Advanced Topics: Explicitly Parallel Instruction Computing Architectures

Supplemental Notes

Yul Williams, D.Sc.

Goals

* At the conclusion of this session, the student should be able to:
* Identify the characteristics of EPIC computers
* Understand instruction scheduling challenges for complex computer architectures
* Understand the important issues involved in the domain of EPIC architectures
* Describe the purpose of region formation for parallel processing architectures

Outline

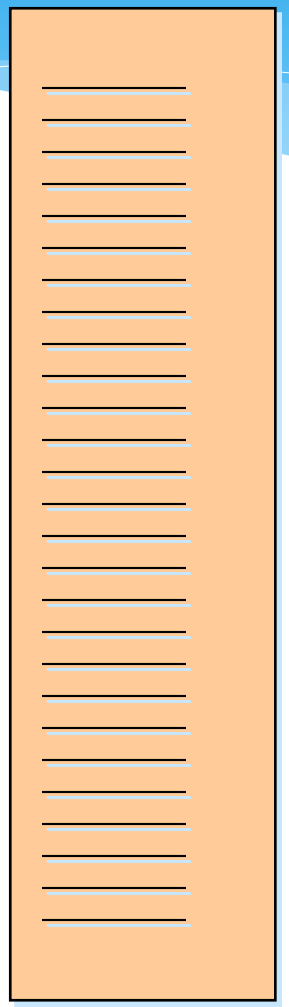
* Instruction Scheduling for EPIC Processors
* The Embedded Superblock
* ILP-Increasing Directives
* Summary

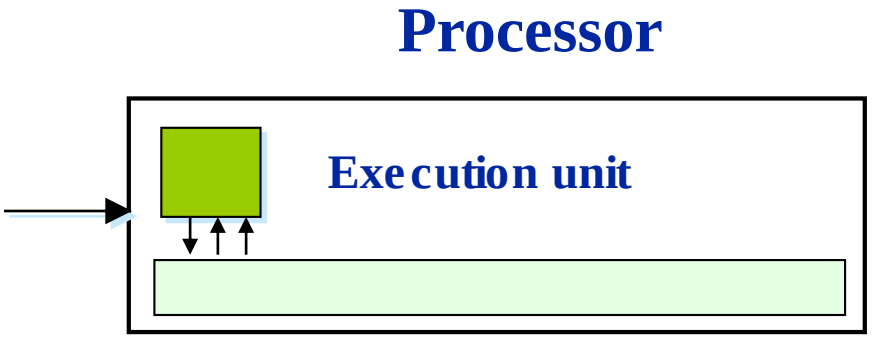
Explicitly Parallel Instruction Computing (EPIC) Architecture

* A computer architecture with many of the attributes of Very Long Instruction Word (VLIW) architectures.
* EPIC processors are very simplistically designed to take full advantage of pre-scheduled parallel instruction streams.
* Focuses on parallelizing code at the instruction-level, hence, it is referred to as an Instruction-Level Parallelism (ILP) technology.

Sequential Processor

Sequential Instructions





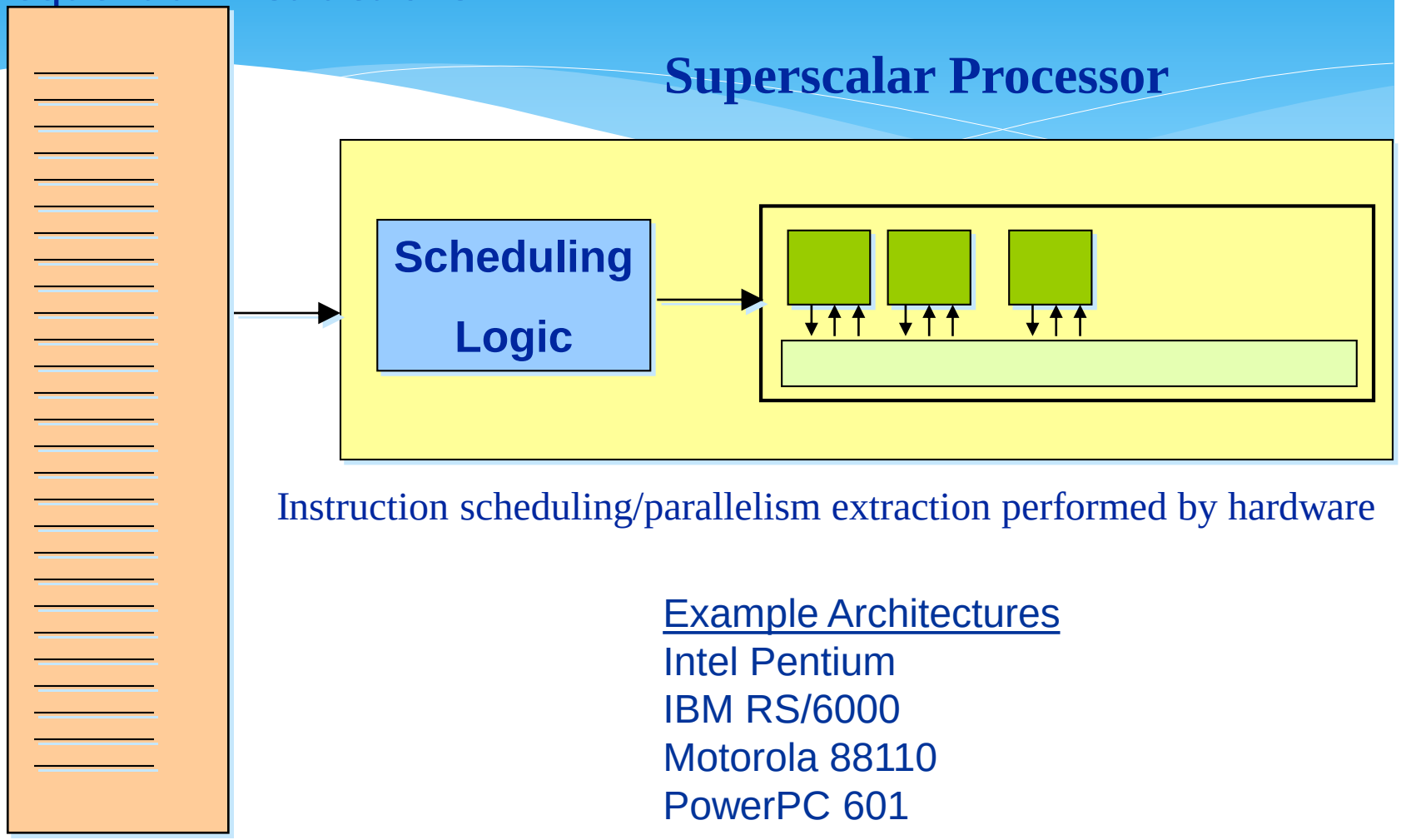
Example Architectures
Intel 8085
Motorola 6800

Sequential Processor Characteristics

* Serial instruction stream
* Multiple operations may access memory
* Small number of registers
* Arithmetic calculations involves an accumulator
* No parallelism supported

ILP Processors:Superscalar

Sequential Instructions



Instruction scheduling/parallelism extraction performed by hardware

Example Architectures
Intel Pentium
IBM RS/6000
Motorola 88110
PowerPC 601

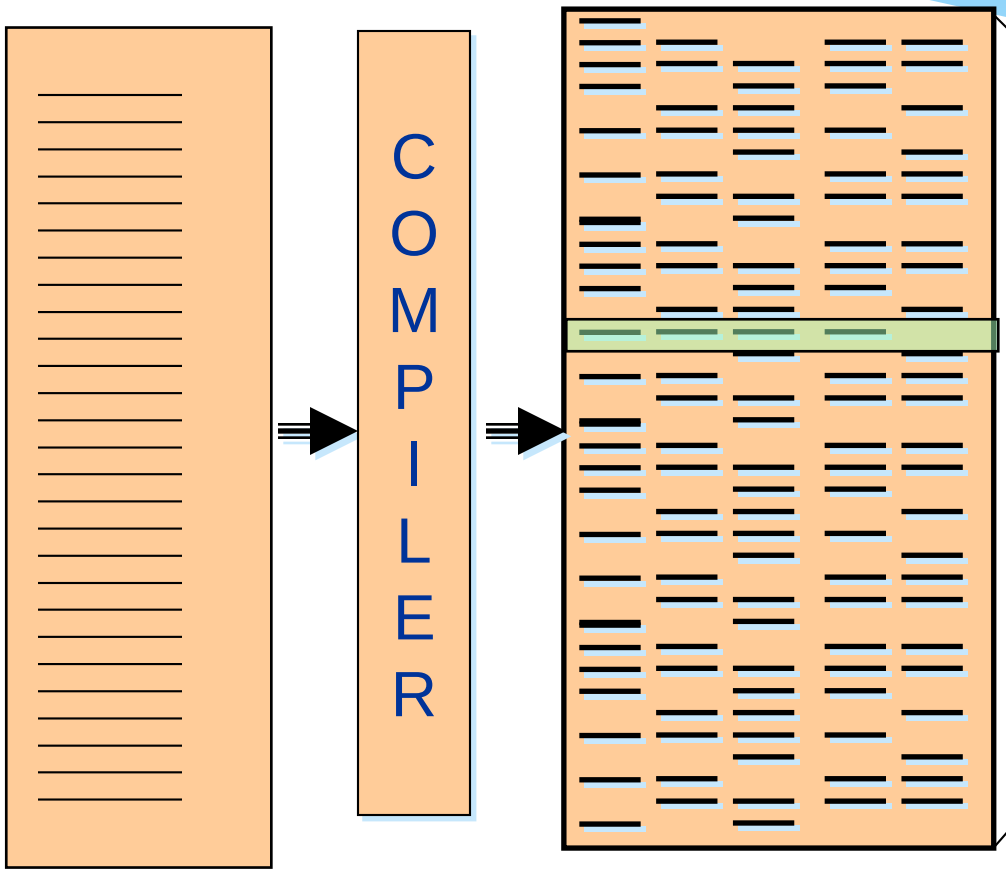
Superscalar Processor Characteristics

* Sequential instruction stream
* Complex on-chip instruction scheduling unit
  * Dependence resolution
  * Limited ILP extraction (based on "window size"). The window represents a small number of operations that will be examined and scheduled at a time.
* Large amount of chip real-estate is used by the instruction scheduling hardware.
* Increasing parallelism means expanding beyond the physical boundaries of the chip

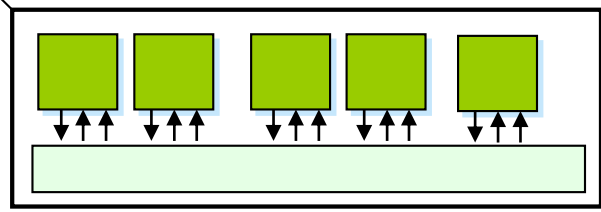
ILP Processors:EPIC

Serial Program

Compiled (Scheduled) Machine Code



EPIC Processor



Example Architectures

Intel Merced (IA-64) - due 2000

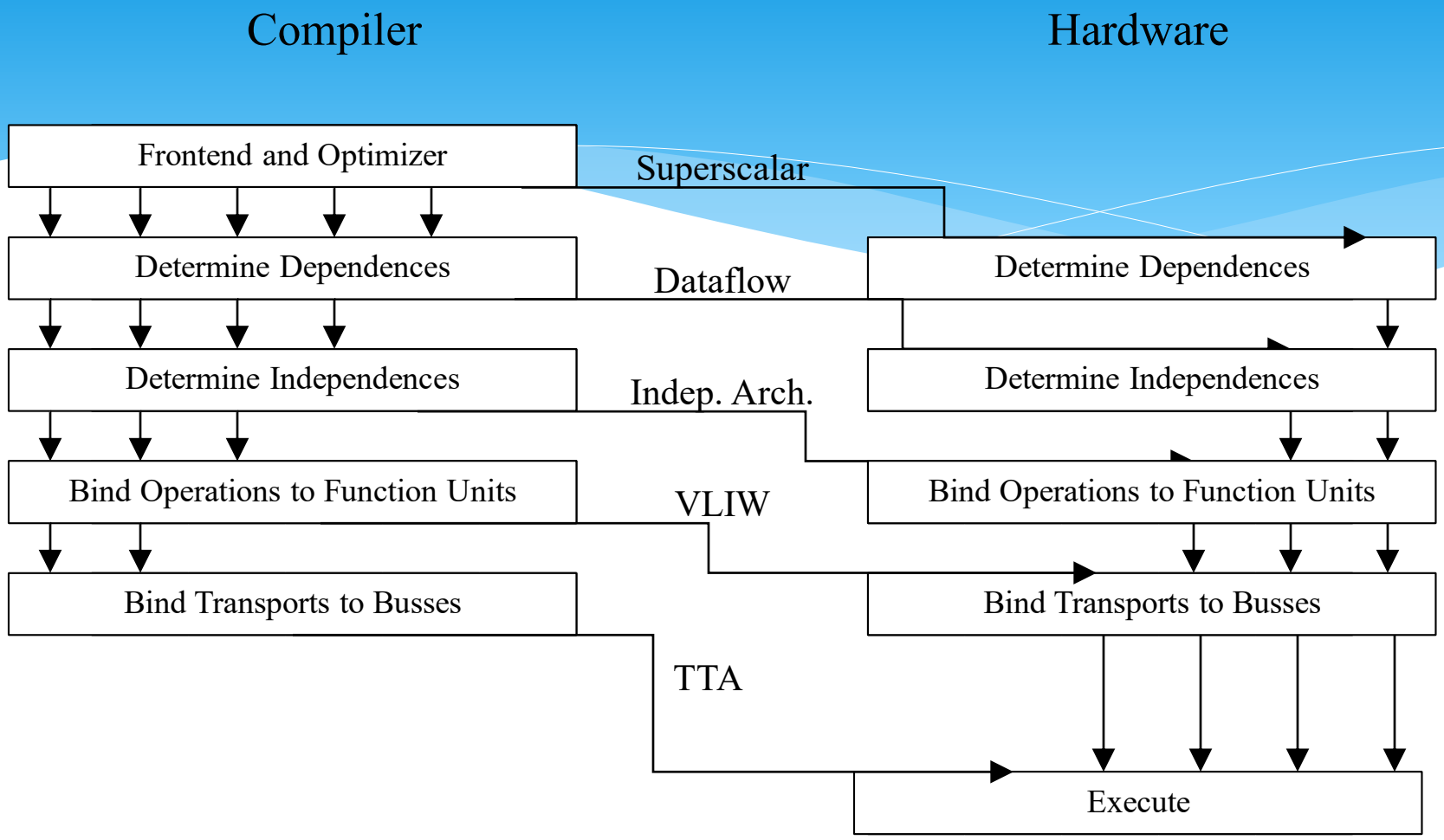
Philips Trimedia

TI TMS320C6X

EPIC/VLIW Processor Characteristics

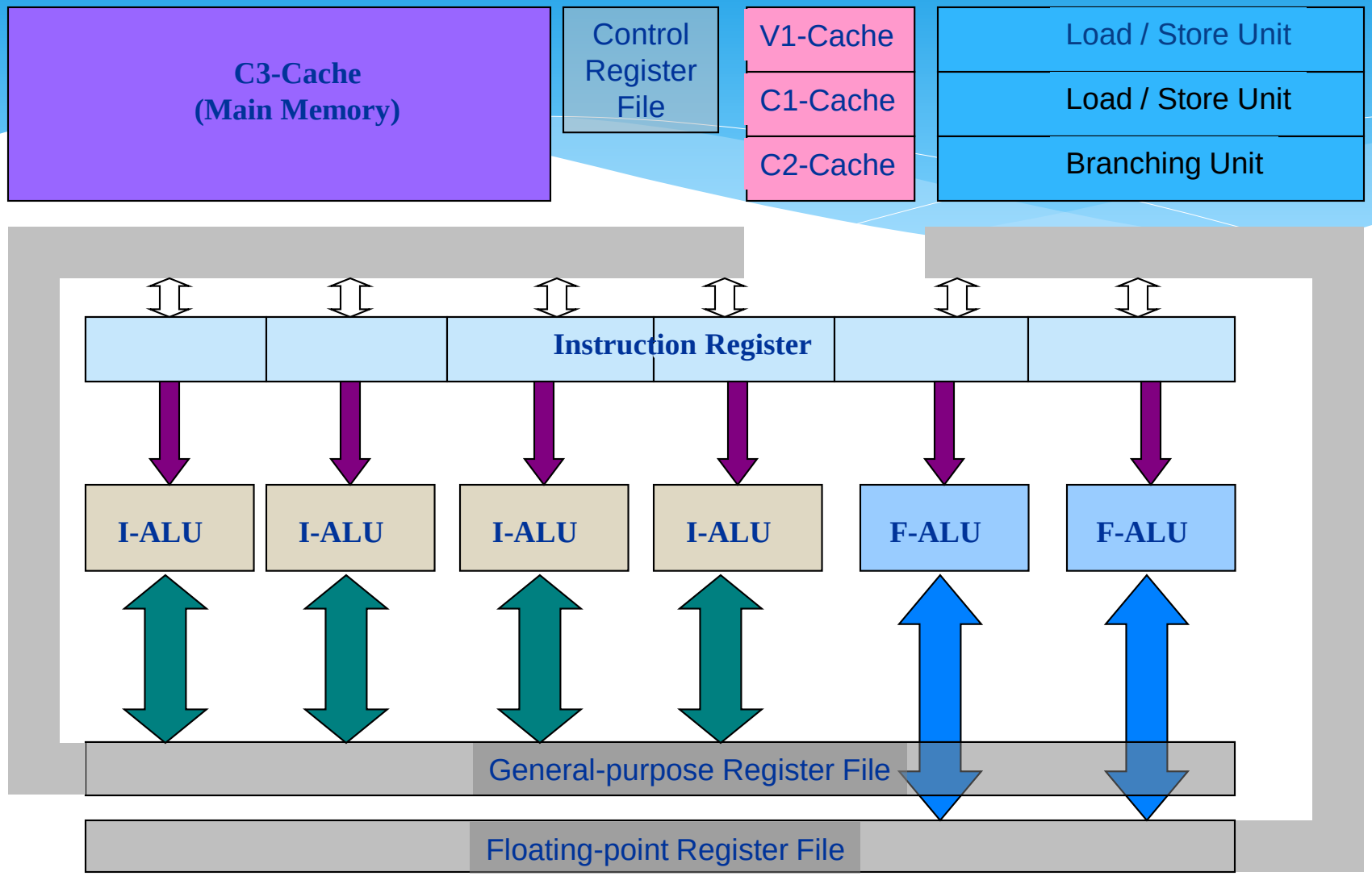
* Dependency-free parallel instruction stream
* Simplistic architecture (as compared to the superscalar example)
* Supports the expansion of wider-issue instruction word.
  * Dependence resolution and instruction scheduling is performed off-chip by the compiler
  * Unlimited “window size”
* EPIC Instruction
  * Multiple operation “slots”
  * Supplies independent operations parallel functional units

Compiler vs. Processor



B. Ramakrishna Rau and Joseph A. Fisher. Instruction-level parallel: History overview, and perspective. The Journal of Supercomputing, 7(1-2):9-50, May 1993.

Example Processor Architecture



EPIC Architectures: Key Points

* All optimizations and code scheduling is done off-chip in software.
  * More execution units may be placed on the chip
* Code arrives at processor interface in a dependency-free state.
* Performance is determined by the effectiveness of code generator in exploiting EPIC processor resources
* Larger "window" allows for more efficient use of processor resources

Instruction Scheduling for EPIC Processors

Region Formation

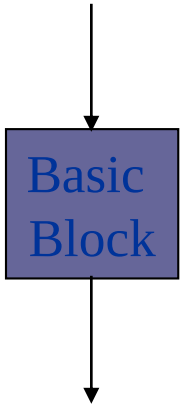
Background: Region Formation

* A region is a unit of program instructions that will be examined together for scheduling.
* The smallest region that will be used in this research effort is called the “basic block”
* There are two other types of regions that will be explored. They are both composed of basic blocks. They are the superblock and the hyperblock, respectively.

Background:Region Formation

The BasicBlock

```
        MOVE R1, 05
        MOVE R2, 10
        PBRR BTR1, Loop1
        MOVE R3, 20
Loop1:  MOVE R4, R1
        ADD R1, R2
        SUB  R3, 1
        CMPP R3, 0
        BRCT Loop1
```

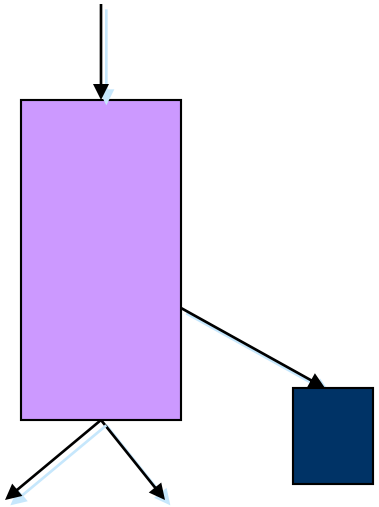


* "A basic block is a straight-line code sequence with no transfers in or out, except at the beginning or end." Hennessy and Patterson
* Simplest form of blocking strategies
* Offers least amount of ILP

Background:Region Formation

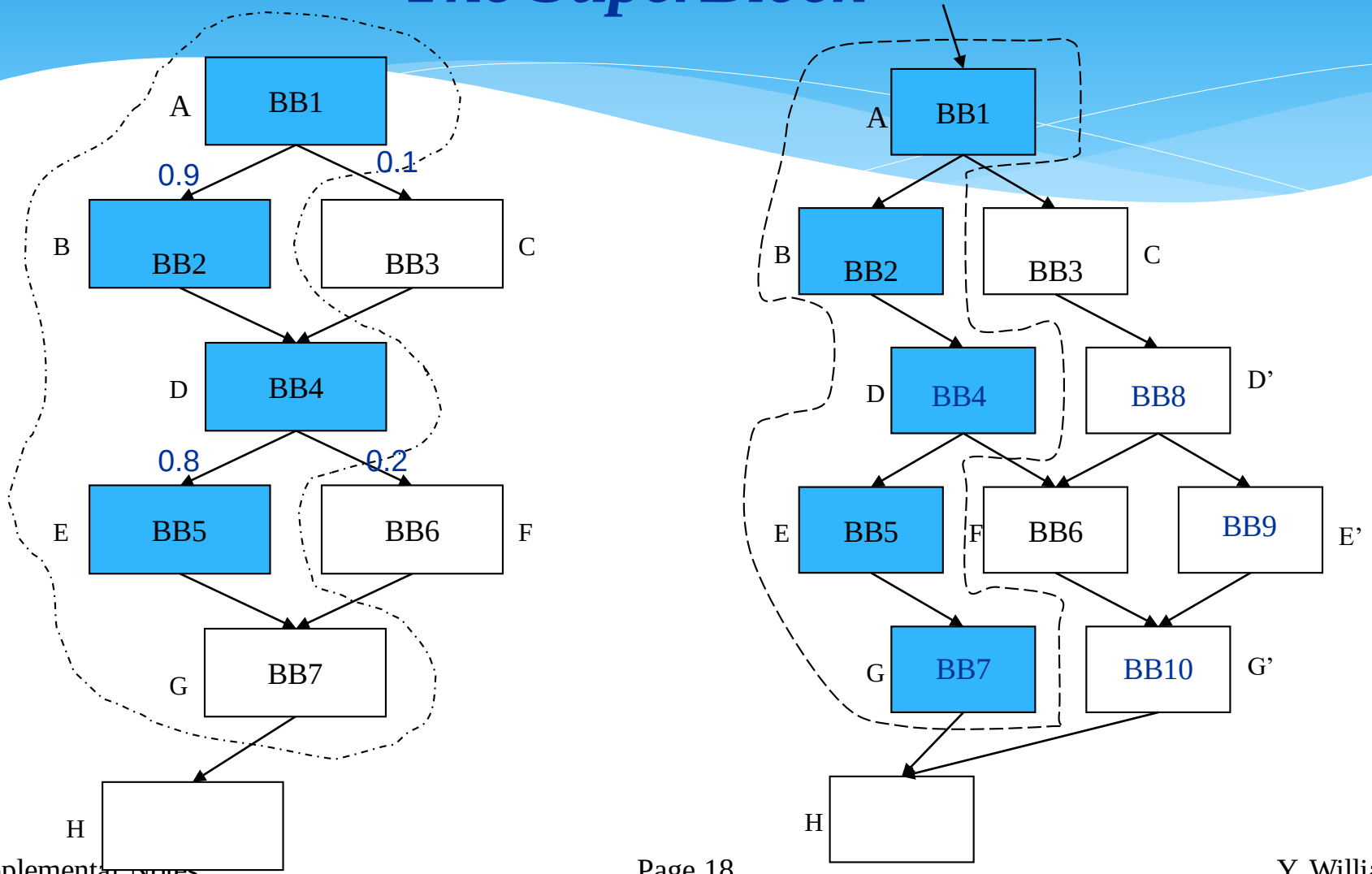
The SuperBlock

* The superblock is a trace composed of basic blocks with a single entry but potentially many exits
* Superblock formation is done in two steps
  * Trace selection
  * Tail duplication



Background: Region Formation

The SuperBlock



Background: Region Formation

The HyperBlock

* Single entry/ multiple exit set of predicated basic blocks (if-conversion)
* There are no incoming control flow arcs from outside basic blocks to the selected blocks other than the entry block
* Nested inner loops inside the selected blocks are not allowed
* Hyperblock formation procedure:
  * Trace selection
  * Tail duplication
  * Loop peeling
  * Node splitting
  * If-conversion

Background: Region Formation

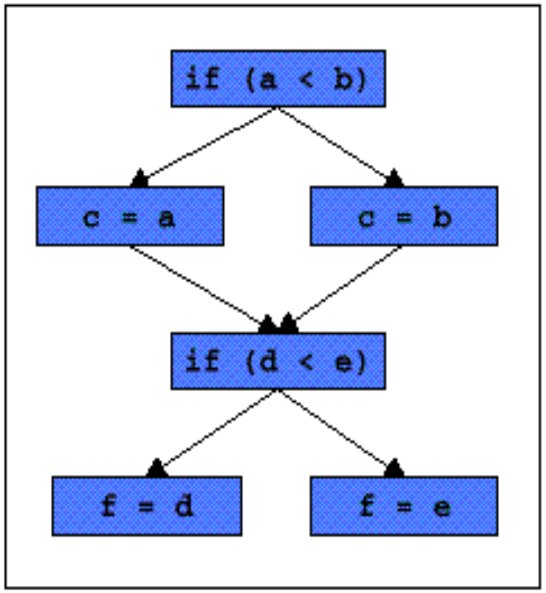
If-Conversion Example

If-conversion replaces conditional branches with predicated operations.

- Those instructions in branches not taken are disabled by predication.

For example, the code generated for:

```
if (a < b)
     c = a;
else
     c = b;
if (d < e)
     f = d;
else
     f = e;
```

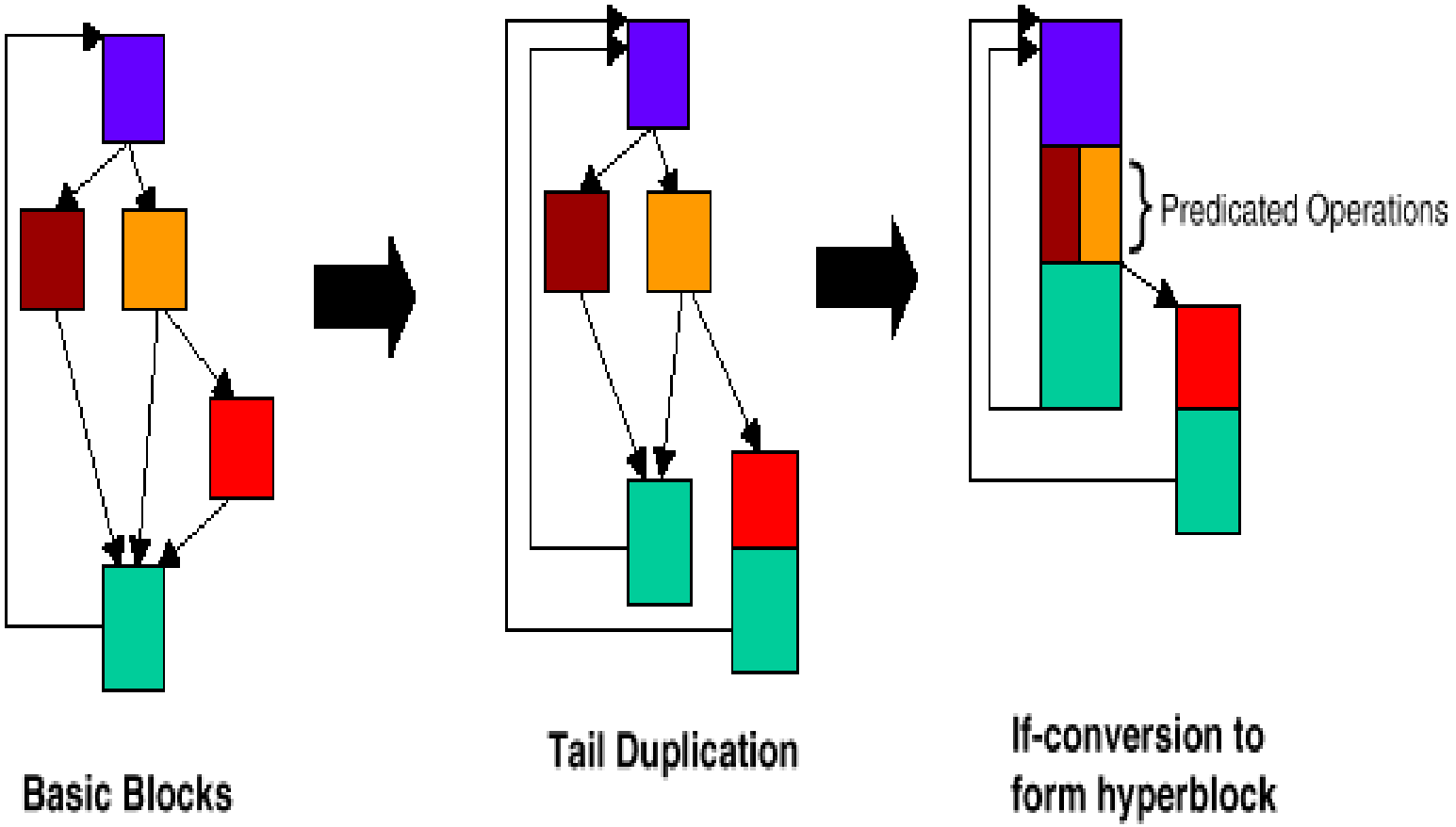


might be the two VLIW instructions:

`P1 = CMPP.< a,b`	`P2 = CMPP.>= a,b`	`P3 = CMPP.< d,e`	`P4 = CMPP.>= d,e`
`c = a if p1`	`c = b if p2`	`F = d if p3`	`F = e if p4`

Background: Region Formation

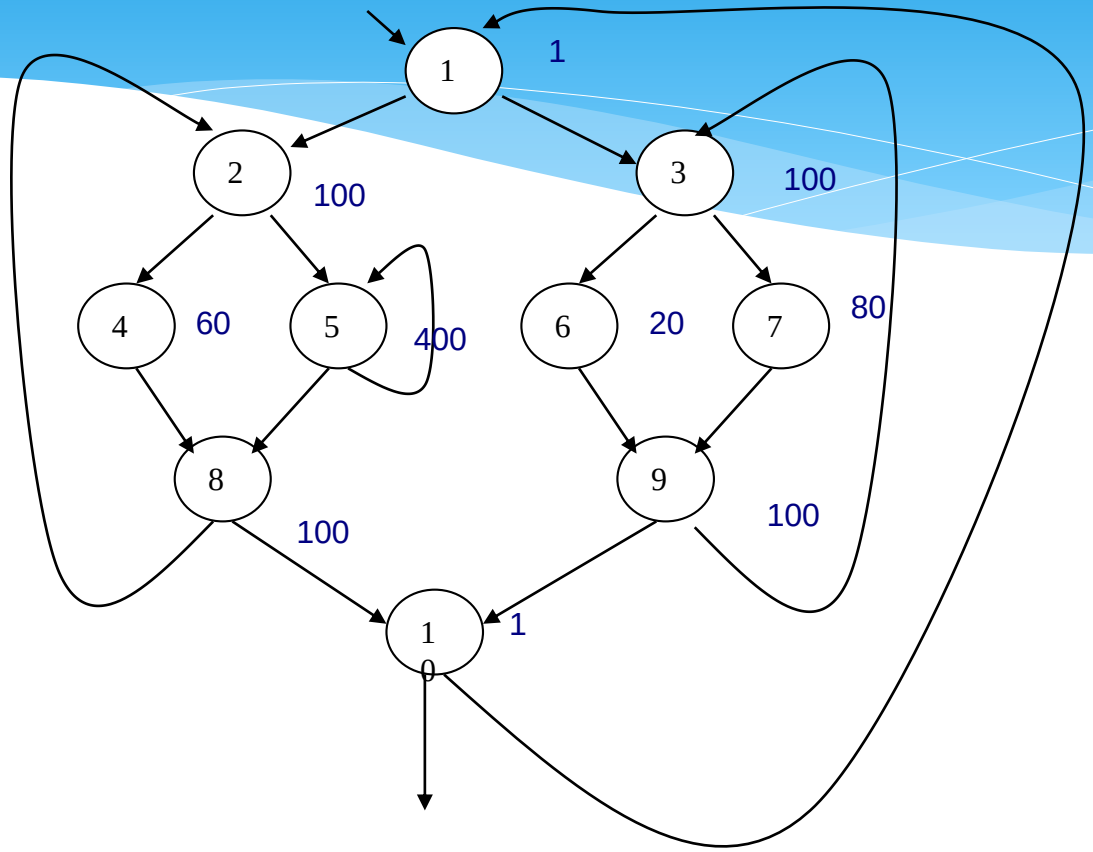
The HyperBlock



Background: Region Formation

* In general, the opportunity to extract more parallelism increases as the region size increases. There are more instructions exposed in the larger region size.
* The compile/assemble time increases as the region size increases. A trade-off in compile/assemble time versus run-time must be considered.

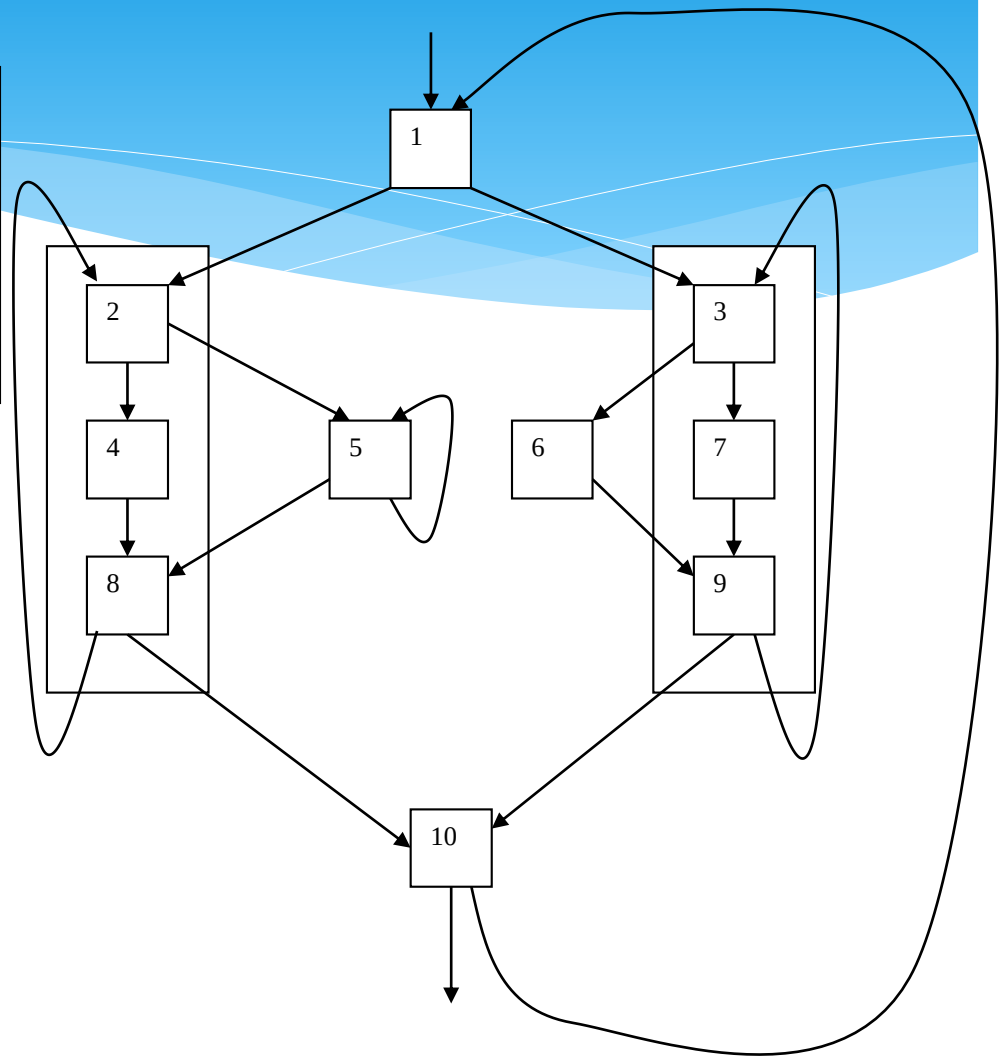
Superblock: Control Flow Graph



Original source graph.

Superblock: Trace Formation

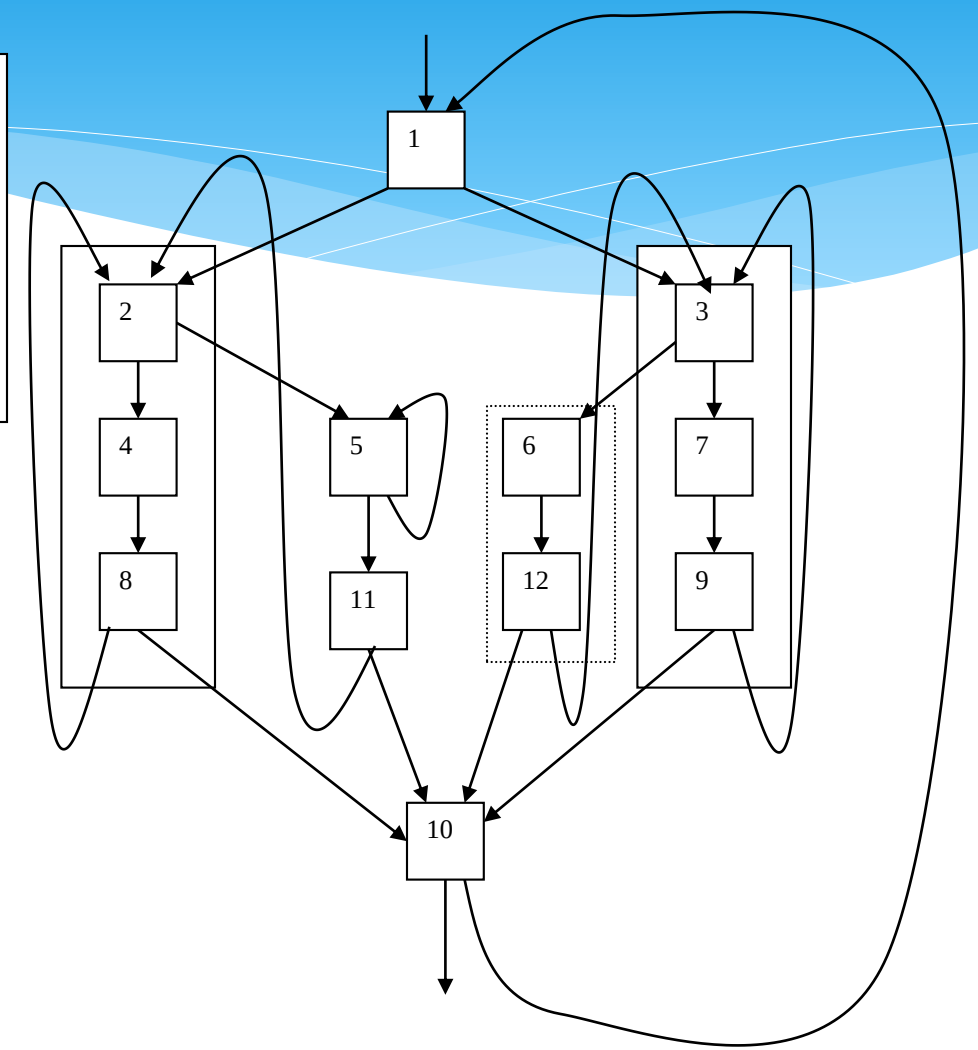
The CFG is constructed hierarchically. The traces contain basic blocks that contain operations. Also note that there are edges going into the trace boundaries. Superblocks do not allow side entry edges [Hwu and Mahlke, 92].



Superblock: Tail Duplication

Note that with tail-duplication, there are no more in-edges going into the trace boundaries. There is the disadvantage, however, of code expansion. This is an important aspect when it comes to generating code for restricted resource embedded systems.

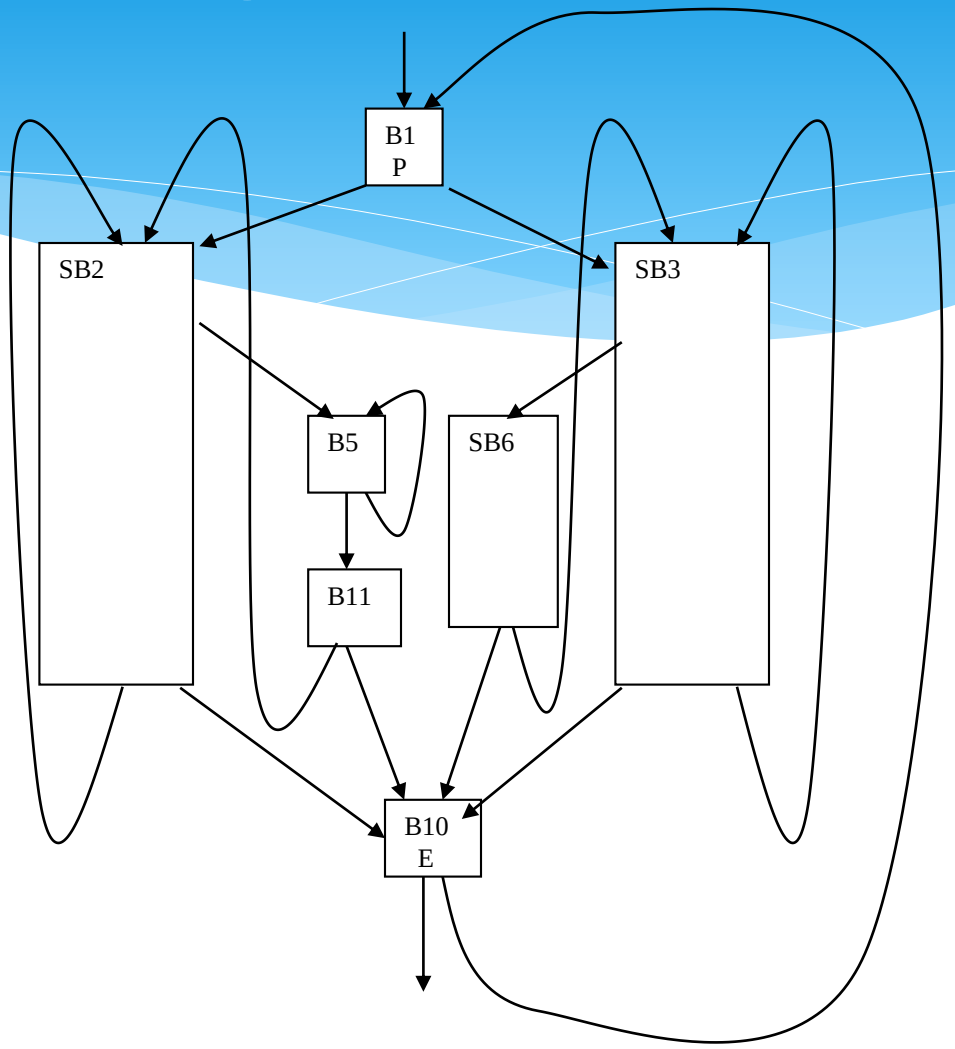
Nodes 11 and 12 are copies of nodes 8 and 9, respectively.



Superblock: Region Production

Note that all operations are copied from the contained BBs and placed into the SB structure at the same level. With this technique, the scheduler views all operations at a single level hierarchy and is able to produce an optimal schedule.

The graph shown illustrates the results from the SB formation algorithm.



The Embedded Superblock

For

Resource-Constrained/High-Performance Embedded Systems

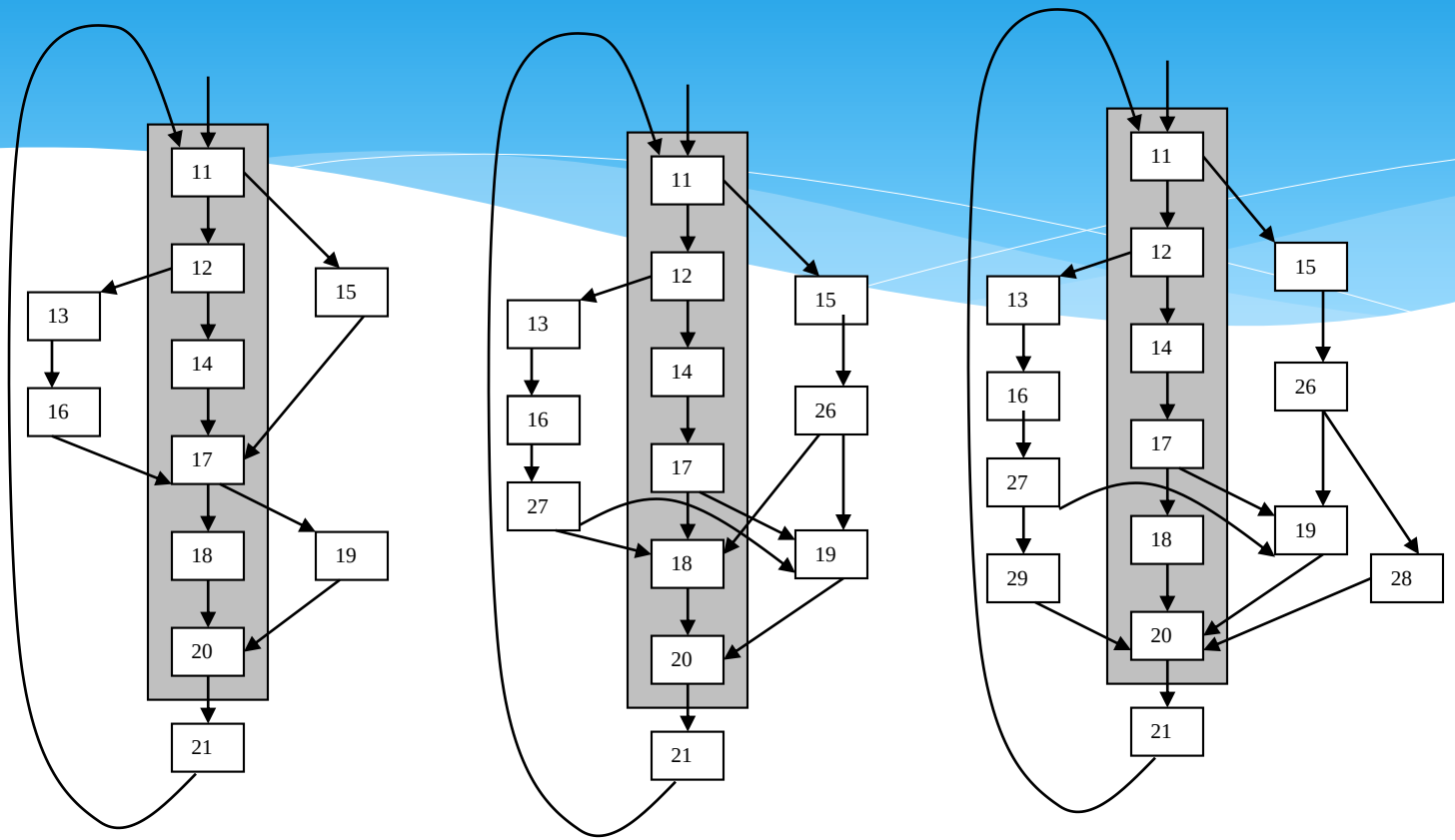
The Embedded Superblock

* Larger region production increases the size of generated code. This requires an increase in system memory.
* Some embedded systems may not have the capacity to accept such large code units.
* A trade-off may be made between code size and ILP
* The embedded superblock maintains code functionality while being sensitive to the restricted resources of the embedded systems class platform

Embedded Systems Traits

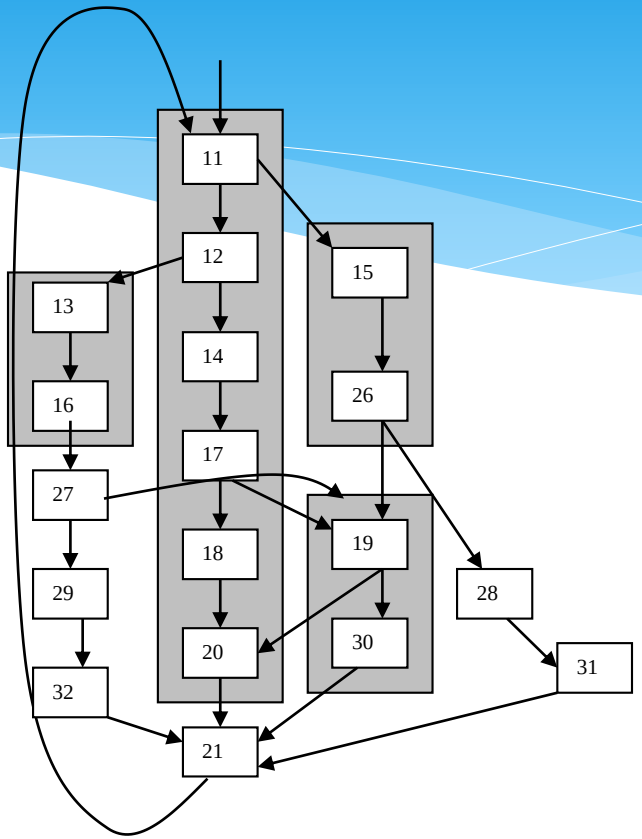
* Usually smaller than a general-purpose platform
* Non-conventional platform
* No general-purpose compilation tools
* Limited on-board resources
* Range of performed tasks is limited to a specific set of functions (based on its intended application)
* Operation may be constrained by a time bound

Superblock Formation: Unconstrained



Given an initial sub-graph, the SB formation takes place in above steps. Both the number of nodes and the number of edges increases during the tail-duplication process.

Superblock Formation: Unconstrained

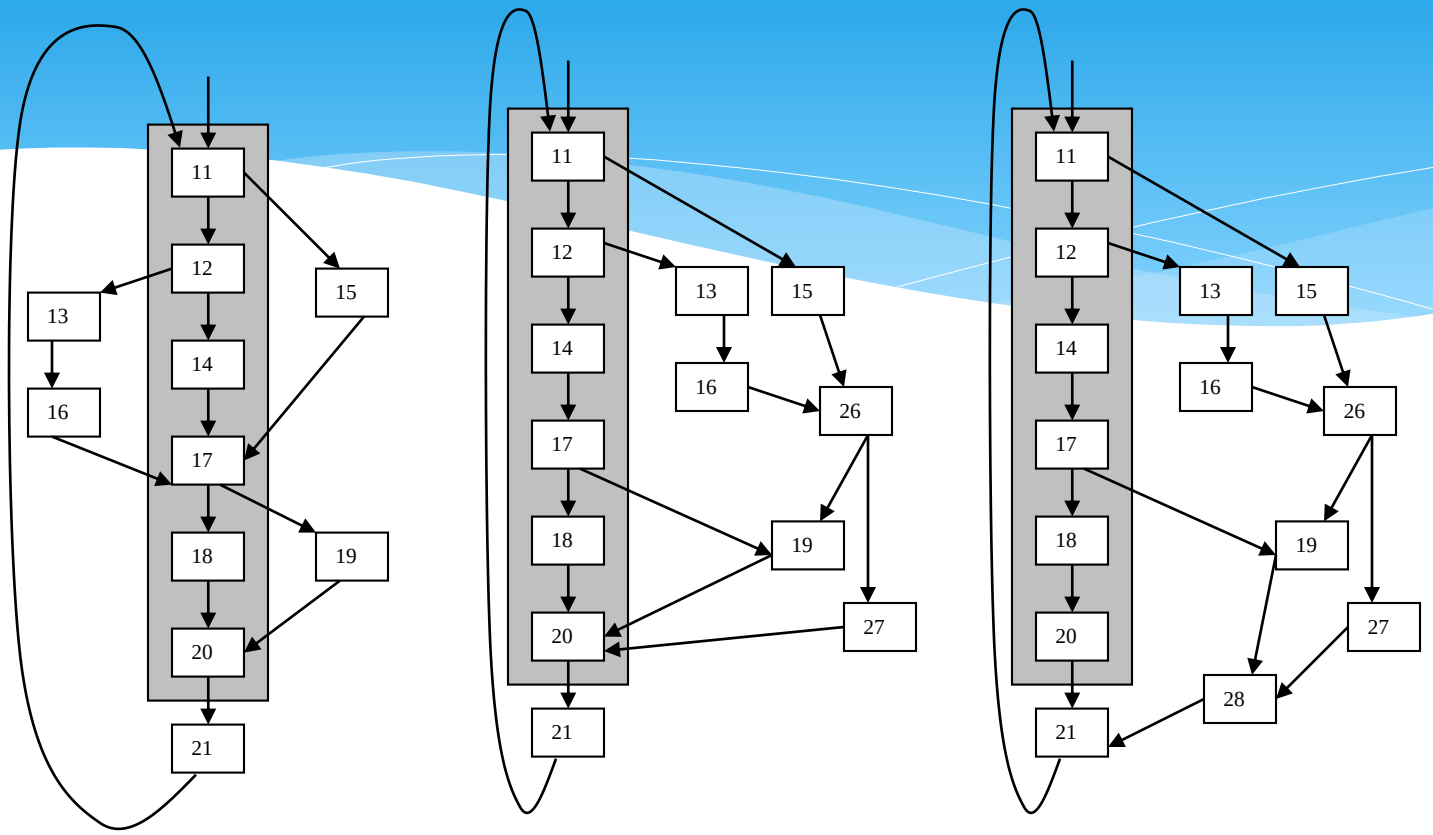


Since the SB formation process is unconstrained, original trace nodes may be duplicated more than once. For instance, nodes 26 and 27 are duplicates of node 17. Nodes 28 and 29 are duplicates of node 18. Nodes 30, 31 and 32 are duplicates of node 20.

The Embedded Superblock

* Maximum Configuration
  * Produces largest amount of code
  * Unconstrained code growth caused by multiple copies of trace nodes
* Minimum Configuration
  * Produces the smallest amount of code
  * Employs code re-use
  * Maximally-constrained code growth using the concept of “equivalence nodes”

Embedded Superblock: Constrained



Using “equivalence nodes (EQN)”, the expansion of the ESB is maximally constrained. Node 26 is an EQN of node 17. Node 28 is an EQN of node 18. Node 28 is an EQN of node 20.

Expansion Factor = Summation {1 to (Trace_height - 1)} [(number targeted trace nodes)]

Observations: Region Formation

* Larger regions may be constructed from assembly language source that expose more ILP and reduce program schedule
* When ESB are formed, a trade-off of reduced ILP must be made if the ESB is not equal in size to the unconstrained SB.
* Since the input of the region formation module is IR code, the source may be generated by either an assembler or an optimizing compiler!!

ILP-Increasing Directives

Factored Loop Unrolling

Matrix Multiplication (MATMUL) Example

Loop Unrolling

* Advantages:

-Reduces Cost of Jump.

-Increases Block size to take advantage of exploiting parallelism across successive iterations of the loop.

Eg:

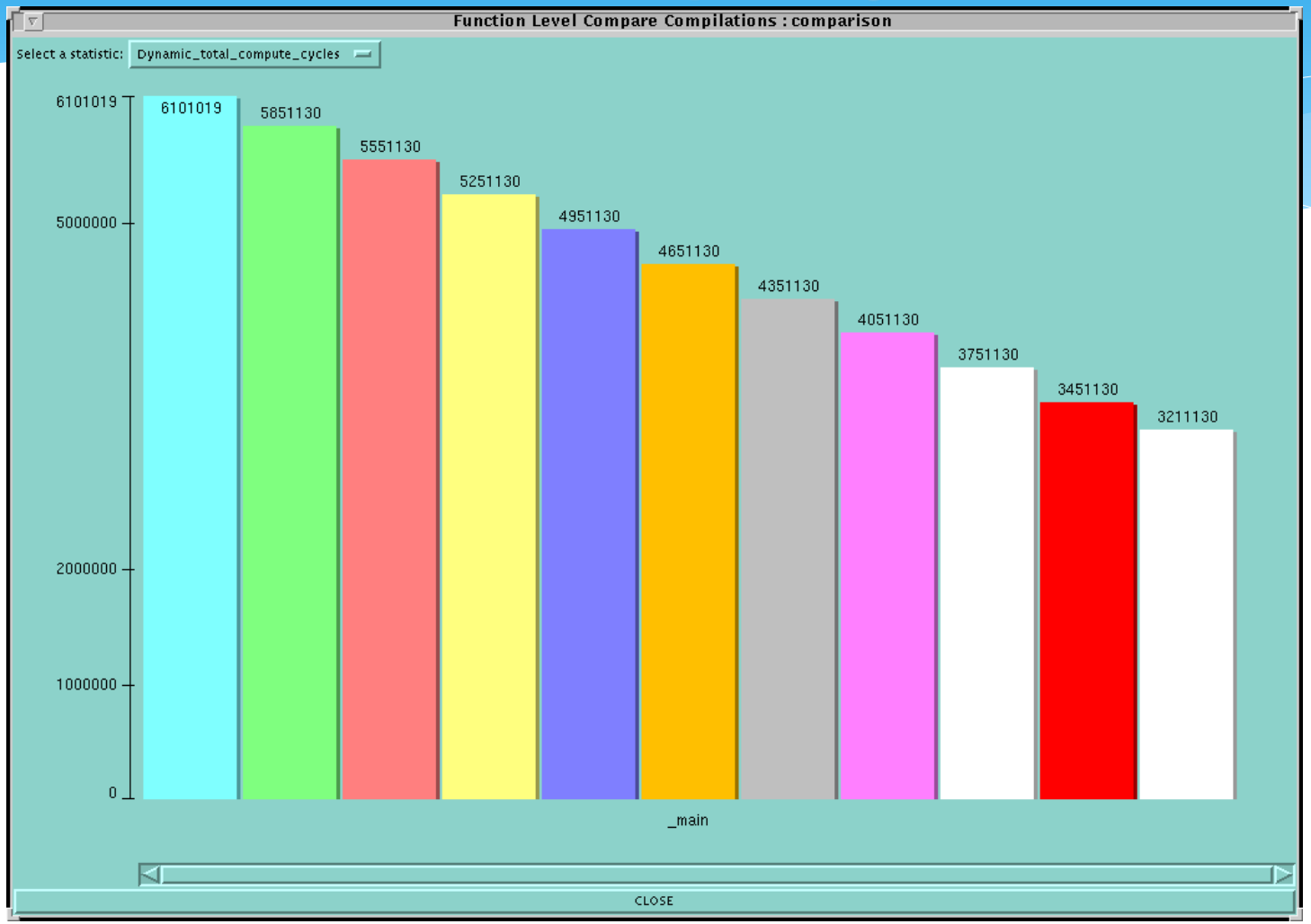
```
loop: add r2=r2,r1
      inc r2,4
      inc r1,4
       decr r3
      bnz loop
```

Eg:

```
loop: add r2=r2,r1
      inc r2,4
      inc r1,4
       decr r3
       add r2=r2,r1
      inc r2,4
      inc r1,4
       decr r3
      bnz loop
```

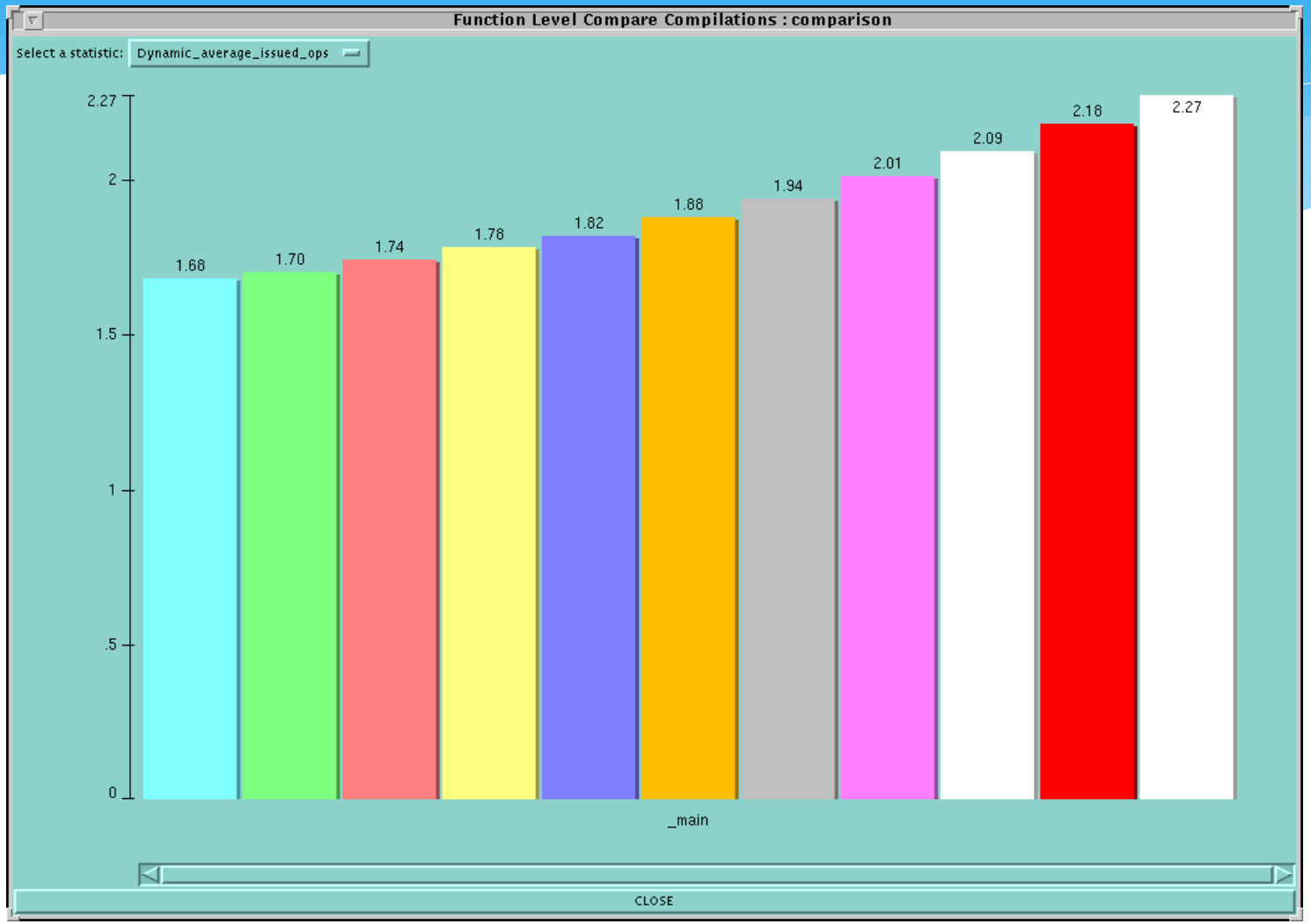
MATMUL

Decrease in total Compute Cycles = 48 %

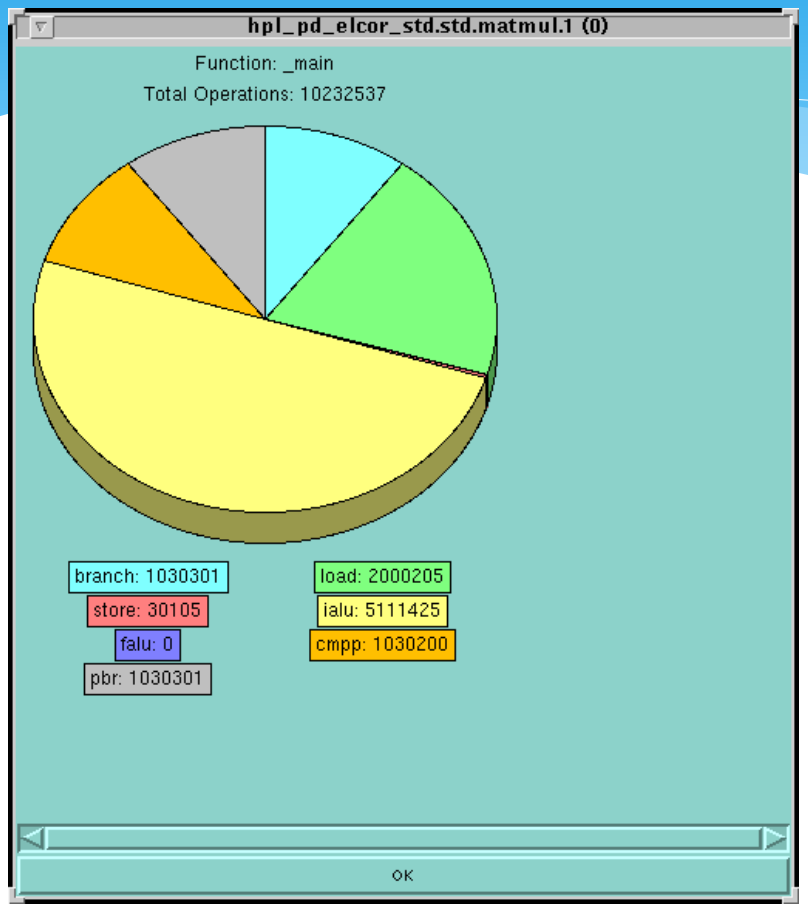


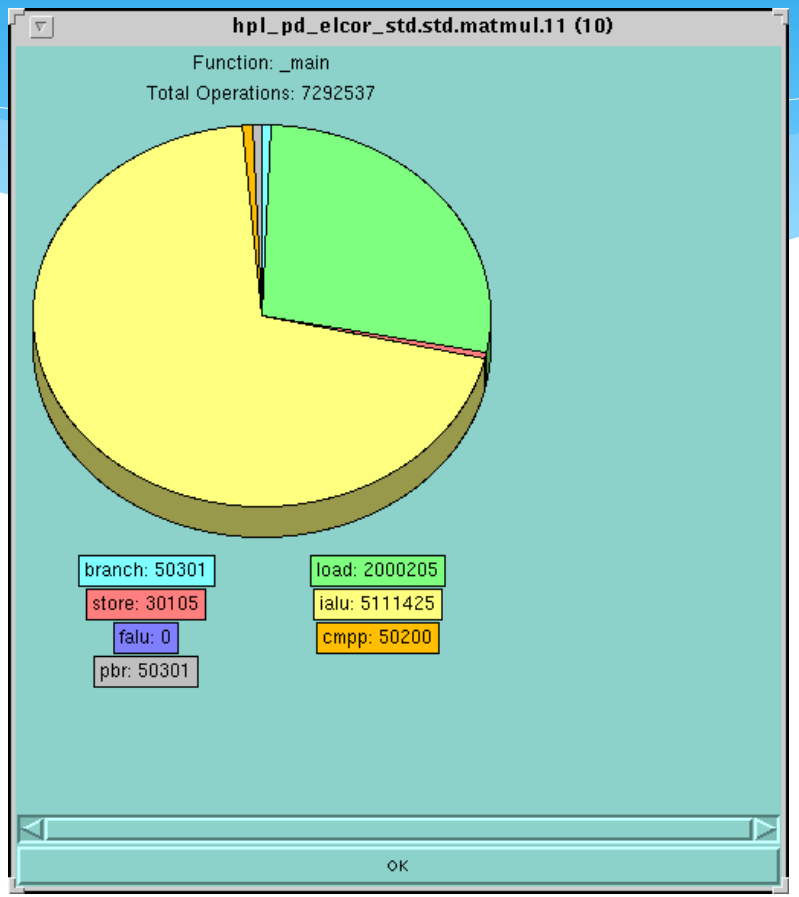
MATMUL

Increase in ILP = 35 %



MATMUL


hpl_pd_elcor_std.std.matmul.1 (0)
Function: _main
Total Operations: 10232537
branch: 1030301
store: 30105
falu: 0
pbr: 1030301
load: 2000205
ialu: 5111425
cmpp: 1030200
OK


hpl_pd_elcor_std.std.matmul.11 (10)
Function: _main
Total Operations: 7292537
branch: 50301
store: 30105
falu: 0
pbr: 50301
load: 2000205
ialu: 5111425
cmpp: 50200
OK

Loop Unrolling: Observations

* As loops are unrolled, the number of required cycles decreases due to reduction in comparison and branch operations.
* Unrolling of loops frees functional units and potentially reduces the schedule length
* There are limitations to schedule reduction using the loop unrolling optimization
* The number of independent computations in the loop body affects the scheduling characteristic of the application

Strength Reduction

The Strength Reduction Directive

```
        .Strength_Reduce
        MOVE R1, 05          ; R1 = 5                                  Latency = 0
        MOVE R2, 10          ; R2 = 10                                 Latency = 0
        MOVE R5, 0           ; Comparison value                        Latency = 0
Loop1:  MOVE R3, 20          ; R3 = 20, Loop counter variable          Latency = 0
        MOVE R4, R1          ; R4 = R1                                 Latency = 0
        MPY R4, R1, 16       ; R4 = R1 * 16                            Latency = 3
        SUB R3, 1            ; Decrement loop counter                  Latency = 1
        ADD R5, 1            ; Increment comparison value              Latency = 1
        DIV R4, R4, 2        ;R4 = R4 / 2                              Latency = 8
        PBRR BTR1, Loop1     ; Load branch target register             Latency = 0
        CMPP R3, R5          ; Finished?                               Latency = 0
        BRCT (BTR1)          ; Continue looping if counter is positive Latency = 1
```

Loop latency = 14

Optimization Performed:

```
        MOVE R1, 05          ; R1 = 5                                  Latency = 0
        MOVE R2, 10          ; R2 = 10                                 Latency = 0
        MOVE R5, 0           ; Comparison value                        Latency = 0
Loop1:  MOVE R3, 20          ; R3 = 20, Loop counter variable          Latency = 0
        MOVE R4, R1          ; R4 = R1                                 Latency = 0
        SHL R4, 4            ; R4 = R1 * 16                            Latency = 1
        SUB R3, 1            ; Decrement loop counter                  Latency = 1
        ADD R5, 1            ; Increment comparison value              Latency = 1
        SHR R4, 1            ;R4 = R4 / 2                              Latency = 1
        PBRR BTR1, Loop1     ; Load branch target register             Latency = 0
        CMPP R3, R5          ; Finished?                               Latency = 0
        BRCT (BTR1)          ; Continue looping if counter is positive Latency = 1
```

Loop latency = 5

Register Virtualization

Register Virtualization

* Removal of WAR and WAW data dependencies
* The default register allocation mode is assumed to be “on”.
* The register allocation mode must be set to “off” in order to preserve the bindings set by the programmer.
* Consequences:
  * Generates more independent operations
  * enhances ILP
  * reduces schedule length

Summary

* EPIC architectures are recent to the computer architecture landscape
* They offer great challenges for hardware designers and compiler and debug tool developers
* Many new server-class computer systems are based on the EPIC architecture