Designing and Presenting Programming Languages in the Broader Research Community

CS 6610

23. November 2015

Today's Plan

Part 1: How are Programming Languages presented to diverse audiences?

Part 2: Create your own language in 8 easy steps! (aka an overview of the RAPID programming language)

Let's Discuss the Readings

- Angstadt, Weimer, and Skadron's *RAPID Programming of Pattern-Recognition Processors*, ASPLOS '16 (to appear)
- Frigo, Leiserson, and Randall's *The Implementation of the Cilk-5 Multithreaded Language*, PLDI '98
- Hnat, et al.'s *MacroLab: A Vector-based Macroprogramming Framework for Cyber-Physical Systems*, SenSys '08

Case Study

	RAPID	Cilk-5	MacroLab
Paper Audience	PL/Arch/OS	PL	CPS
Hardware Target	Automata Processor (Pattern-Recognition Processors)	Shared Memory Multi-Processor	Wireless Sensor Networks
Problem Addressed	Abstraction	Abstraction	Abstraction
# New Features	~7	~4	~3
Optimization	Tessellation	Fast/Slow Clone	DSCD

Supercomputing 2015

19. November 2015

CIVL: The Concurrency Intermediate Verification Language

The framework includes a verification tool (Sec. 3), based on model checking and symbolic execution, which can verify the following *standard properties*: absence of assertion violations, deadlocks, memory leaks, improper pointer dereferences or arithmetic, out-of-bound array indexes, reads of uninitialized variables, and divisions by 0.

In addition to the standard properties, a large number of *dialect-specific properties* are verified. For MPI, these

The verifier can also produce a *minimal counterexample* when a property violation is found. This is an execution trace of minimal length culminating in failure—something which greatly facilitates understanding, isolating, and repairing defects. Again, this contrasts with most testing and debugging methods in HPC, which often involve traces with astronomical numbers of threads, processes, or execution steps.

Some CIVL features are of course found in programming languages. CIVL's `$range` and `$domain` types are borrowed

...

used). Cilk [21] and Erlang [3] also provide `spawn` primitives, but don't allow nested procedures. One language which does provide all these features is Racket [20], and in fact the basic structure of a CIVL model can be represented in Racket in a straightforward way, though it is not clear how easily other aspects of C (e.g., pointers and heaps) could be represented.

Computer Scientists

- This Turing Award winner is best known for his work on distributed systems, including: notions of time, sequential consistency, the bakery algorithm for mutual exclusion, and the Byzantine Generals Problem. Additionally, he was an initial developer of LaTeX.

Trivia

- On November 9, 1989, government spokes-person Günter Schabowski improvised an answer to a reporter's question ("As far as I know, effective immediately … without delay"), leading to a massive "upheaval" in world plolitics. Name the event or the official name of the country Shabowski represented.

RAPID Programming of Pattern-Recognition Processors

Understand your target (and your audience)

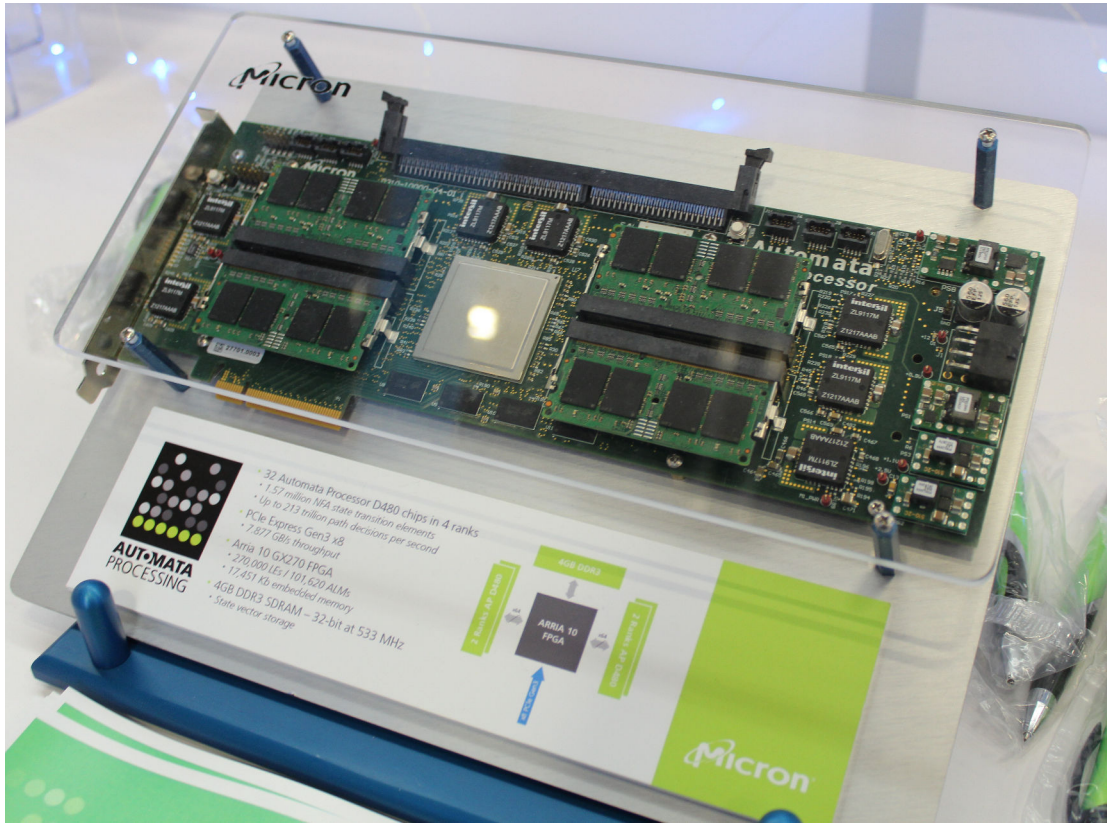
STEP 1

What is a Pattern?

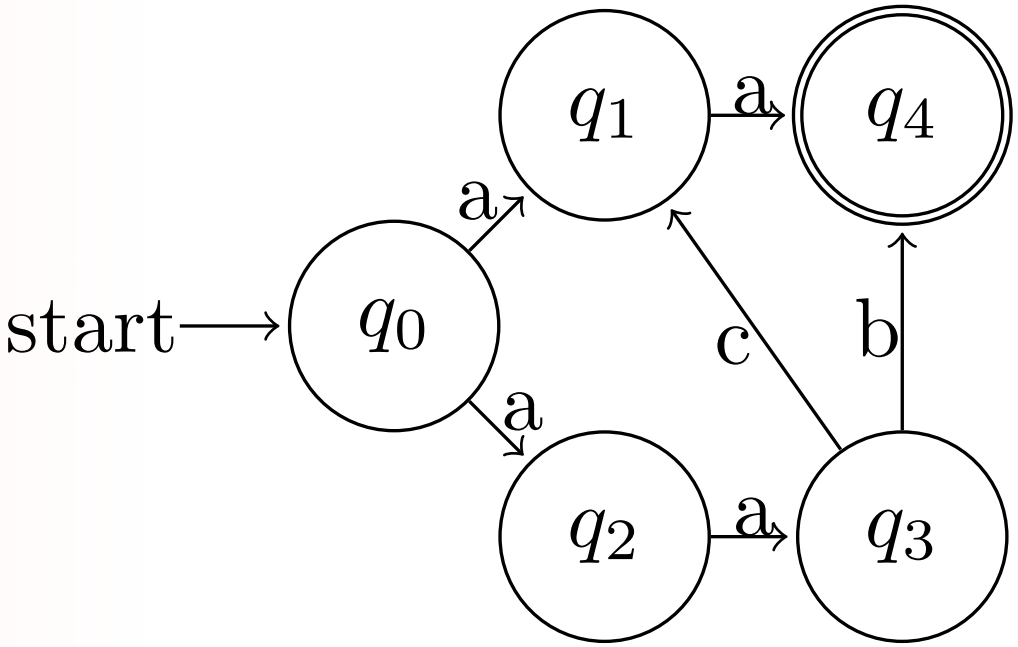
- Sequence of data that should be found within another collection of data
- We'll focus on NFAs/Regular Expressions
- Many problems can be phrased as pattern-recognition!
  - NLP: Brill tagging
  - ML: Association Rule Mining, Random Forests
  - Bioinformatics: Motif searches, K-mer matching
  - Entity Resolution
  - Network Intrusion Detection

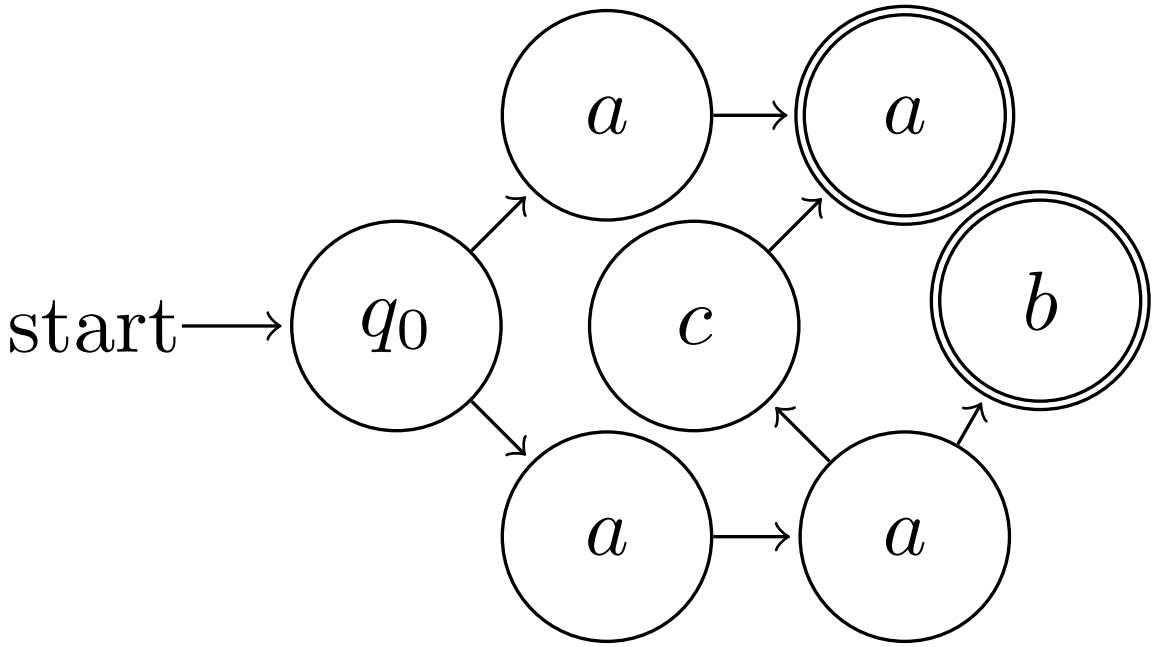
What is the Automata Processor?

- DRAM-based engine for executing NFAs
- 1.57 million state capacity
- 133 MHz Clock Speed
- 213 trillion transitions per second
- Added 12-bit counters and Boolean logic

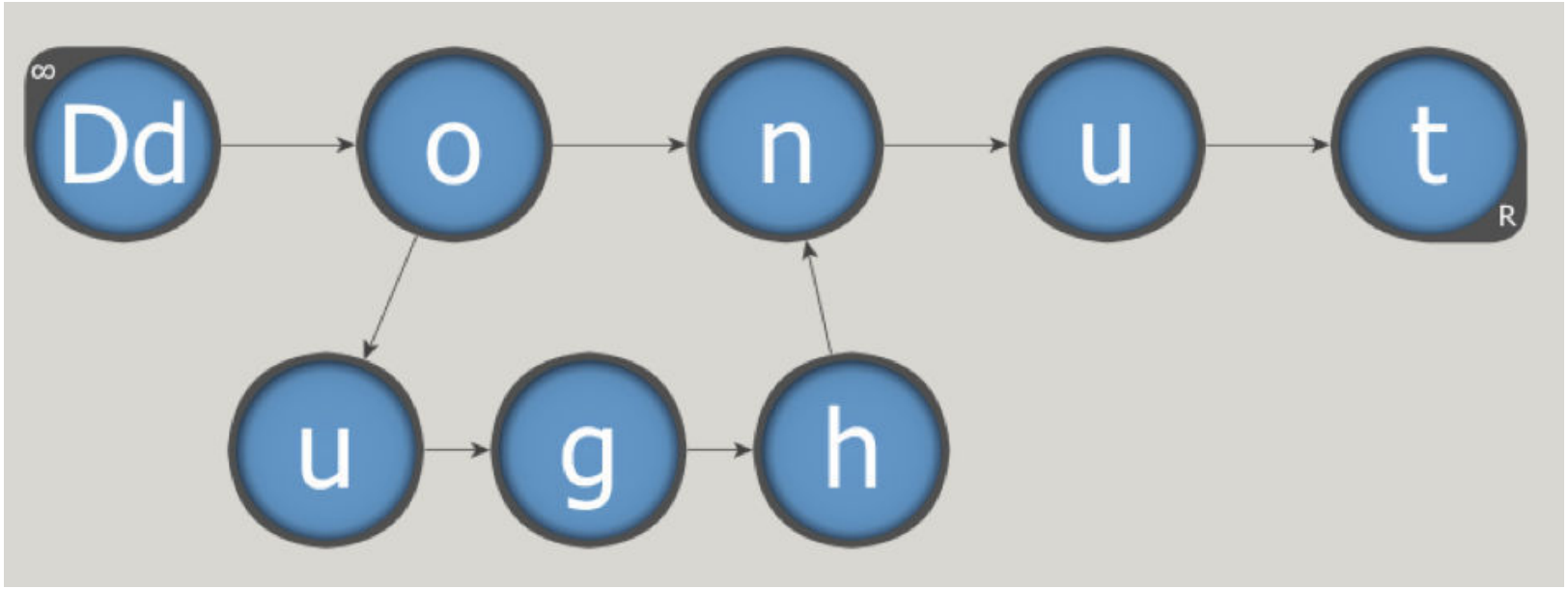


Homogeneous NFAs

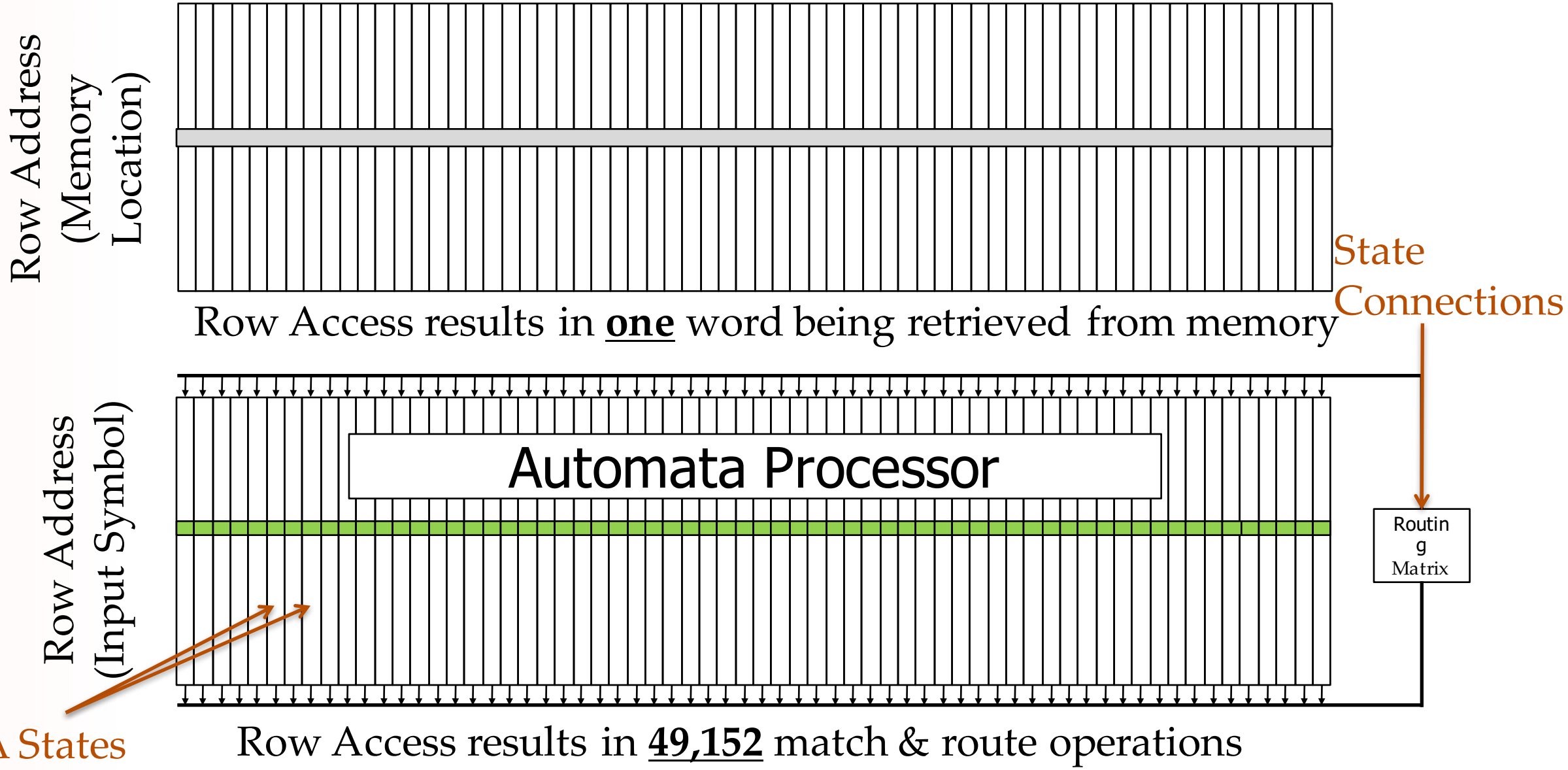




Example: Build a Homogeneous NFA



.*[Dd](o|ough)nut


Row Address (Memory Location)
Row Access results in one word being retrieved from memory
State Connections
Row Address (Input Symbol)
Automata Processor
Routin g Matrix
NFA States
Row Access results in 49,152 match & route operations

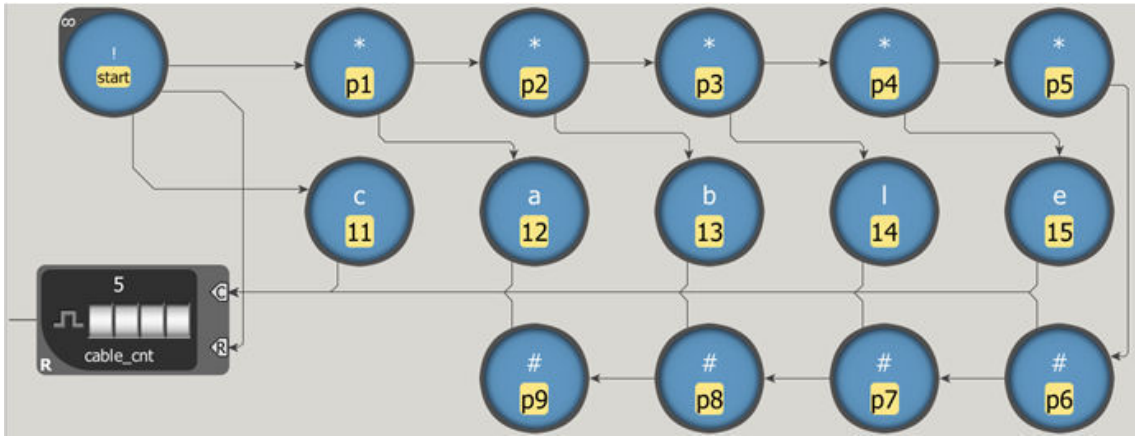
Current Programming Models

ANML

RegEx

ANML (Finite Automata)

```
from micronap.sdk import *

# Initialize the automaton
A = Anml()
AN = A.CreateAutomataNetwork(anmlId='hamming_distance')

# Add states
...
c = AN.AddSTE('c', anmlId='c')
a = AN.AddSte('a', anmlId='a')
b = AN.AddSte('b', anmlId='b')
l = AN.AddSte('l', anmlId='l')
e = AN.AddSte('e', anmlId='e')
...

# Add transitions
...
AN.AddAnmlEdge(p1, a, 0)
AN.AddAnmlEdge(p2, b, 0)
AN.AddAnmlEdge(p3, l, 0)
AN.AddAnmlEdge(p4, e, 0)
...

# Export the automaton to an ANML file
AN.ExportAnml('hamming_distance.anml')
```



- Verbose (62 lines of code)
- Not easily maintained
- Conceptual burden

Regular Expressions

```
/ right/JJ to/[^\s]+ /
/ the/[^\s]+ back/RB /
/ [^/]+/DT longer/[^\s]+ /
/ ,/[^\s]+ have/VB /
/ [^/]+/VBD by/[^\s]+ /
/ her/PRP\$ ,/[^\s]+ /
/ [^/]+/DT right/[^\s]+ /
/ [^/]+/IN 's/[^\s]+ /
/ [^/]+/RBS of/[^\s]+ /
/ had/[^\s]+ had/VBD /
/ set/VBN of/[^\s]+ /
/ [^/]+/VBG room/[^\s]+ /
/ [^/]+/VB of/[^\s]+ /
/ [^/]+/JJ by/[^\s]+ /
/ one/CD 's/[^\s]+ /
```

- Must know patterns at compile time
- Often exhaustive enumeration
  - How do we generate?
- Not easily maintained

Okay, so you need a new language (or extension)…

Pick an example application

STEP 2

2014 IEEE 28th International Parallel & Distributed Processing Symposium

Finding Motifs in Biological Sequences using the Micron Automata Processor

Indranil Roy and Srinivas Aluru
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, GA 30332
Email: iroy@gatech.edu; aluru@cc.gatech.edu

***Abstract*—Finding approximately conserved sequences, called *motifs*, across multiple DNA or protein sequences is an important problem in computational biology. In this paper, we consider the (l, d) *motif search* problem of identifying one or more motifs of length l present in at least q of the n given sequences, with each occurrence differing from the motif in at most d substitutions. The problem is known to be NP-hard, and the largest solved instance reported to date is $(26, 11)$. We propose a novel algorithm for the (l, d) motif search problem using streaming execution over a large set of Non-deterministic Finite Automata (NFA). This solution is designed to take advantage of the Micron Automata Processor, a new technology close to deployment that can simultaneously execute multiple NFA in parallel. We estimate the run-time for the $(39, 18)$ and $(40, 17)$ problem instances using the resources available within a single Automata Processor board. In addition to solving larger instances of the (l, d) motif search problem, the paper serves as a useful guide to solving problems using this new accelerator technology.**

***Index Terms*—computational biology; finite automaton; graph algorithms; hardware acceleration; motif detection.**

I. Introduction

Large repositories of genetic data have been produced through numerous sequencing projects, a trend that has significantly accelerated during the last decade. An important part of the analysis of this data consists of finding patterns in DNA, RNA or protein sequences. Discovery of these patterns called *motifs* helps in the identification of transcription factor binding sites, transcriptional regulatory elements and their consequences on gene functions, variants causing human diseases, and therapeutic drug targets.

The problem of motif search has been classified into the following three types: *Planted (l, d) Motif search Problem* (PMP), *Simple Motif search Problem* (SMP), and *Edit-distance-based Motif search Problem* (EMP). Of these three types, PMP has been studied in the greatest detail and is described as follows. Given n sequences of length m each, find a motif M of length l which occurs in all the sequences with upto d mismatches. For example, given the input sequences $AGTCTCTCGAG$, $TTAGACGGTCA$, and $GATCAGTTCAC$, and $l = 4$, $d = 1$, the motif $CTCA$ occurs in all the three sequences. Note that the motif itself need not be present in its exact form in any of the sequences, which is the case in this example. The generic form of the PMP is defined as the *Quorom Planted Motif search Problem* (qPMP), where the motif M is present in at least q of the n sequences. PMP (and therefore qPMP) has been shown to be NP-hard [1].

Though the first motif was discovered in 1970 [2], this problem is far from being satisfactorily solved. Heuristic algorithms [3], [4], [5], [6], [7] use statistical analysis to reach for solutions faster, but are not guaranteed to identify all motifs [8], sometimes even significant ones. On the other hand, *exact* algorithms [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20] can identify all the motifs, but take large amount of time and memory to do so. Since all exact algorithms return the same answers, they are evaluated based on their execution time. For this evaluation, 20 sequences of length 600 are randomly generated by using characters from the alphabet. Similarly, a motif M of length l is generated and planted at a random location in each sequence. Run-times are only reported for instances of (l, d) which are known to be *challenging*. For these instances, the expected number of (l, d) motifs found in the sequences apart from the planted motif is at least 1. Examples of such instances are $(17, 6)$, $(19, 7)$, $(21, 8)$, $(23, 9)$, and $(25, 10)$. At the time of this writing, the shortest execution times reported for the $(26, 11)$ instance on a 48 core machine is 46.9 hours, and the $(28, 12)$ instance remains unsolved to the best of our knowledge.

The Micron Automata Processor [21] is a novel non-Von Neumann semiconductor architecture which can be programmed to execute thousands of Non-deterministic Finite Automata (NFA) in parallel to identify patterns in a data stream. We designed an exact algorithm for solving PMP and qPMP using NFA programmed into the Micron Automata Processor. Our algorithm, termed MOTOMATA for *MOTif autOMATA*, not only has lower execution times but can also handle more challenging instances of the problem which are hitherto unsolved. To the best of our knowledge, this is the first instance of accelerating a complex application using the Automata Processor, and can be helpful to the community as this new accelerator technology becomes available.

The rest of the paper is organized as follows. A high-level description of the Micron Automata Processor is provided in Section II. Our parallel algorithm for the (l, d) motif search problem is described in Sections III through VI. Section VII contains the expected run-times for various sizes of the (l,d) motif search problem vis-à-vis prominent software based methods. Our findings are summarized in Section VIII.

1530-2075/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPS.2014.51

Properties to Look For

- Well-described and defined problem with a known solution
- Not easily solved with current techniques
- Seems to be representative of a large class of problems in the application space

Hamming distance: the number differences in corresponding characters between two strings

Extract the KERNEL computation and implement

Think about code generation at the same time!

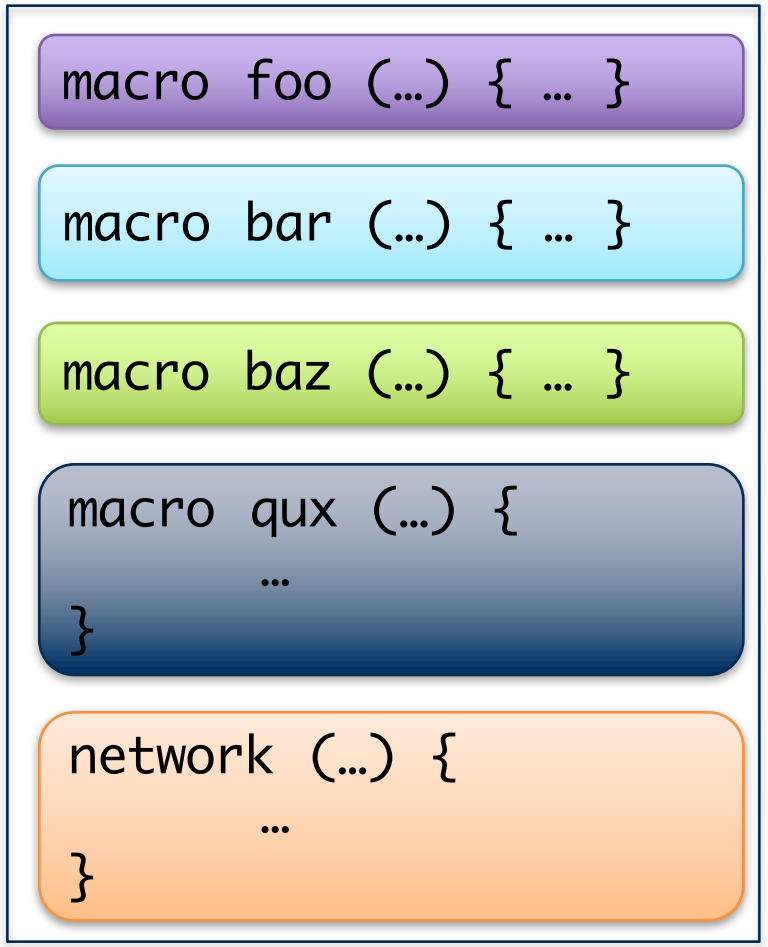
STEP 3

Pick a "Base" Language

- RAPID starts out with a C-like base language
- Add a few abstractions for ease of programming (foreach)
- Boolean expressions as statements (inspired by literature search)
- Make sure you can support the basic features of the architecture

Program Structure

- **Macro**
  - Basic unit of computation
  - Sequential control flow
  - Boolean Expressions as Statements
  - More dynamic than ANML macros
- **Network**
  - High-level pattern matching
  - Parallel control flow
  - Parameters to set run-time values



Types in Rapid

- `int`
- `char`
- `boolean`
- `String` (`.length()`, `charAt(i)`, etc.)
- `Counter` (`.count()` and `.reset()`)
- Arrays

Data in RAPID

- Input data stream as special function
  - Stream of characters
  - `input()`
    - Calls to `input()` are synchronized across all active macros
    - All active macros receive the same input character

Hamming Distance in RAPID

```
1 macro hamming_distance (String s, int d) {
2       Counter cnt;
3       foreach (char c : s)
4           if(c != input()) cnt.count();
5       cnt <= d;
6       report;
7 }
```

Okay, that went well…

Let's try to implement more of the algorithm (adding language features as we need them)

STEP 4

Need to Iterate Over Multiple Candidates in Patterns and Input Stream

- Special Symbols
  - `ALL_INPUT`
  - `START_OF_INPUT`
- We also need parallel control flow

Either/Orelse Statements

```
1 either {
2     hamming_distance(s,d); //hamming distance
3     'y' == input();        //next input is 'y'
4     report;                //report candidate
5 } orelse {
6     while('y' != input()); //consume until 'y'
7 }
```

- Perform parallel exploration of input data
- Static number of parallel operations

Some Statements

- Parallel exploration may depend on candidate patterns
- We need a way of dynamically generating parallel computation
- Some statement is a “parallel for loop” over data

Some Statements

```
1  macro hamming_distance (String s, int d) {
2      Counter cnt;
3      foreach (char c : s)
4          if(c != input()) cnt.count();
5      cnt <= d;
6      report;
7  }
8  network (String[] comparisons) {
9      some(String s : comparisons)
10         hamming_distance(s,5);
11 }
```

Iterate with new application examples

Do you need any additional program features?

STEP 5

2015 IEEE 29th International Parallel and Distributed Processing Symposium

Association Rule Mining with the Micron Automata Processor

Ke Wang†, Yanjun Qi†, Jeffrey J. Fox‡, Mircea R. Stan§ and Kevin Skadron†
†*Dept. of Comp. Sci.,* ‡*Dept. of Mater. Sci.,* §*Dept. of Elec. & Comp. Eng.*
University of Virginia
Charlottesville, VA, 22904 USA
{*kewang, yanjun, jjf5x, mircea, skadron*}*@virginia.edu*

***Abstract*—Association rule mining (ARM) is a widely used data mining technique for discovering sets of frequently associated items in large databases. As datasets grow in size and real-time analysis becomes important, the performance of ARM implementation can impede its applicability. We accelerate ARM by using Micron's Automata Processor (AP), a hardware implementation of non-deterministic finite automata (NFAs), with additional features that significantly expand the APs capabilities beyond those of traditional NFAs. The Apriori algorithm that ARM uses for discovering itemsets maps naturally to the massive parallelism of the AP. We implement the multipass pruning strategy used in the Apriori ARM through the APs symbol replacement capability, a form of lightweight reconfigurability. Up to 129X and 49X speedups are achieved by the AP-accelerated Apriori on seven synthetic and real-world datasets, when compared with the Apriori single-core CPU implementation and Eclat, a more efficient ARM algorithm, 6-core multicore CPU implementation, respectively. The AP-accelerated Apriori solution also outperforms GPU implementations of Eclat especially for large datasets. Technology scaling projections suggest even better speedups from future generations of AP.**

***Keywords*-Automata Processor; association rule mining; frequent set mining**

I. Introduction

Association Rule Mining (ARM), also referred to as Frequent Set Mining (FSM), is a data-mining technique that identifies strong and interesting relations between variables in databases using different measures of "interestingness". ARM has been the key module of many recommendation systems and has created many commercial opportunities for on-line retail stores. In the past 10 years, this technique has also been widely used in web usage mining, traffic accident analysis, intrusion detection, market basket analysis, bioinformatics, etc.

As modern databases continue to grow rapidly, the execution efficiency of ARM becomes a bottleneck for its application in new domains. Many previous studies have been devoted to improving the performance of sequential CPU-based ARM implementations. Different data structures were proposed, including horizontal representation, vertical representation, and matrix representation [1]. Multiple algorithms have been developed including *Apriori* [2], *Eclat* [3] and *FP-growth* [4]. A number of parallel acceleration based solutions have also been developed on multi-core CPU [5], GPU [6] and FPGA [7].

Recently, Micron proposed a novel and powerful non-von Neumann architecture, the Automata Processor (AP). The AP architecture demonstrates a massively parallel computing ability through a large number of state elements. It also achieves fine-grained communication ability through its configurable routing mechanism. These advantages make the AP suitable for pattern-matching centered tasks like ARM. Very recently, the AP has been successfully used to accelerate the tasks of regular expression matching [8] and DNA motif searching [9].

In this paper, we propose an AP-based acceleration solution for ARM. A Non-deterministic Finite Automata (NFA) is designed to recognize the sets of frequent items. Counter elements of the AP are used to count the frequencies of itemsets. We also introduce a number of optimization strategies to improve the performance of AP-based ARM. On multiple synthetic and real-world datasets, we compare the performance of the proposed AP-accelerated *Apriori* versus the *Apriori* single-core CPU implementation, as well as multicore and GPU implementations of the *Eclat* algorithm. The proposed solution achieves up to 129X speedups when compared with the *Apriori* single-core CPU implementation and up to 49X speedups over multicore implementation of *Eclat*. It also outperforms GPU implementations of *Eclat* in some cases, especially on large datasets.

Overall, this paper makes three principal contributions:

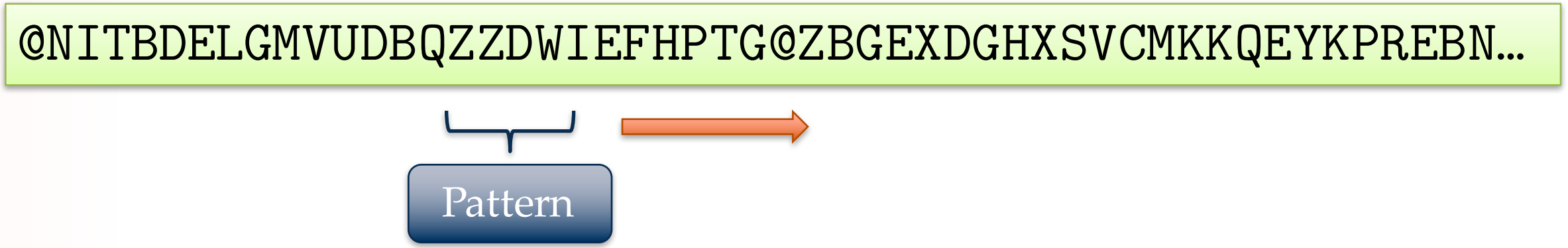
1) We develop a CPU-AP computing infrastructure to improve the *Apriori* algorithm based ARM.
2) We design a novel automaton structure for the matching and counting operations in ARM. This structure avoids routing reconfiguration of the AP during the mining process.
3) Our AP ARM solution shows performance improvement and broader capability over multi-core and GPU implementations of *Eclat* ARM on large datasets.

II. Association Rule Mining

Association rule mining (ARM) among sets of items was first described by Agrawal and Srikant [10]. The ARM problem was initially studied to find regularities in the

1530-2075/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPS.2015.101

Sliding Window Search



ARM needs custom sliding window!

All RAPID programs perform sliding window search on `START_OF_INPUT` (automatically inferred)

Whenever Statements

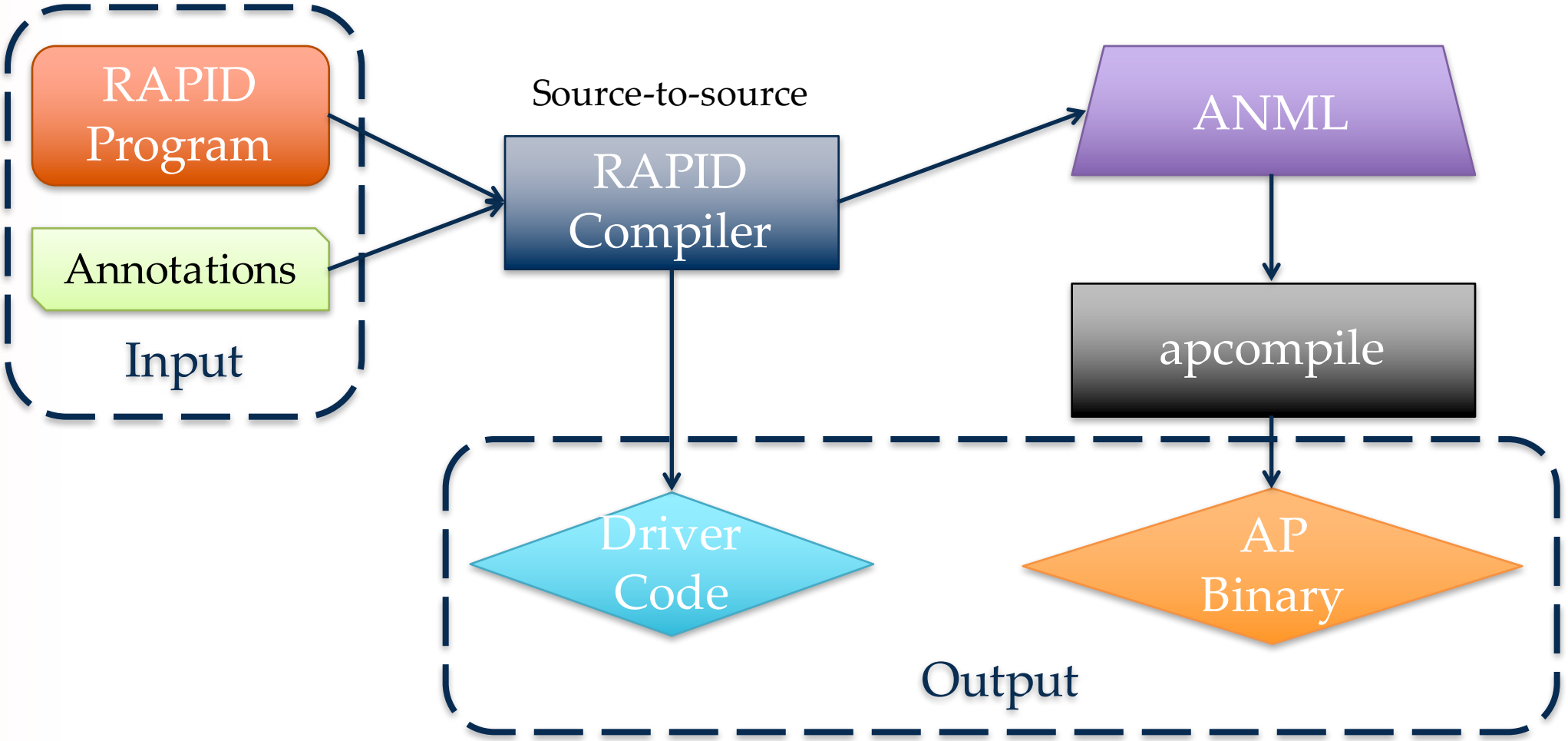
```
1 whenever( ALL_INPUT == input() ) {
2       foreach(char c : "rapid")
3           c == input();
4       report;
5 }
```

- Parallel equivalent of while loop
- Body triggered whenever guard becomes true

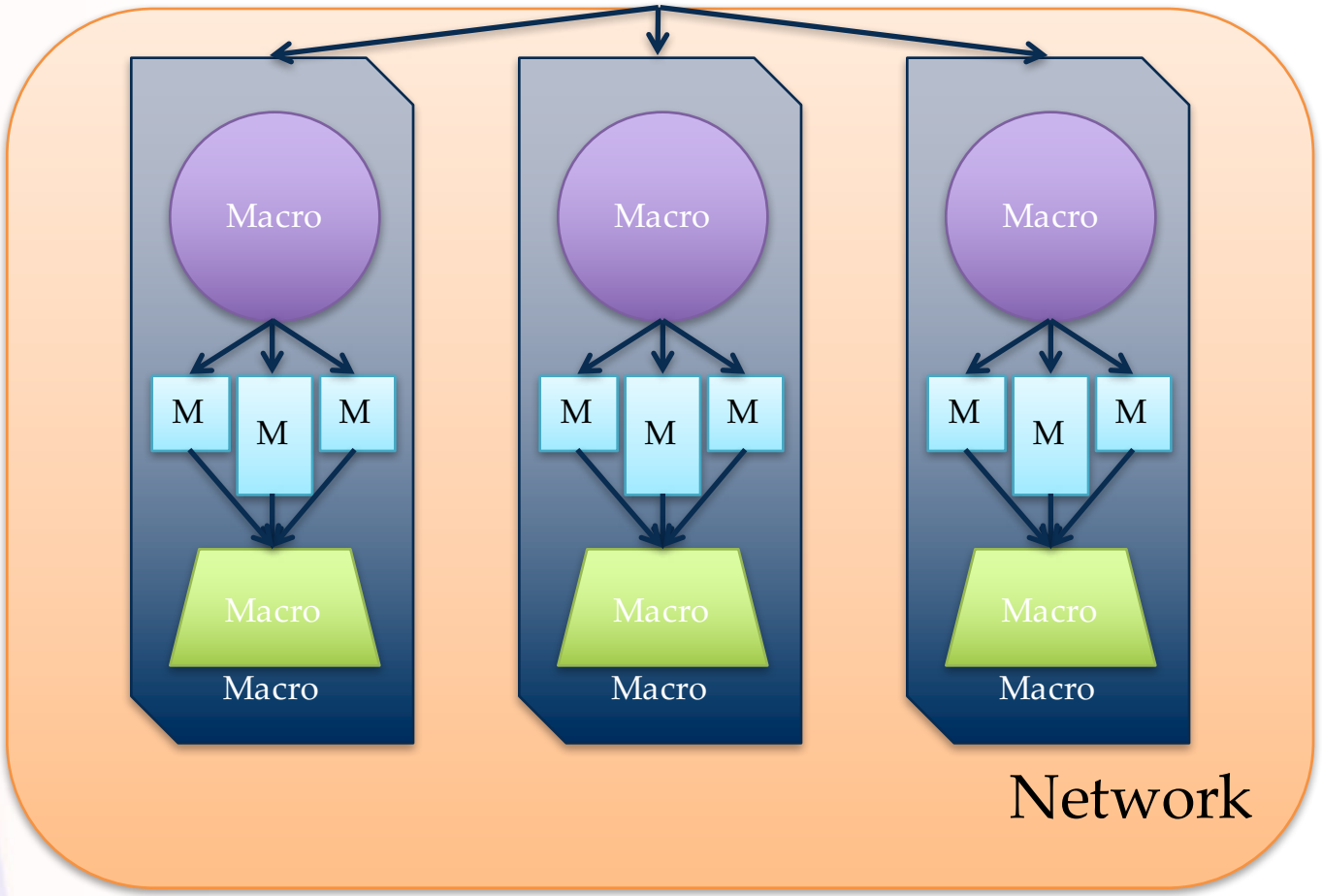
Make sure you can generate code!

STEP 6

System Overview



Program Structure

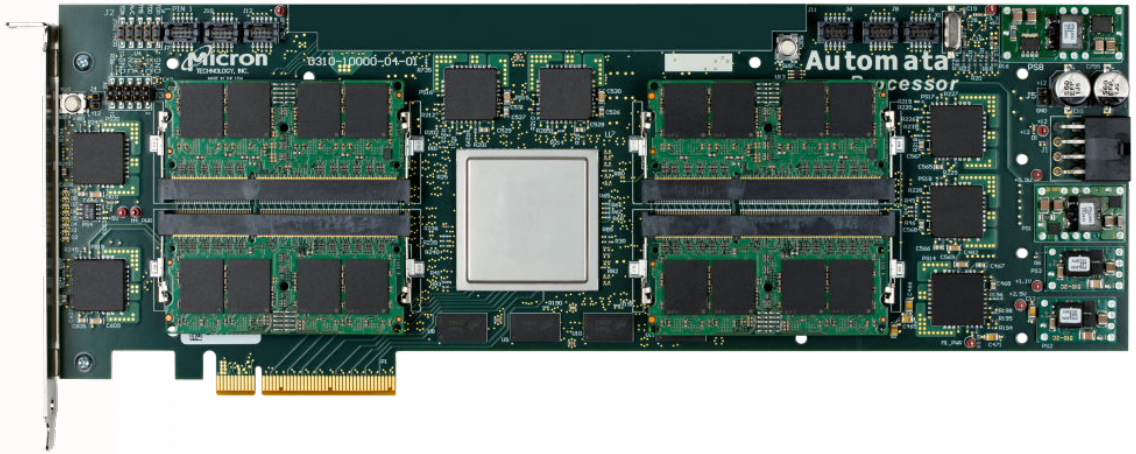



```
macro foo (…) { … }

macro bar (…) { … }

macro baz (…) { … }

macro qux (…) {
        …
}

network ( … ) {
        …
}
```

Code Generation

- Recursively transform RAPID program
  - Input Stream → States
  - Counters → 1 or more physical counter(s)
- Similar to RegEx → NFA transformation

Staged Computation



Runtime

Comparisons with Input Stream
Counters

Compile Time

Integer Computation
Iteration Over Strings
etc.

Compiling for the full AP board

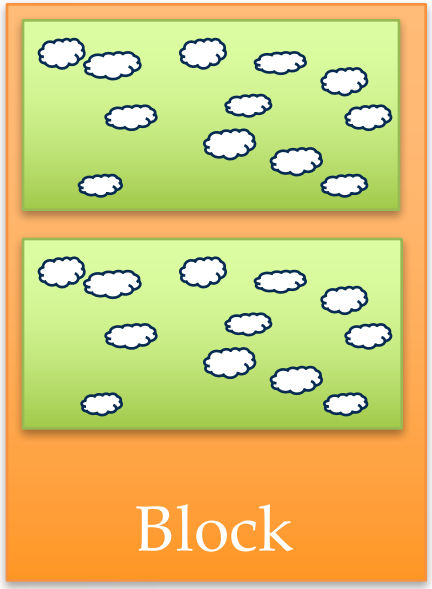
AUTO-TUNING TESSELLATION

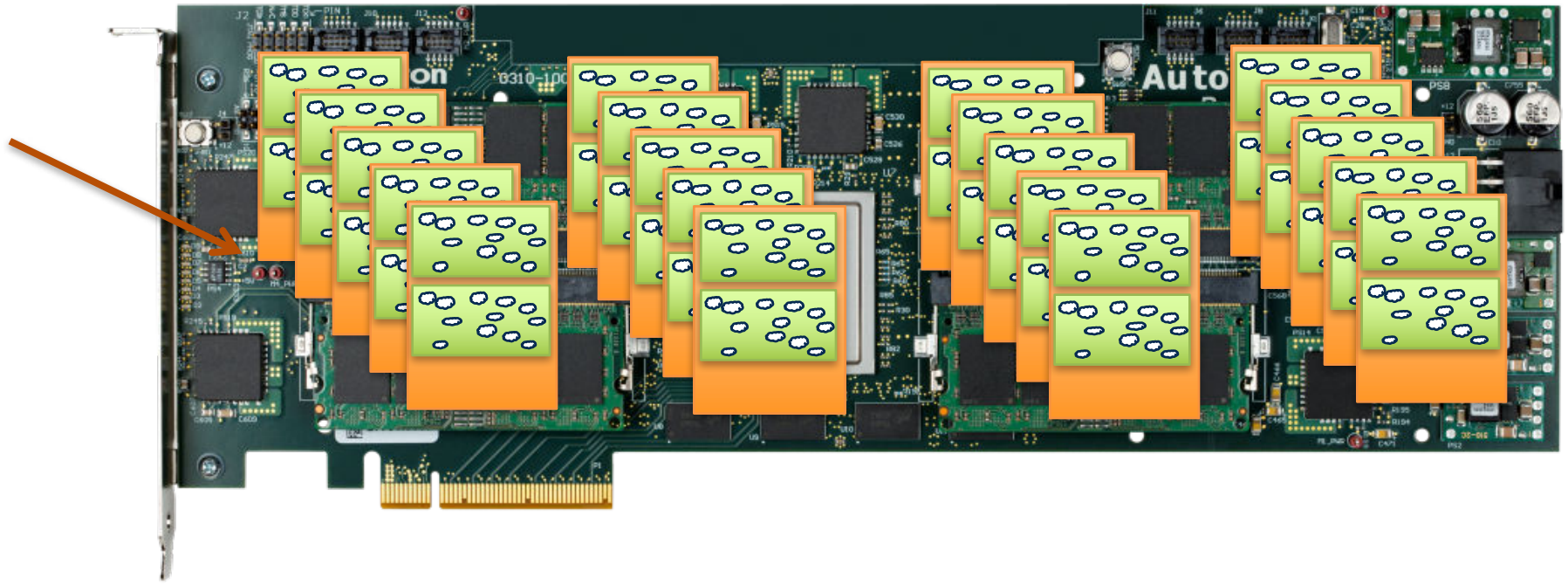
Oh wait! It doesn't compile well? That's a pity…

STEP 7

Block-Level Configuration

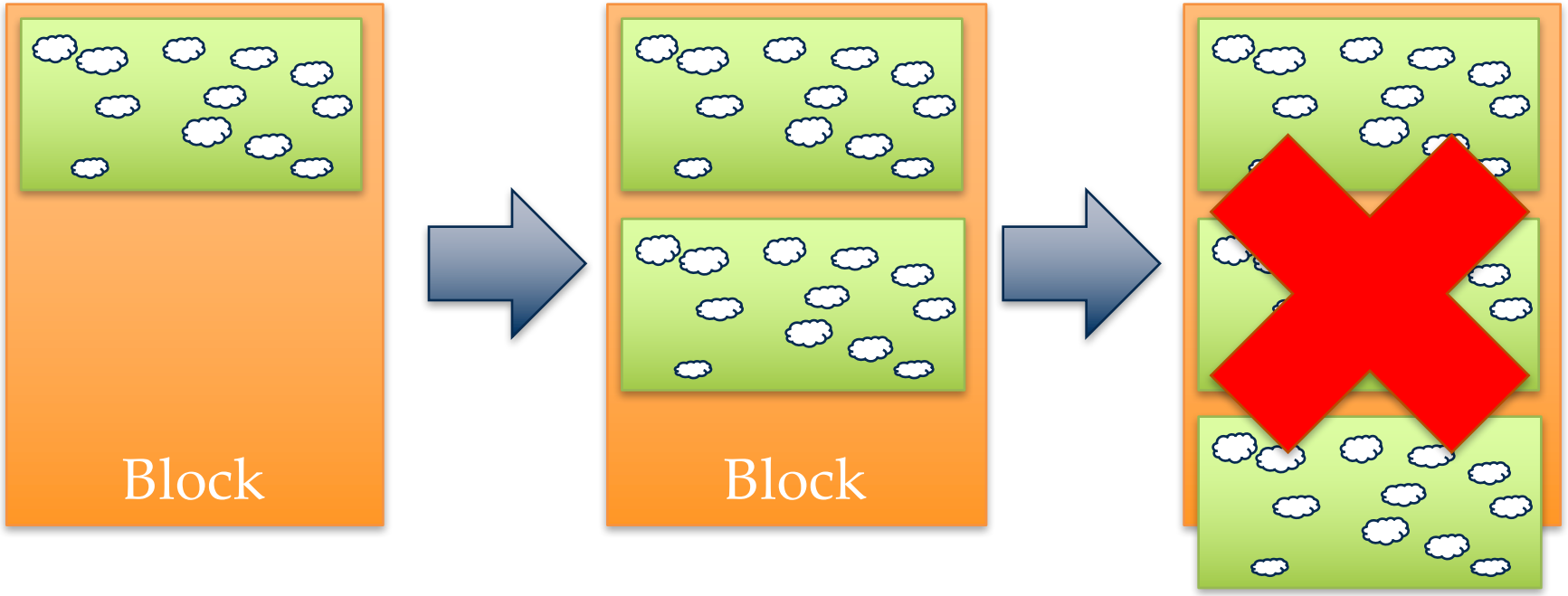
Each computation takes up **SPACE** on the AP (**TIME** is dependent on the input)



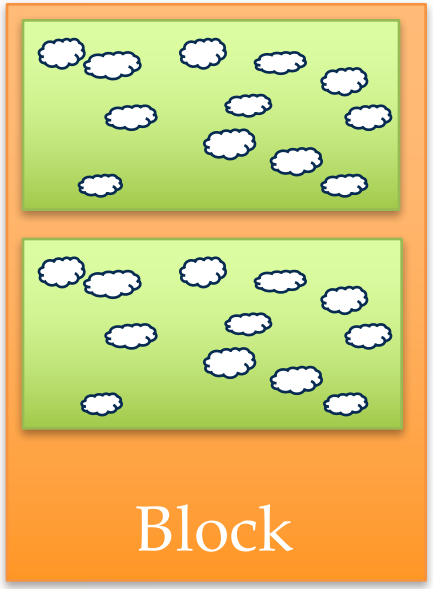


Load blocks dynamically and set parameters at runtime

Auto-Tuning



Block-Level Configuration



- Easier to dynamically load designs at runtime
- Relatively fast compilation
- Auto-tuning heuristic improves utilization

Evaluation!

STEP 8

Description of Benchmarks

Benchmark	Description	Generation Method	Sample Instance Size
ARM	Association rule mining	Python + ANML	24 Item-Set
Brill	Rule re-writing for Brill part of speech tagging	Java	219 Rules
Exact	Exact match DNA sequence search	Workbench	25 Base Pairs
Gappy	DNA string search with allowances for gaps between characters	Workbench	25-bp, Gaps $\leqslant 3$
MOTOMATA	Fuzzy matching for planted motif Search in bioinformatics	Workbench	(17,6) Motifs

RAPID LOC Comparison

Benchmark		LOC	ANML LOC	STEs	Device STEs
ARM	R	18	214	58	56
	H	118	301	79	58
Brill	R	688	10,594	3,322	1,429
	H	1,292	9,698	3,073	1,514
	Re	218	–‡	4,075	1,501
Exact	R	14	85	29	27
	H	–†	193	28	27
Gappy	R	30	2,337	748	399
	H	–†	2,155	675	123
MOTOMATA	R	34	207	53	72
	H	–†	587	150	149

R – RAPID *H – Hand-coded* *Re – Regular Expression*

† The GUI-tool does not have a LOC equivalent metric.

‡ No ANML statistics are provided by the regular expression compiler.

Device Utilization

Benchmark		Total Blocks	Clock Divisor	STE Util.	Mean BR Alloc.
ARM	R	1	1	21.9%	20.8%
	H	1	1	23.4%	20.8%
Brill	R	8	1	84.0%	52.6%
	H	12	1	57.9%	65.4%
	Re	10	1	71.4%	60.6%
Exact	R	1	1	10.9%	4.2%
	H	1	1	10.9%	4.2%
Gappy	R	2	1	89.5%	70.8%
	H	2	1	37.5%	77.1%
MOTOMATA	R	1	2	33.6%	75.0%
	H	4	1	17.2%	75.0%

R – RAPID H – Hand-coded Re – Regular Expression

Automatic Tessellation

Benchmark		Problem Size (# instances)	Total Blocks	Generate Time (sec)	Place and Route Time (sec)	Total Time (sec)
ARM	B†	8,500	–	5.38	–	–
	P	8,500	6,100	6.53	771.16	770.70
	R	8,500	2,125	3.70	0.41	4.12
Exact	B	46 000	4 730	24.52	22 011.10	22 035.62
	P	46 000	6 075	30.17	1 676.88	1 707.05
	R	46 000	5 750	0.80	0.08	0.88
Gappy	B	2,000	5,354	0.99	9158.00	9158.99
	P†	2,000	–	0.99	–	–
	R	2,000	4,000	9.17	2.20	11.36
MOTOMATA	B	1 500	5 320	1.49	5 874.51	5 875.99
	P	1 500	6 001	1.45	210.62	212.07
	R	1 500	1 500	2.26	0.37	2.63

B – Baseline (No Pre-Compilation) *P – Pre-Compiled Designs* *R – RAPID Tessellation*

† The current AP software is not able to support placement and routing for this benchmark.

Conclusions

- RAPID admits clear, concise, maintainable, and efficient pattern-recognition programs
- Language development is an iterative process
- Make the common tasks easy (remember your target audience!)
- The fewer new features, the better (but they'll have high impact!)