

Automatic Data Path Generation from C Code for Custom Processors

Jelena Trajkovic and Daniel D. Gajski
{jelenat, gajski}@cecs.uci.edu

Center for Embedded Computer Systems
University of California, Irvine



Outline

- **Introduction and Problem Definition**
- **Data Path Extraction from C Code**
- **Experimental Results**
- **Conclusion and Future Directions**



Introduction

1. First architecture then code (A→C)

- a. No control over 3rd party or legacy code
- b. Compiler must *discover* ways to match code with architecture
- X Best architectural features may not exist
- X Unnecessary architectural features will go unused

2. First code then architecture (C→A)

- a. Include architectural features that optimize code execution
- b. Exclude features that cannot be utilized by the application
- ✓ Best performance possible for given C code
- ✓ Best architecture utilization for given performance goal



Problem Definition

- **What is the best data path to execute given C code?**
 - “The best” architecture is one that meets given constraints
 - Manual design
 - Not scalable for large application
 - Extracting data path from C code
 - Set of properties of C code
 - Set of HW components, structures/templates
 - Rules for deriving HW net-list from C code properties
- **Goals**
 - Automate data path extraction
 - Constraint: performance
 - Optimization goal: utilization
 - Ability to handle ~10000 lines of code
 - Controllability of data path design



Outline

- **Introduction and Problem Definition**
- **Data Path Extraction from C Code**
- **Experimental Results**
- **Conclusion and Future Directions**



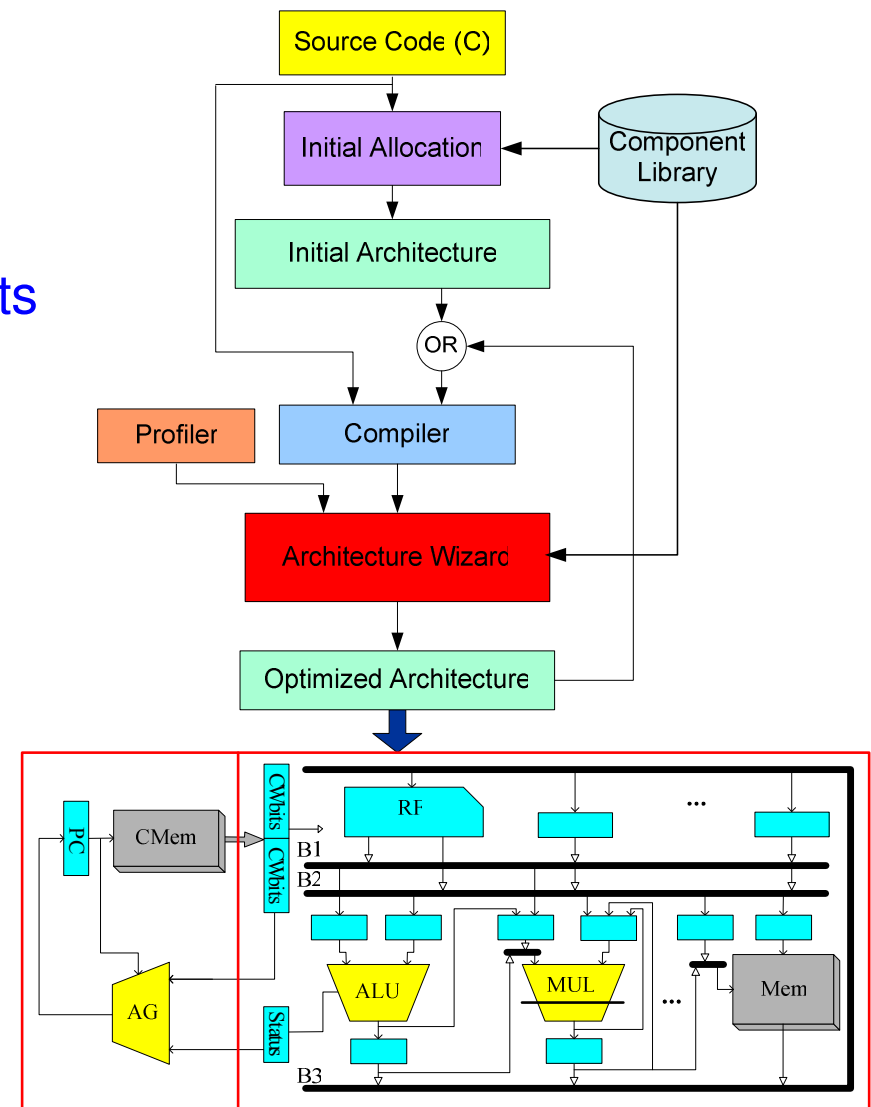
Data Path Extraction from C - Overview

1. Initial Allocation

- Minimum execution time
- Constrained by available components (functional units, storage units, buses...)

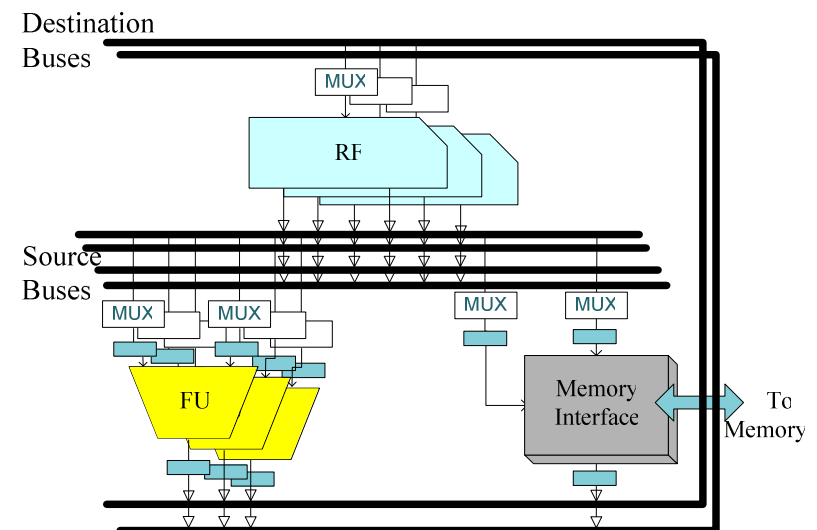
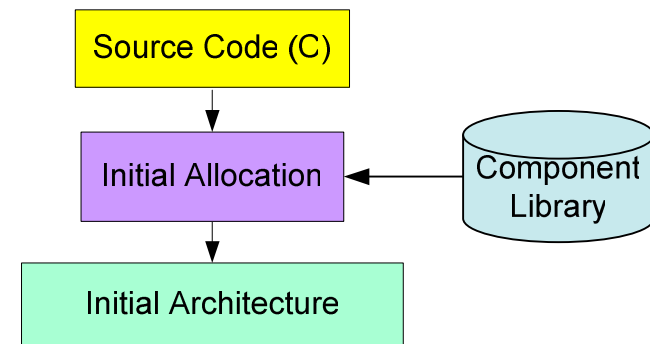
2. Datapath Optimization

- Satisfy performance constraint and maximizing utilization
- Critical code selection
- DP estimation and refinement



Initial Allocation and Component Selection

- **Define C code properties**
 - From ALAP-like schedule
 - Statistics for operators and data transfer
- **Use heuristics to map properties to components/connections**
 1. Most general FU for given operation
 2. Greedy interconnect allocation
 3. Pipelined data path design



Architecture Wizard - Overview

- **Goal of Phase II**

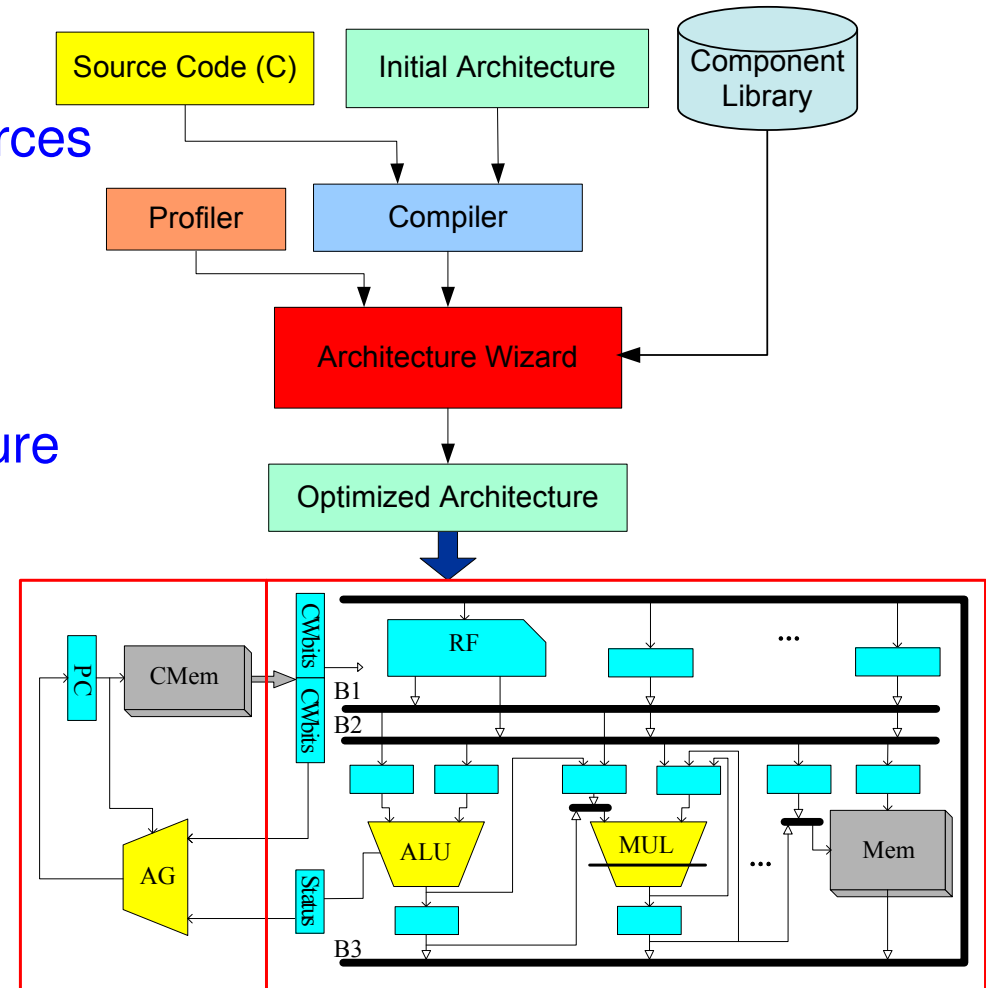
- Reducing number of used resources
- Under performance constraints

- **Inputs:**

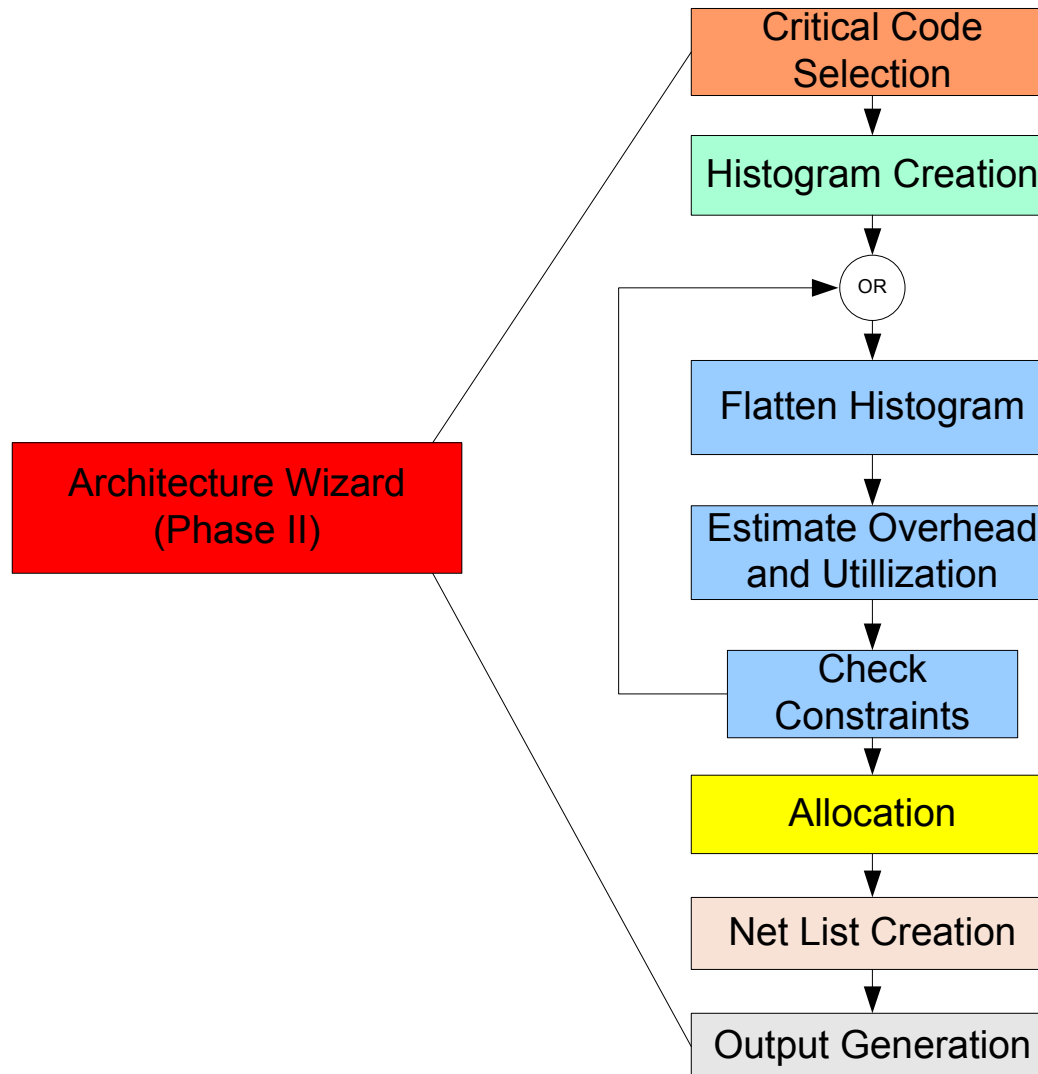
- Schedule for the Initial Architecture
- Execution frequencies (Profiler)
- **Performance** constraints and **utilization** goal (Designer)
- Component Library

- **Outputs:**

- Architecture Net-List

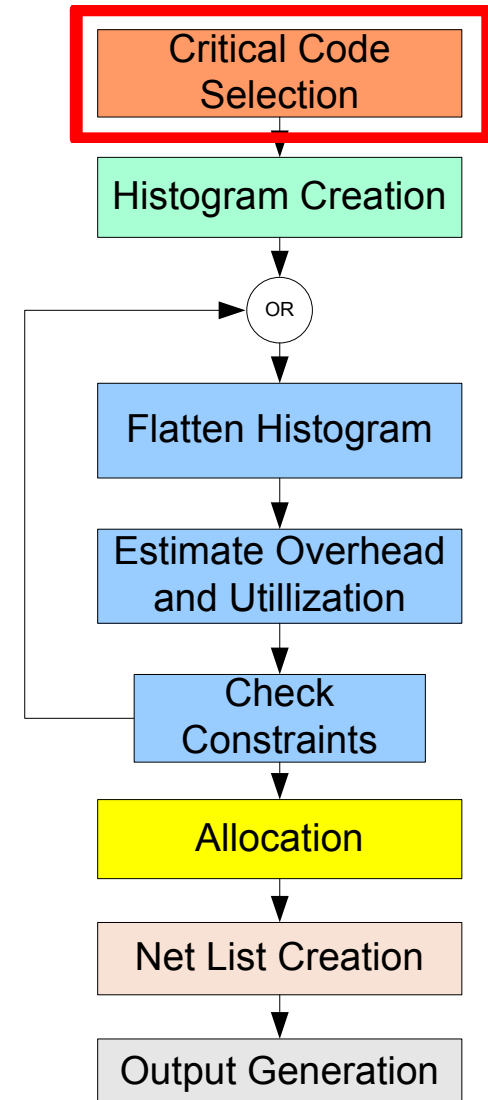


DP Creation and Evaluation



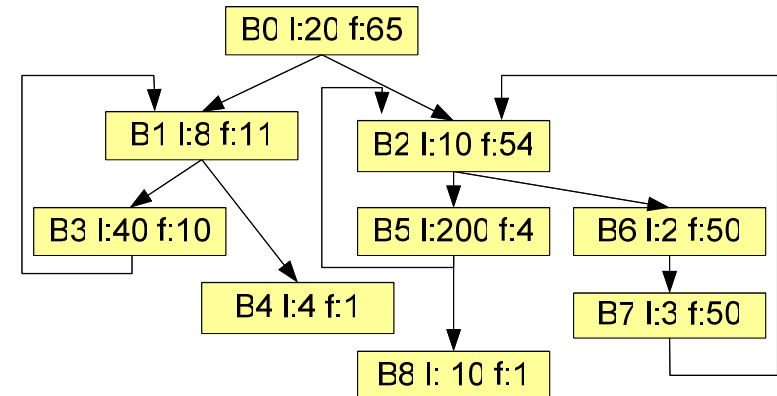
DP Creation and Evaluation

1. **Select basic blocks as function of frequency and length**
2. **Derive component usage/cycle (histogram) from schedule**
3. **For each component**
 - a. Recalculate number of instances
 - b. Estimate overhead and utilization using histogram
 - c. Repeat a. – c. until constraints satisfied
4. **Allocate updated number of units and create net-list**



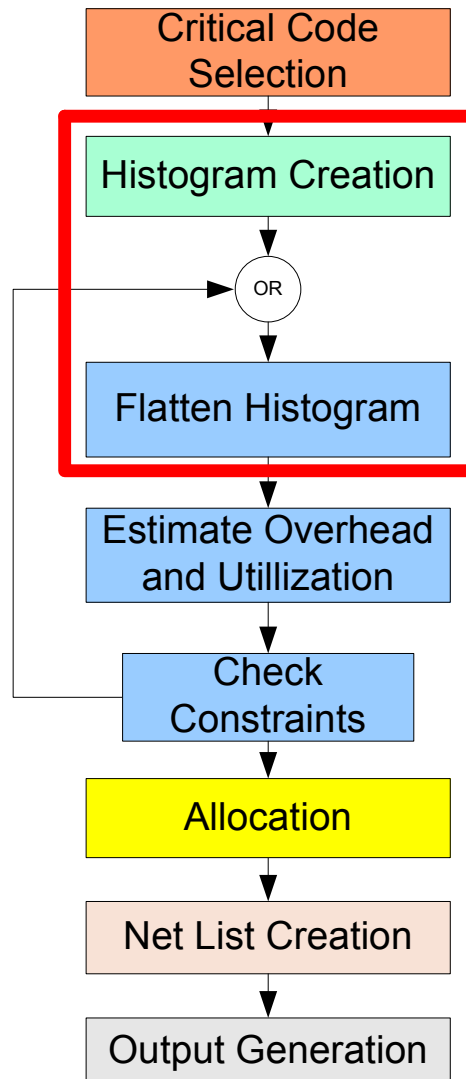
Critical Code Selection

- **Critical Code:**
 - BBs that contribute the most to the execution time
- **Every BB annotated with**
 - f = frequency, l = length
- **Select BB for optimization with**
 - High length
 - High frequency
 - High frequency-length product
- **Designer specifies cutoff**
 - Using parameters P_l , P_f , P_{fl}



- **e.g.**
 - $P_l = 0.5$, $P_f = 0.8$, $P_{fl} = 0.1$
- l_i : **B5**, B3, B0, B2, B8, B1, B4, B7, B6
- f_i : **B0**, B2, B6, B7, B1, B3, B5, B4, B8
- $f_i \cdot l_i$: **B0**, **B5**, **B2**, **B3**, B7, B6, B1, B8, B4
- **Selected: B0, B5, B2, B3**

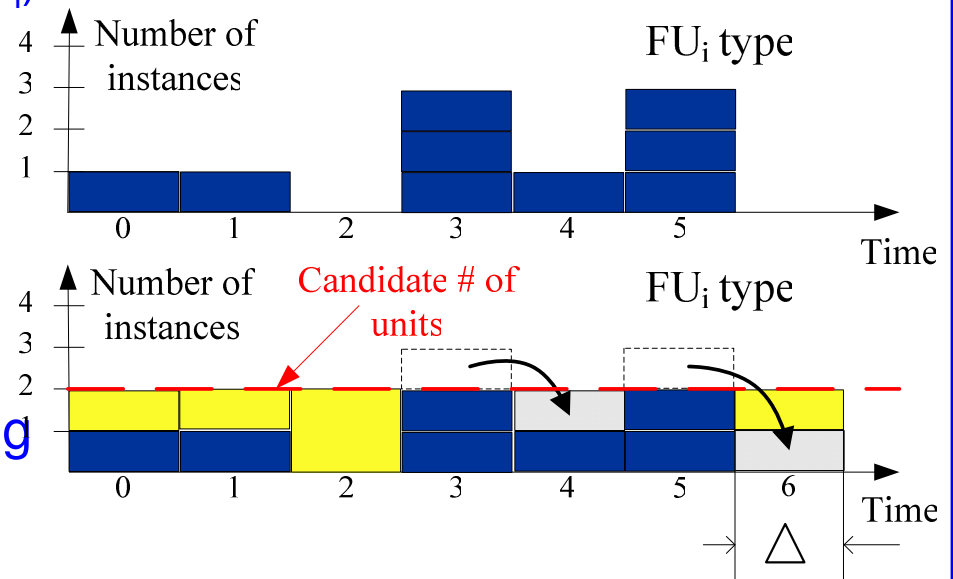
DP Creation and Evaluation (cont.)






“Spill” - Flattening Algorithm

- **Histogram: usage/cycle**
 - For each component type (e.g. FU_i)
- **Reduce number of units of FU**
 - Candidate number: the average number of FUs
- **For candidate number of units compute**
 - Overhead cycles (Δ) by postponing operations into empty slots
 - Utilization

$$U = \frac{\sum \text{Used FUs}}{\text{total\#} * \text{Exec. Time}}$$

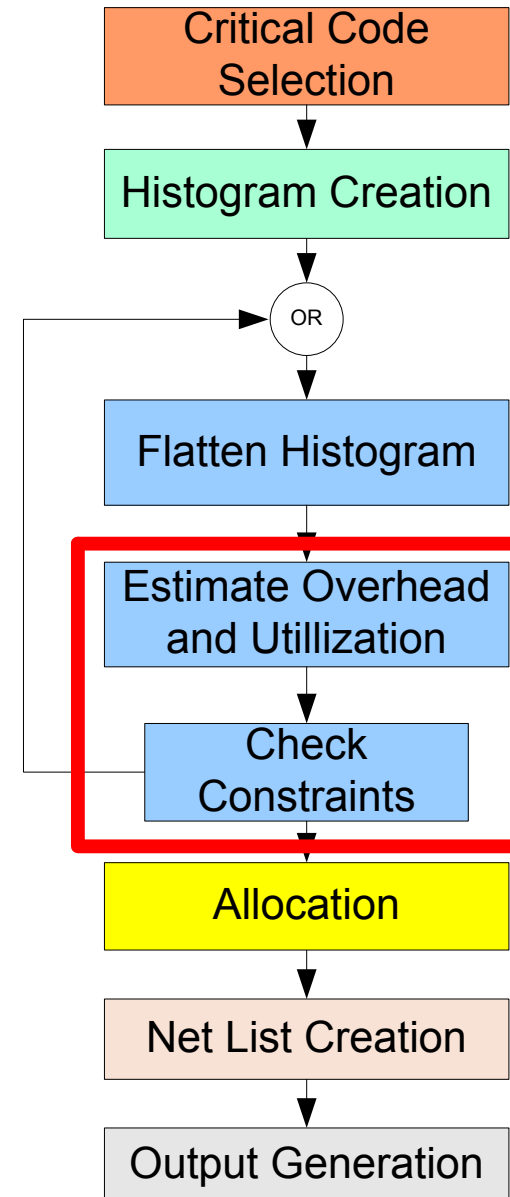


	FU in use in current cycle
	Estimated use of FU
	Available FU not in use

- **Computes *per block estimation* for Δ and U**

Estimation and Refinement

- **To compute Δ and utilization across all BBs**
 - Incorporate execution frequencies
- **For each component C**
 - Total Δ** is a sum of
B's frequency * Δ per block
 - Total U** is a sum of
B's frequency * U per block
- while **Total Δ** greater than **specified Δ**
 - Increase # of instances of C
- while **Total Δ** less than **specified Δ** and **Total U** less than **specified U**
 - Decrease # of instances of C
- **Repeat Flattening and Estimation**



Outline

- **Introduction and Problem Definition**
- **Data Path Extraction from C Code**
- **Experimental Results**
- **Conclusion and Future Directions**



Results (1/2)

- Application: bdist2 (MPEG2 encoder), Sort (bubble sort), dct32 (MP3), Mp3
- Baseline: Initial Architecture
- $\Delta_{spec} = 20\%$, $P_l = 0.7$, $P_f = 0.85$, $P_{fl} = 0.5$

Bench	FUs [%]	Buses [%]	Tri-State [%]	Pipe. Regs [%]	Δ [%]	Avg. Utilization	
						FU [%]	Buses [%]
bdist2	-50.0 (6)	-50.0 (6)	-70.0 (40)	-42.9 (21)	+18.8 (11021)	+214.8 (10.4)	+30.2 (70.7)
Sort	-25.0 (4)	-50.5 (6)	-67.7 (36)	-20.0 (15)	+0.3 (154421)	+23.8 (25.7)	+98.0 (37.4)
dct32	-33.3 (6)	-16.7 (6)	-52.5 (40)	-28.6 (21)	+19.6 (137964)	+42.5 (15.7)	+0.8 (73.2)
Mp3	-37.5 (8)	0.0 (6)	-40.9 (44)	-34.6 (26)	+1.1 (1077977)	+46.0 (8.0)	-8.2 (62.6)
Average	-36.5	-29.2	-57.5	-35.5	+9.9	+82.3	+30.2

Relative number of instances of components of generated data paths (number of instances for the baseline); Δ - simulated time overhead; Avg. Utilization – Average utilization per instance

Results (2/2)

- Performance constraint satisfied for all benchmark
- DP generated in minutes, whereas manual design would require months
- Applicable to large applications (Mp3)

Bench	Avg. Iter.	T [min]	LoC
bdist2	1.8	0.9	61
Sort	3.9	0.6	33
dct32	1.2	12.6	1006
Mp3	1.4	79.5	13898
Average	2.1	23.4	NA

Execution characteristics: Avg. Iter. – Average number of iterations component number changes; T – total tool run time; LoC – number of lines of C code

Conclusion

- **Established concept of extracting architecture from C code**
- **Presented automatic technique for architecture extraction**
- **Demonstrated its feasibility with tool implementation**
- **Results with large C applications show**
 - Automatically extracted architectures meet performance constraints
 - Architecture is extracted in order of minutes
- **Can be further refined manually for even better results**
- **Future work and issues**
 - Reduce area
 - Reduce complexity of FU, further reduce interconnect
 - Features
 - Forwarding, chaining, special function units and memory hierarchy



Acknowledgements

- **Thanks to:**
 - Mehrdad Reshadi and Bitra Gorjiara for compiler support, Verilog generator and stimulating discussions that have improved this work
 - Pramod Chandraiah for providing the Mp3 source code



Thank you

Questions?





References addendum

- Keywords: Architecture Description Language, application-specific processor, system design, modeling, synthesis, NISC, GNR, XML
- B. Gorjiara, M. Reshadi, P. Chandraiah, D. Gajski, "Generic Netlist Representation for System and PE Level Design Exploration", International Symposium on Hardware/Software Codesign and System Synthesis (CODES+ISSS), October 2006.
- B. Gorjiara, M. Reshadi, D. Gajski, "Generic Architecture Description for Retargetable Compilation and Synthesis of Application-Specific Pipelined IPs", International Conference on Computer Design (ICCD), October 2006.