and will be used later.
- The *live range* of a variable is the execution range between definitions and uses of a variable.



**Scheme of the optimistic register allocator**




The *Select phase* selects a color(= machine register) for a variable that minimizes the heuristic:

$$h1 = \; fdep(col, var) + caller_callee(col, var)$$

where:

$fdep(col, var)$ : a measure for the introduction of false dependencies

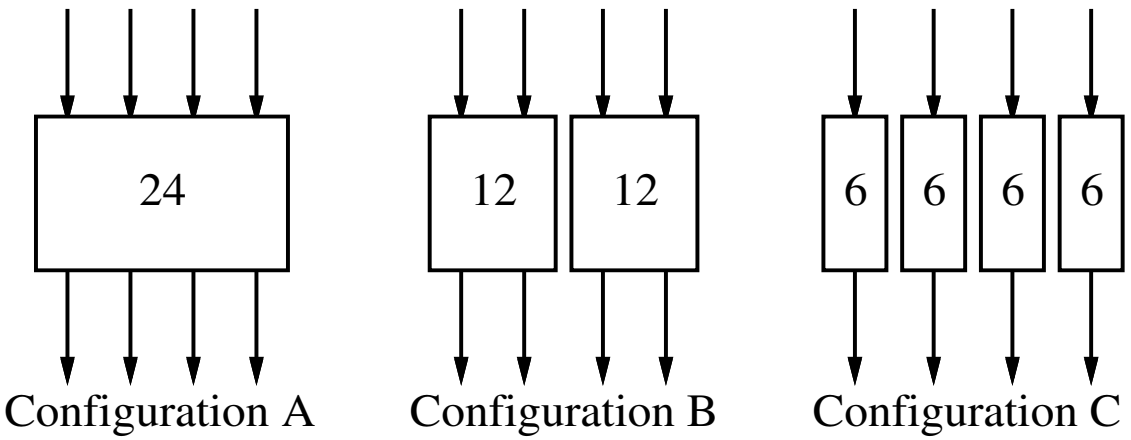
$caller_callee(col, var)$ : cost for mapping $var$ on a caller or callee saved register

# Register allocation for a partitioned RF

## The register file configurations

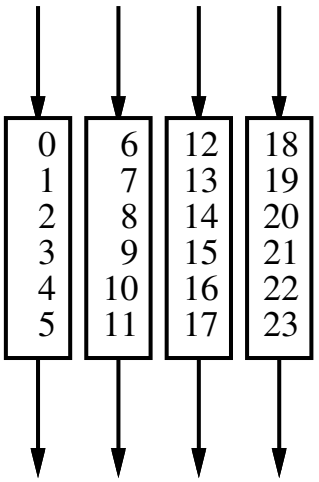
For each configuration:

- Sum of write ports is 4
- Sum of read ports is 4
- Sum of registers is 24

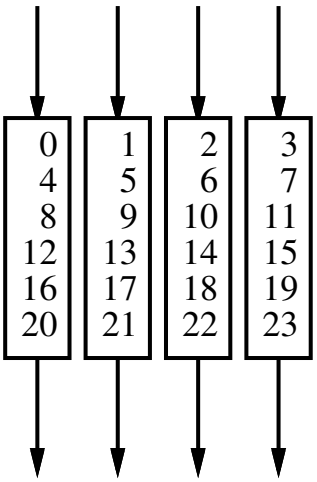


# Register allocation for a partitioned RF

## Simple Distribution methods




Vertical distribution



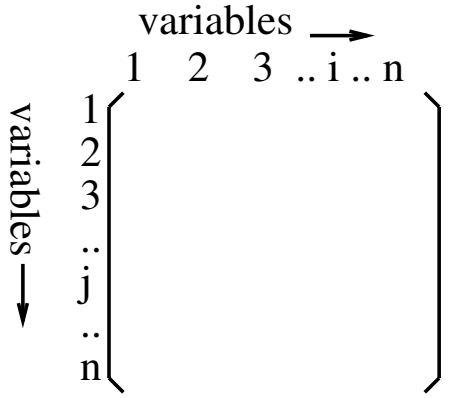

Horizontal distibution

| **benchm.** | **OPC** | $v(B)$ | $v(C)$ | $h(B)$ | $h(C)$ |
|---|---|---|---|---|---|
| bison | 2.7 | 102.1% | 113.2% | 101.6% | 105.2% |
| compress | 2.2 | 103.9% | 106.2% | 103.6% | 105.3% |
| diff | 2.1 | 101.6% | 124.7% | 100.7% | 101.8% |
| dhrystone | 2.9 | 101.9% | 119.0% | 100.4% | 106.7% |
| djpeg | 2.7 | 104.7% | 116.0% | 102.7% | 105.2% |
| expand | 3.1 | 104.1% | 122.7% | 104.1% | 115.5% |
| flex | 2.7 | 103.0% | 112.3% | 100.9% | 104.3% |
| mpeg_p. | 2.7 | 104.9% | 115.0% | 102.2% | 106.6% |
| od | 3.1 | 104.0% | 117.1% | 102.4% | 107.9% |
| sieve | 2.7 | 107.7% | 119.0% | 100.0% | 112.2% |
| sort | 3.1 | 101.8% | 123.0% | 102.9% | 107.2% |
| stanford | 2.9 | 103.0% | 117.6% | 100.6% | 106.4% |
| unexpand | 2.9 | 100.3% | 114.6% | 103.3% | 107.3% |
| uniq | 3.3 | 101.2% | 117.9% | 102.6% | 110.4% |
| wc | 2.9 | 100.0% | 100.5% | 100.0% | 106.2% |
| **average** | 2.8 | 102.9% | 115.9% | 101.9% | 107.2% |

# Register allocation for a partitioned RF

## Data independence heuristic

Tries to avoid the mapping of variables of transports that are independent to the same RF.




Matrix Entry:

$$m(i,j) = (\frac{p(BB_1, BB_2) * freq_1 * freq_2}{dist(transport_1, transport_2)}$$

- $i, j$ are the sources or destinations of the transports
- $p$ is the probability that basic block $BB_1$ to which $transport_1$ belongs is succeeded by $BB_2$ to which $transport_2$ belongs
- $freq_1, freq_2$ are the frequencies of the basic blocks.
- $dist(transport_1, transport_2)$ is the distance between the transports.

## Register allocation for a partitioned RF

During allocation a color is selected that has the lowest RF-conflicts with the already allocated variables using:

$$Indep(col, var) = \sum_{j}^{n} m(var, j) \quad \forall \ j \ \ allocated\ in\ RF\ of\ col$$

In the *Select phase* the following heuristic is used:

$$h2 = \quad fdep(col, var) + caller_callee(col, var) + Indep(col, var)$$

where $Indep(col, var)$ returns a value that is a measure for the number of independences when $var$ is placed in the RF of $col$.

## Intra-operation heuristic

Tries to avoid multiple reads within a single operation from the same RF.

$$h3 = \ fdep(col, var) + caller_callee(col, var) + Intra(col, var)$$

where $Intra(col, var)$ returns a value that is a measure for the number of clashes when $var$ is placed in the RF of $col$.

# Register allocation for a partitioned RF

## Results of the advanced distribution methods

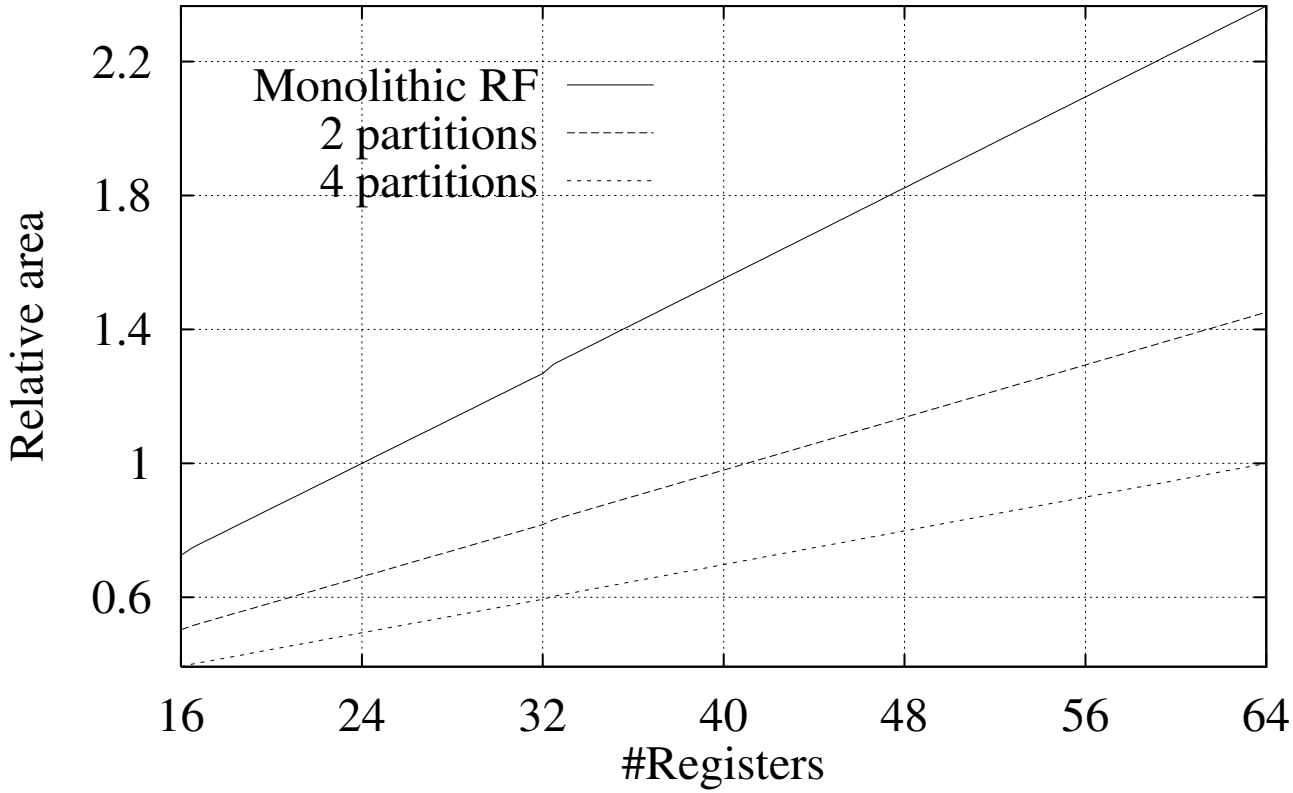
| **benchmark** | $h2(B)$ | $h2(C)$ | $h3(B)$ | $h3(C)$ |
|---|---|---|---|---|
| bison | 102.2% | 102.0% | 100.4% | 103.4% |
| compress | 100.1% | 110.1% | 101.6% | 103.7% |
| diff | 99.9% | 101.0% | 99.9% | 100.5% |
| dhrystone | 98.9% | 111.9% | 99.3% | 104.5% |
| djpeg | 101.7% | 104.0% | 101.7% | 107.3% |
| expand | 104.5% | 100.0% | 104.5% | 120.2% |
| flex | 101.2% | 103.5% | 101.4% | 104.0% |
| mpeg_play | 101.3% | 112.6% | 99.7% | 103.4% |
| od | 103.5% | 109.3% | 104.3% | 107.1% |
| sieve | 101.4% | 106.2% | 101.7% | 105.3% |
| sort | 100.9% | 107.5% | 100.2% | 105.3% |
| stanford | 101.7% | 105.1% | 100.5% | 107.6% |
| unexpand | 103.1% | 107.0% | 96.7% | 103.7% |
| uniq | 102.9% | 108.1% | 102.1% | 106.4% |
| wc | 105.5% | 109.5% | 105.8% | 103.9% |
| **average** | 101.9% | 106.5% | 101.3% | 105.8% |

## Cycle Count Evaluation

- Only a small performance loss in terms of cycles has to be accepted
- The choice of the distribution method has a large impact on the performance (vertical 15.9% versus horizontal 7.2%)
- Usage of the advanced heuristics may reduce the performance loss to 1.3% and 5.8%.
- Compared to results of a distributed RF our results are much better, there are two reasons for this: no extra copy operations needed and bypassing.

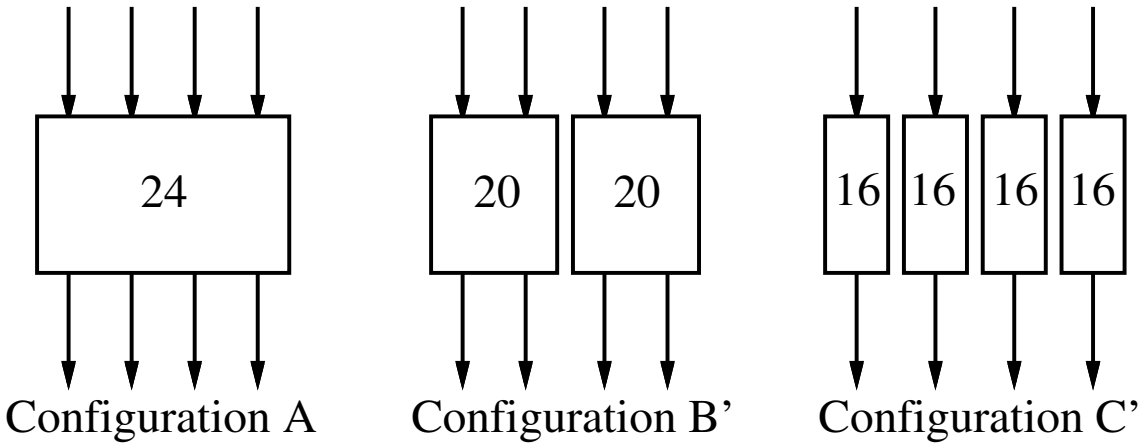
## Relative RF areas

- A partitioned RF occupies less chip-area than the corresponding monolithic RF.



## New RF configurations

$$Area(8\ ported, 24) \simeq 2 * Area(4\ ported, 20) \simeq 4 * Area(2\ ported, 16)$$



# Equal area compiling

## Equal area results

| **benchmark** | $h3(B')$ | $h3(C')$ |
|---|---|---|
| bison | 101.7% | 100.4% |
| compress | 101.5% | 105.3% |
| dhrystone | 96.6% | 103.6% |
| diff | 99.5% | 99.1% |
| djpeg | 98.1% | 101.6% |
| expand | 104.5% | 114.6% |
| flex | 98.3% | 99.0% |
| mpeg_play | 76.5% | 75.2% |
| od | 98.3% | 100.0% |
| sieve | 97.5% | 106.2% |
| sort | 95.0% | 97.7% |
| stanford | 97.5% | 100.0% |
| unexpand | 103.2% | 106.7% |
| uniq | 95.9% | 88.6% |
| wc | 106.2% | 105.8% |
| **average** | 98.0% | 100.3% |

Evaluation:

- Using equal area compiling the performance losses are negligible.

# Conclusions

- Partitioned RFs fit very well in TTAs
- RF partitioning for TTAs does not lead to a large performance loss
- A partitioned RF is easy to realize due to the low complexity of the components
- The required chip area may reduce
- Small RFs are faster which may result in an even better performance.
- For single ported RFs we expect that the results are even more promising