SAND2022-2234C

Sandia National Laboratories

Exceptional service in the national interest

Simulating Next-Gen Dataflow Architectures for HPC

2022 SIAM Conference on Parallel Processing for Scientific Computing

Presented by Clay Hughes, Sandia National Laboratories

Clay Hughes, Gwendolyn Voskuilen, Arun Rodrigues, and Simon Hammond, Sandia National Laboratories

SAND2022-

U.S. DEPARTMENT OF ENERGY

NNSA

Introduction

Post-exascale era in computing will look much different from today’s landscape

Slowing of Moore’s law is driving the computing community toward more specialized forms of compute to achieve power, performance, & reliability

- Explosion of accelerator designs from both industry and academia

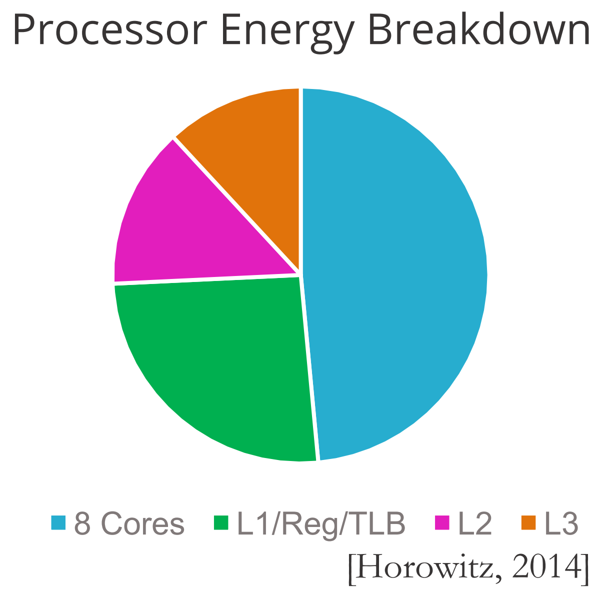
The number of possibilities for system and node design in this new paradigm will necessitate new simulation models and capabilities

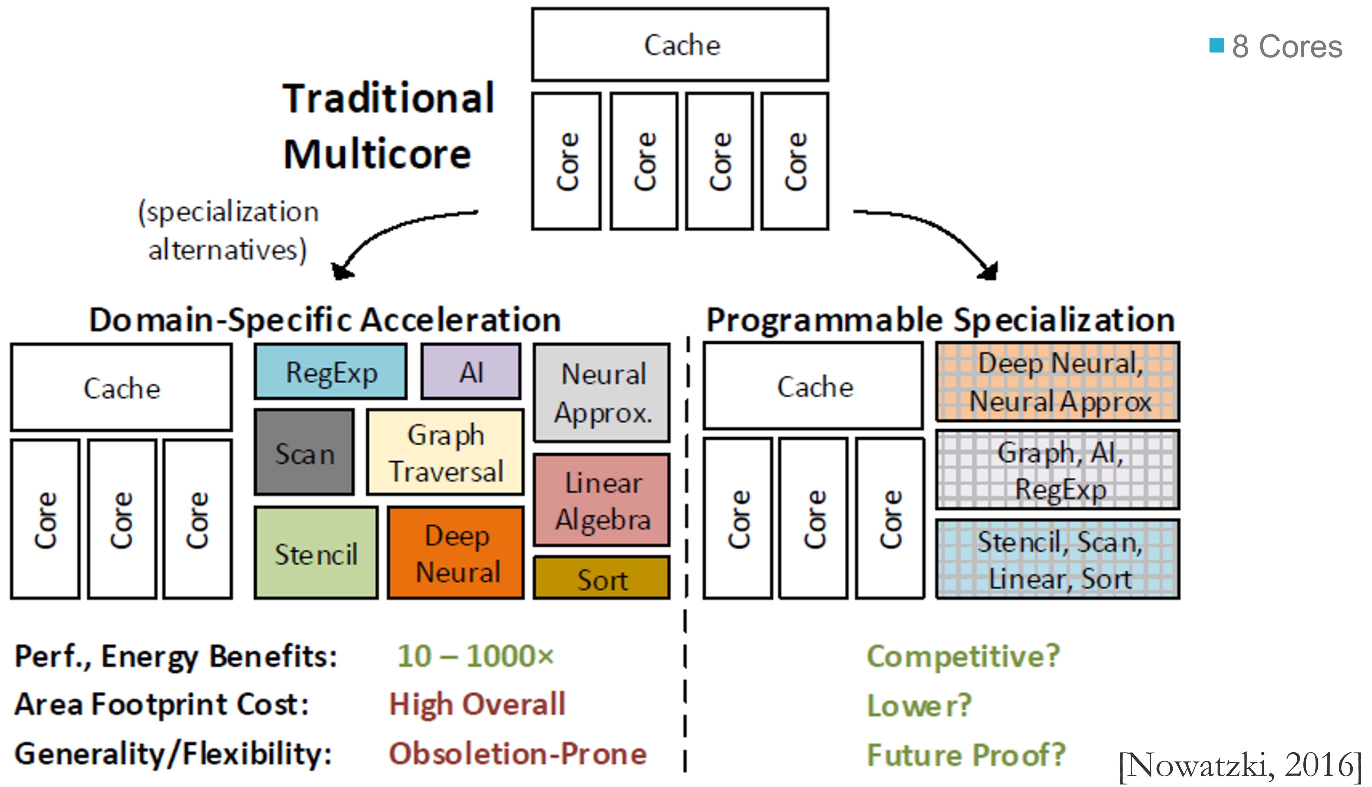
Age of the Accelerator?

Accelerators will likely provide the best power-performance for many domains in the future

But how much specialization is needed?

What are the tradeoffs?



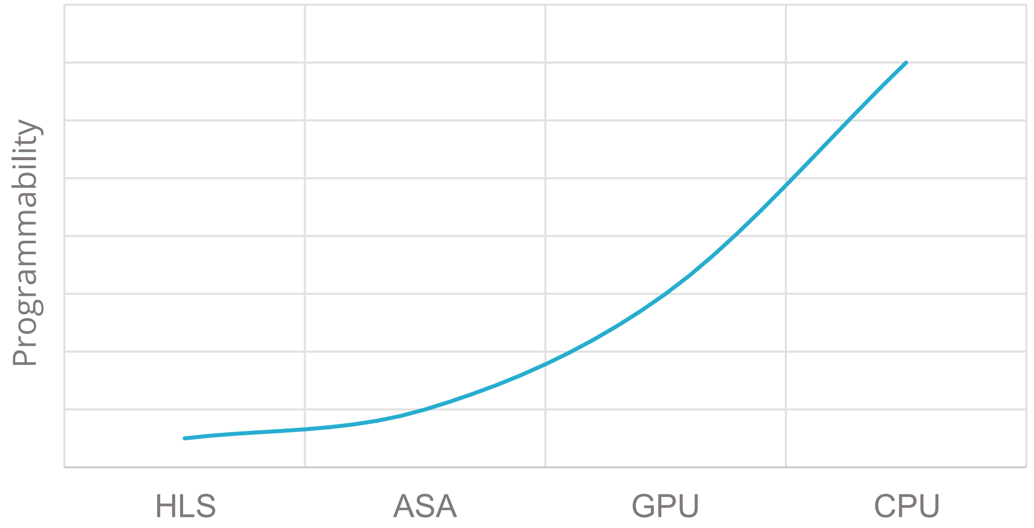


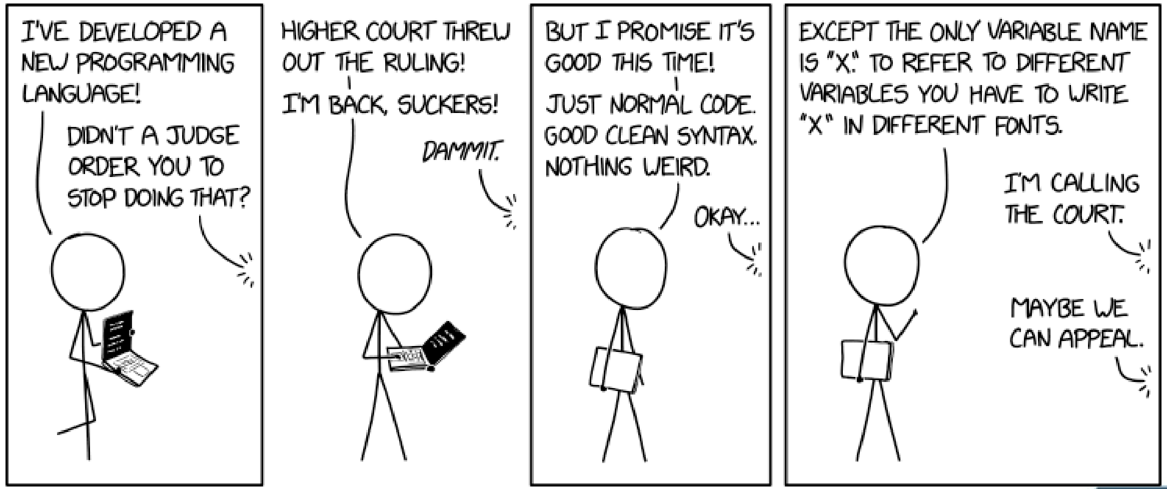
Accelerator Domains

High-Level Synthesis
- Designed for a very specific task (C/C++ → RTL)
- Examples: FPGA, CGRA

Application-Specific Architectures
- Co-designed hardware for single application/domain
- Examples: MAERI, SIGMA, TPU, ASV

General Purpose
- Suitable for many applications/domains
- Examples: GPU, DSP




I'VE DEVELOPED A NEW PROGRAMMING LANGUAGE!
DIDN'T A JUDGE ORDER YOU TO STOP DOING THAT?
HIGHER COURT THREW OUT THE RULING!
I'M BACK, SUCKERS!
DAMMIT.
BUT I PROMISE IT'S GOOD THIS TIME!
JUST NORMAL CODE. GOOD CLEAN SYNTAX. NOTHING WEIRD.
OKAY...
EXCEPT THE ONLY VARIABLE NAME IS "X." TO REFER TO DIFFERENT VARIABLES YOU HAVE TO WRITE "X" IN DIFFERENT FONTS.
I'M CALLING THE COURT.
MAYBE WE CAN APPEAL.
xkcd.com/2309

Which Accelerator?

Open question with many potential research areas

Many research projects are exploring spatial accelerators

- Might be easier to program
  - Common APIs and compiler infrastructure
- Energy efficient
  - Similar to CGRAs
- Performant? Probably better than SIMT for some application domains
  - Particle transport
  - CNN/DNN (MatVec, MatMat, SpMM)

Spatial/Dataflow Accelerators

Compilers build internal representations of applications that represent the behavior as a series of graphs -- abstracting the control and data flow

Traditional processors execute instruction sequentially, destroying an application's inherent instruction-level parallelism (ILP)

- Superscalar OoO processors go to great lengths to reconstruct the ILP
  - Multiple queues and complex logic allow instructions to issue when operands are available rather than in program order
  - Results are placed in additional queues and made visible to the system in program order even if they are completed out-of-order

Dataflow architectures are able to execute these graphs directly, without the need to flatten the graph and artificially recover the parallelism

Dataflow Graph -- *multiply_test()*

C++ Function

```
void  multiply_test(int* a, int *b, int* const c) {
   int f = 3 * (*a);
   int g = 2 * (*b);
   *c = f + g;
}
```

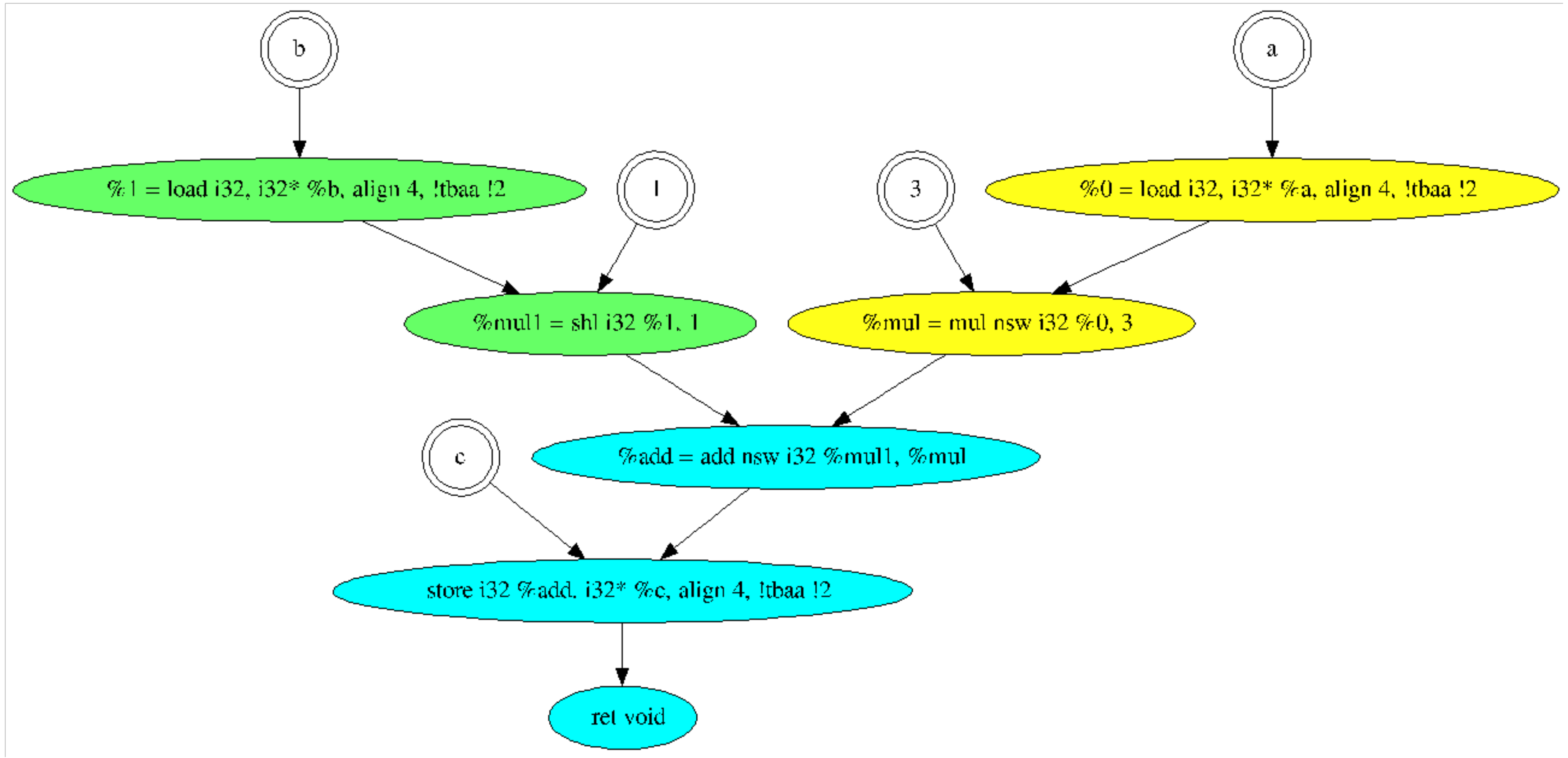
LLVM IR

```
; Function Attrs: norecurse nounwind uwtable
define void @multiply_test(i32* %a, i32* %b, i32* %c) local_unnamed_addr #0 {
entry:
 %0 = load i32, i32* %a, align 4, !tbaa !2
 %mul = mul nsw i32 %0, 3
 %1 = load i32, i32* %b, align 4, !tbaa !2
 %mul1 = shl i32 %1, 1
  %add = add nsw i32 %mul1, %mul
  store i32 %add, i32* %c, align 4, !tbaa !2
  ret void
}
```

Dataflow Graph -- *multiply_test()*

C++ Function

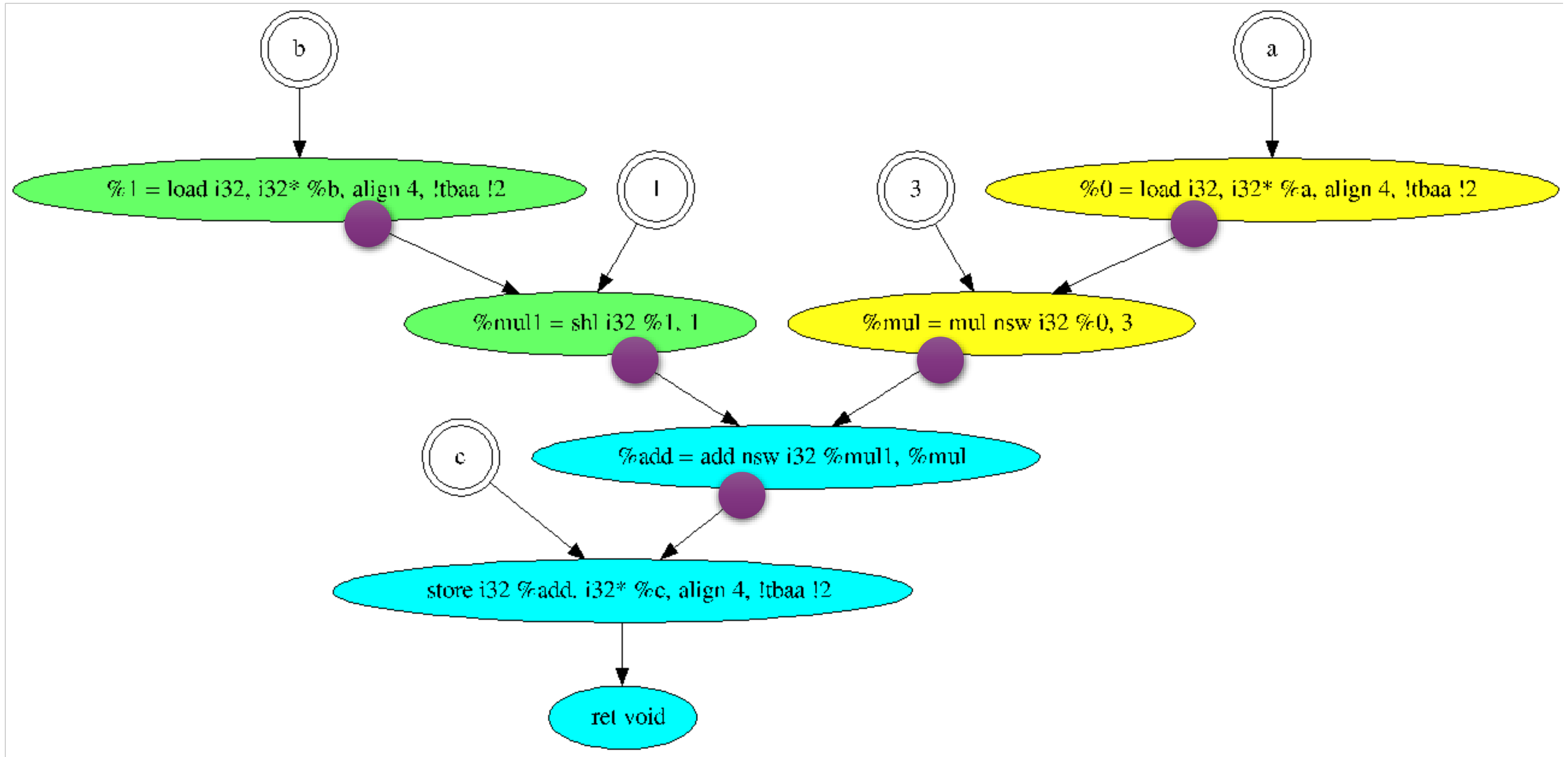
```
void  multiply_test(int* a, int *b, int* const c) {
   int f = 3 * (*a);
   int g = 2 * (*b);
   *c = f + g;
}
```


b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void

Dataflow Graph -- *multiply_test()*

C++ Function

```
void  multiply_test(int* a, int *b, int* const c) {
   int f = 3 * (*a);
   int g = 2 * (*b);
   *c = f + g;
}
```



b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void

Simulating Dataflow Architectures

There are examples of each of the different types of accelerators in the literature

- Dedicated/Static
  - Softbrain
- Dedicated/Dynamic
  - Plasticine, SPU, MAERI
- Shared/Static
  - CGRA
- Shared/Dynamic
  - TRIPS, SGMF

Currently developing dataflow component for Structural Simulation Toolkit (SST) for dedicated/dynamic designs

- Flexible interface to add custom PEs
- Arbitrary connectivity
- Mappers are dynamically loaded allowing them to be swapped at runtime
- Leverages SST component interface to enable scalable simulations

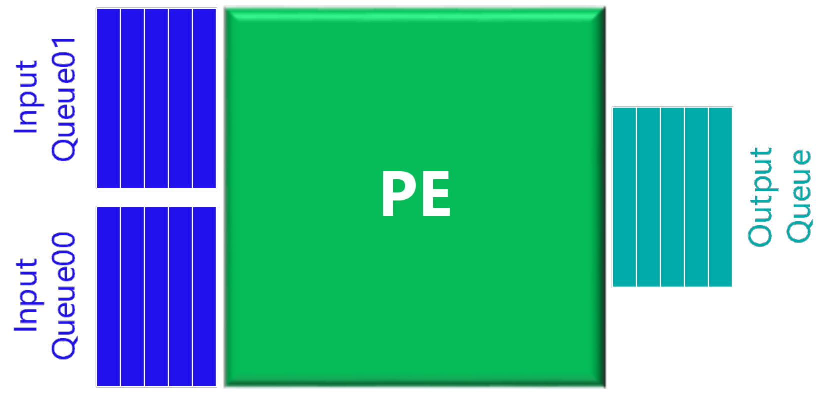
SST Dataflow Component

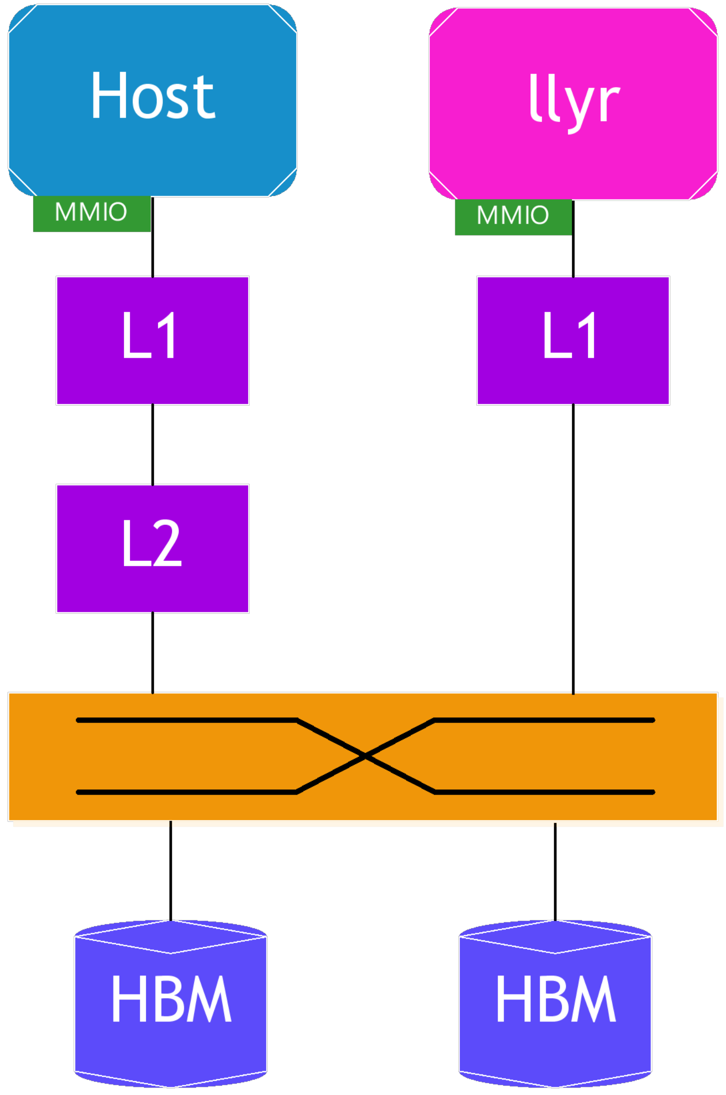
Multiple instances, allows for different configurations of each instance

- Arbitrary memory hierarchy
- Some limitations on node configurations
- Reconfigurable at runtime*

PEs have a compute unit, input buffers, and output buffers

- Number of buffers bounded by connectivity
- Buffer depth is configurable but is uniform for all PEs

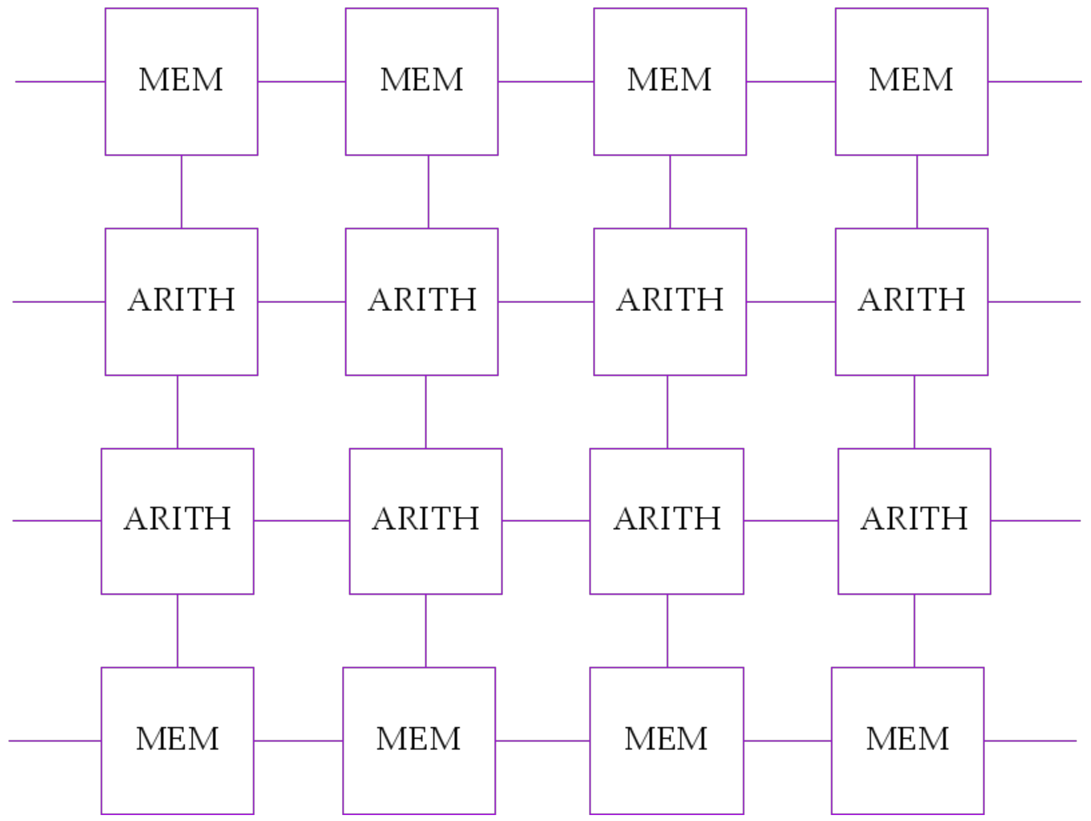




SST llyr Component

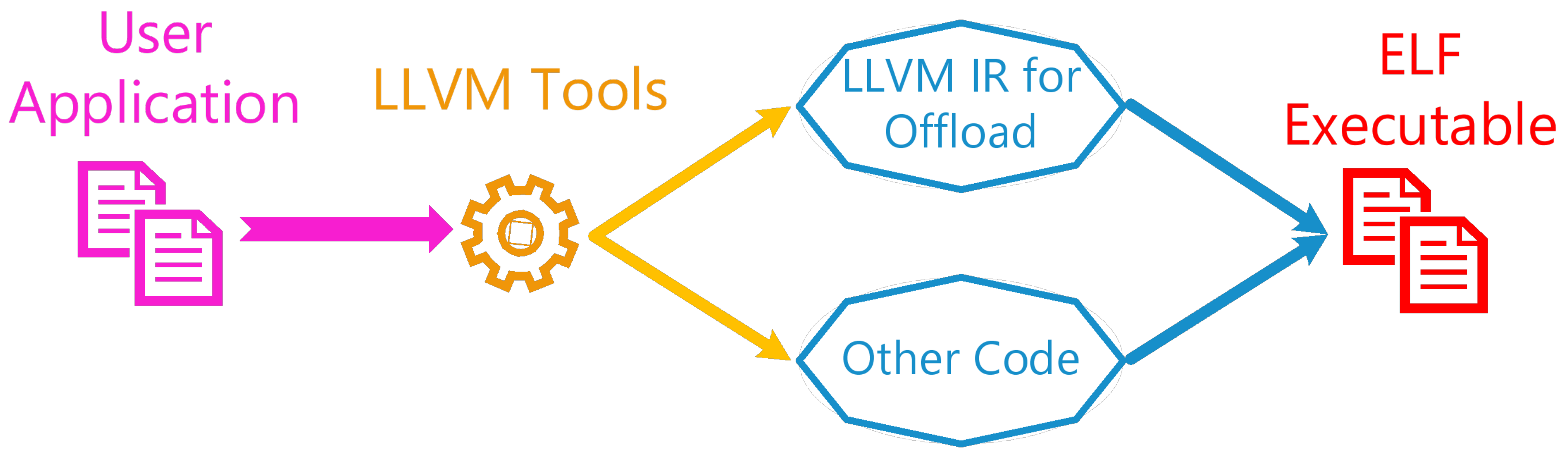
Layout constructed using a hardware description file

- Describes connectivity between PEs
- Describes allowable operations per-PE



```
0 [pe_type=ANYMEM]          0 -- 1
1 [pe_type=ANYMEM]          0 -- 4
2 [pe_type=ANYMEM]          1 -- 0
3 [pe_type=ANYMEM]          1 -- 2
4 [pe_type=ANYARITH]        1 -- 5
5 [pe_type=ANYARITH]        2 -- 1
6 [pe_type=ANYARITH]        2 -- 3
7 [pe_type=ANYARITH]        2 -- 6
8 [pe_type=ANYARITH]        3 -- 2
9 [pe_type=ANYARITH]        3 -- 7
10 [pe_type=ANYARITH]       4 -- 1
11 [pe_type=ANYARITH]       4 -- 5
12 [pe_type=ANYMEM]         4 -- 8
…                           …
```

LLVM and llyr Parsing

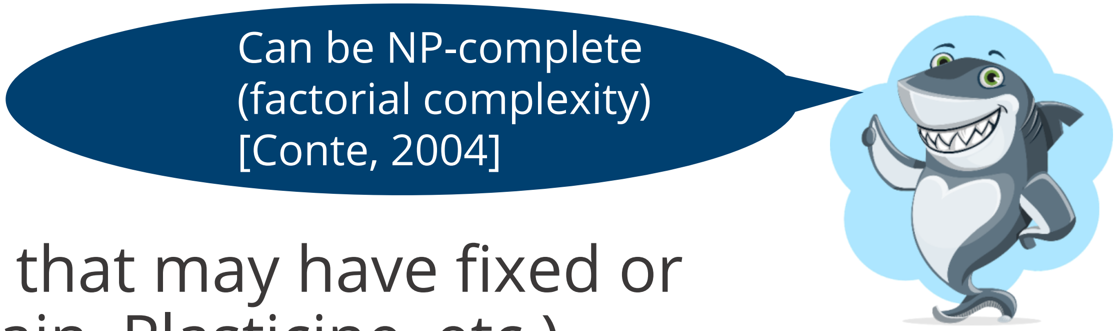


Work in progress…

Offload targets are marked in the user application – currently under study

- __attribute__ ((offload (device, 0))) myFunction()
- #pragma secret offload directive device(0)

These functions/loops will be compiled with the user code but the LLVM IR will be embedded with them in the final executable

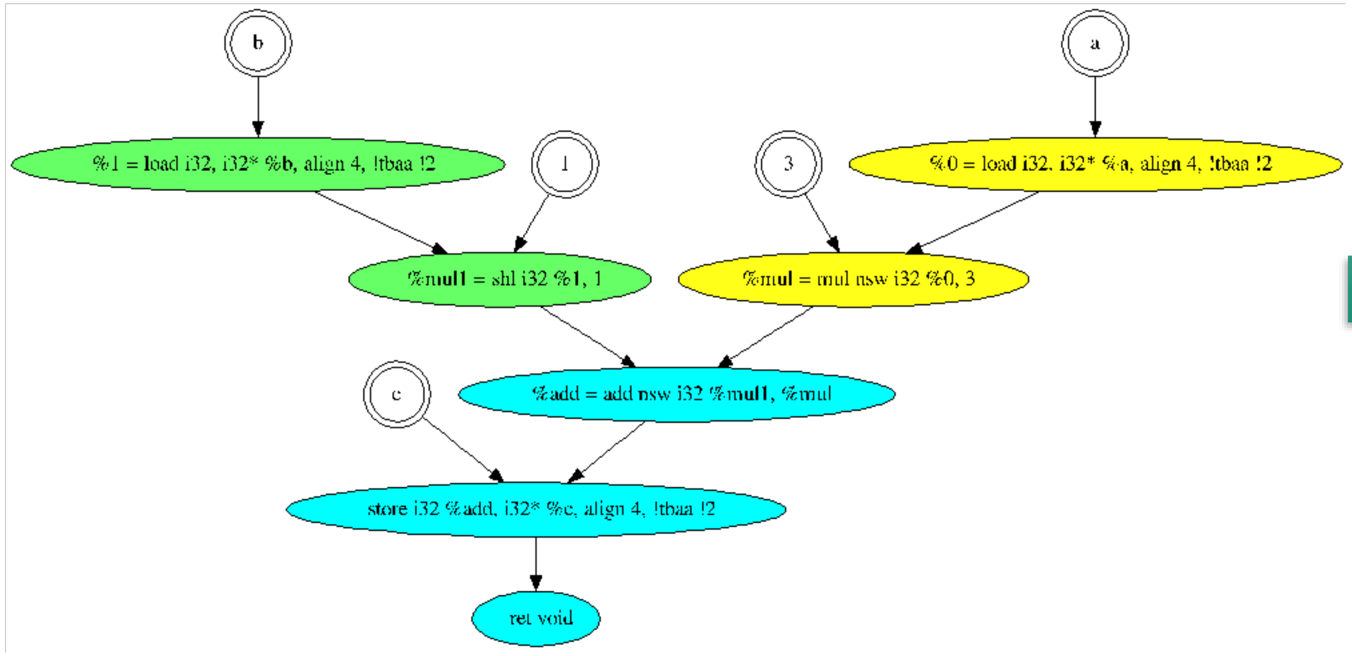


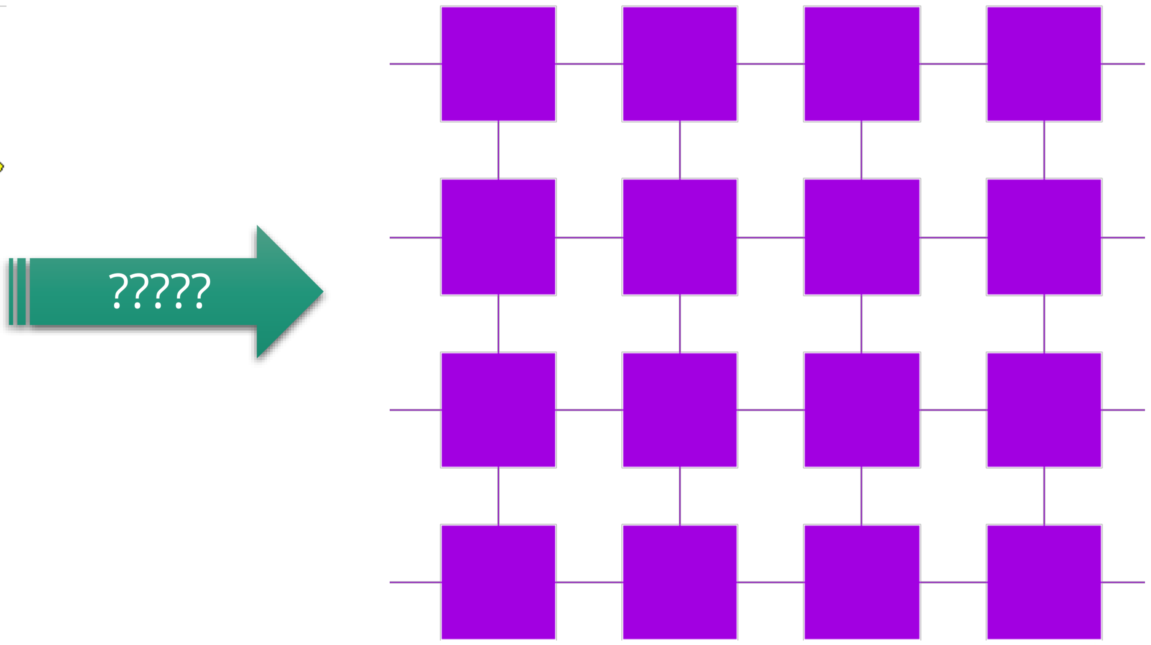
Embedding Problem

Dataflow hardware consists of a set of PEs that may have fixed or programmable connectivity (MAERI, Softbrain, Plasticine, etc.)

A user application consists of a set of operations that map to one or more PE types (add, sub, mul, etc.)

Goal: Map user application operations onto set of hardware PEs


b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void



Current Approach

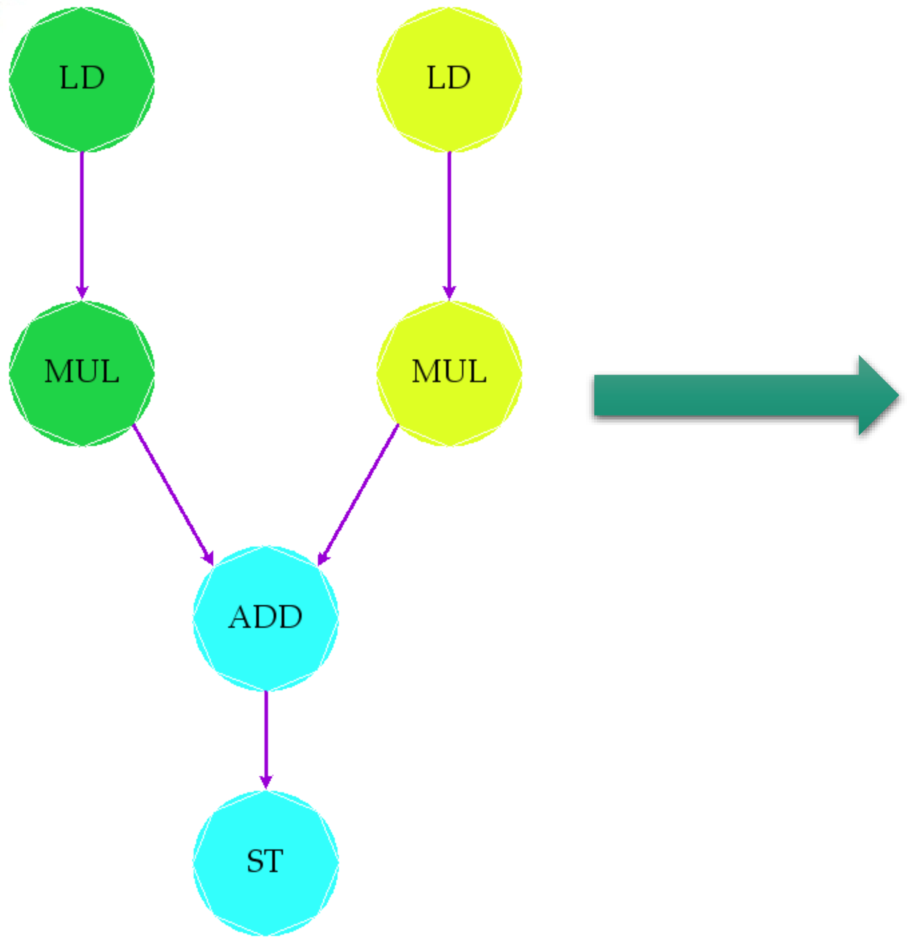
Running standalone VF3 implementation from the University of Salerno

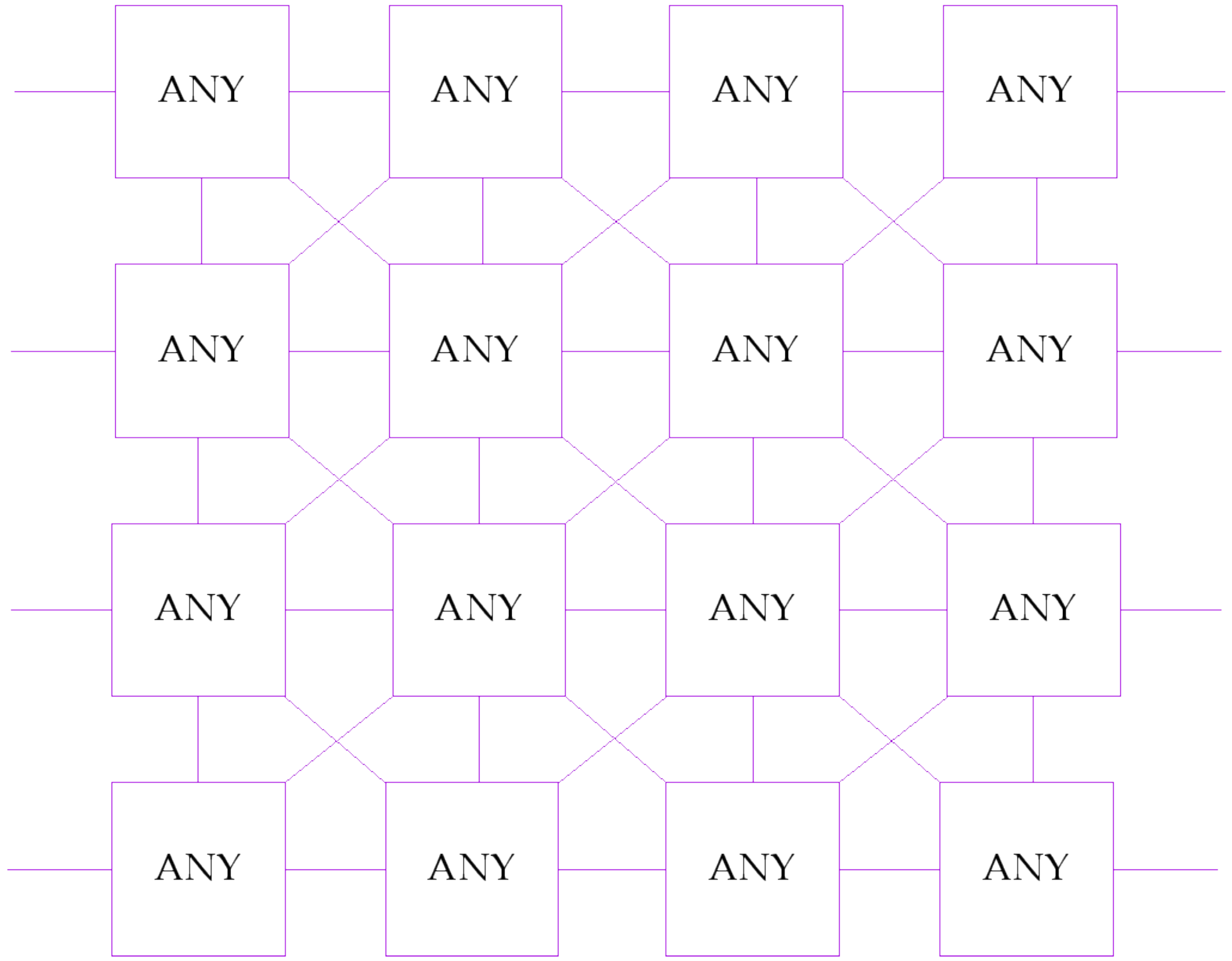
Have tested VF3 on multiple graphs (up to 512 nodes)

Current problem is that constraints can result in no valid mapping

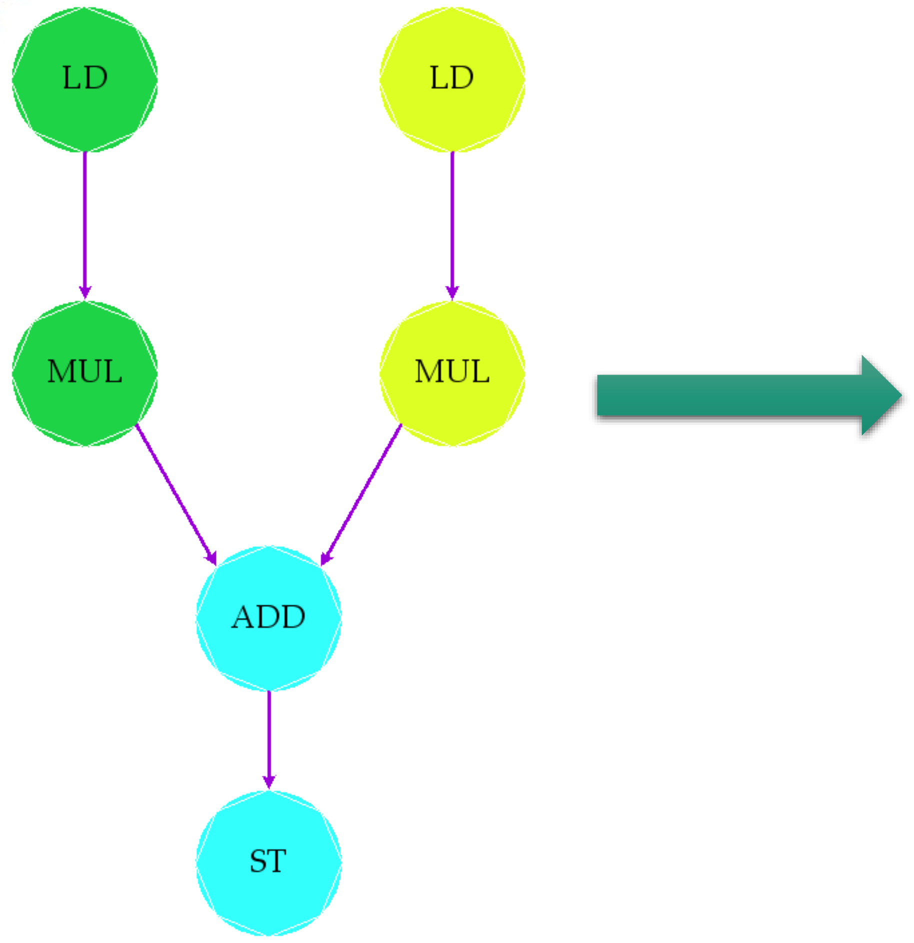
- Need a way to bypass a node
- Will require some tweaking to be able to perform lookahead in the algorithm

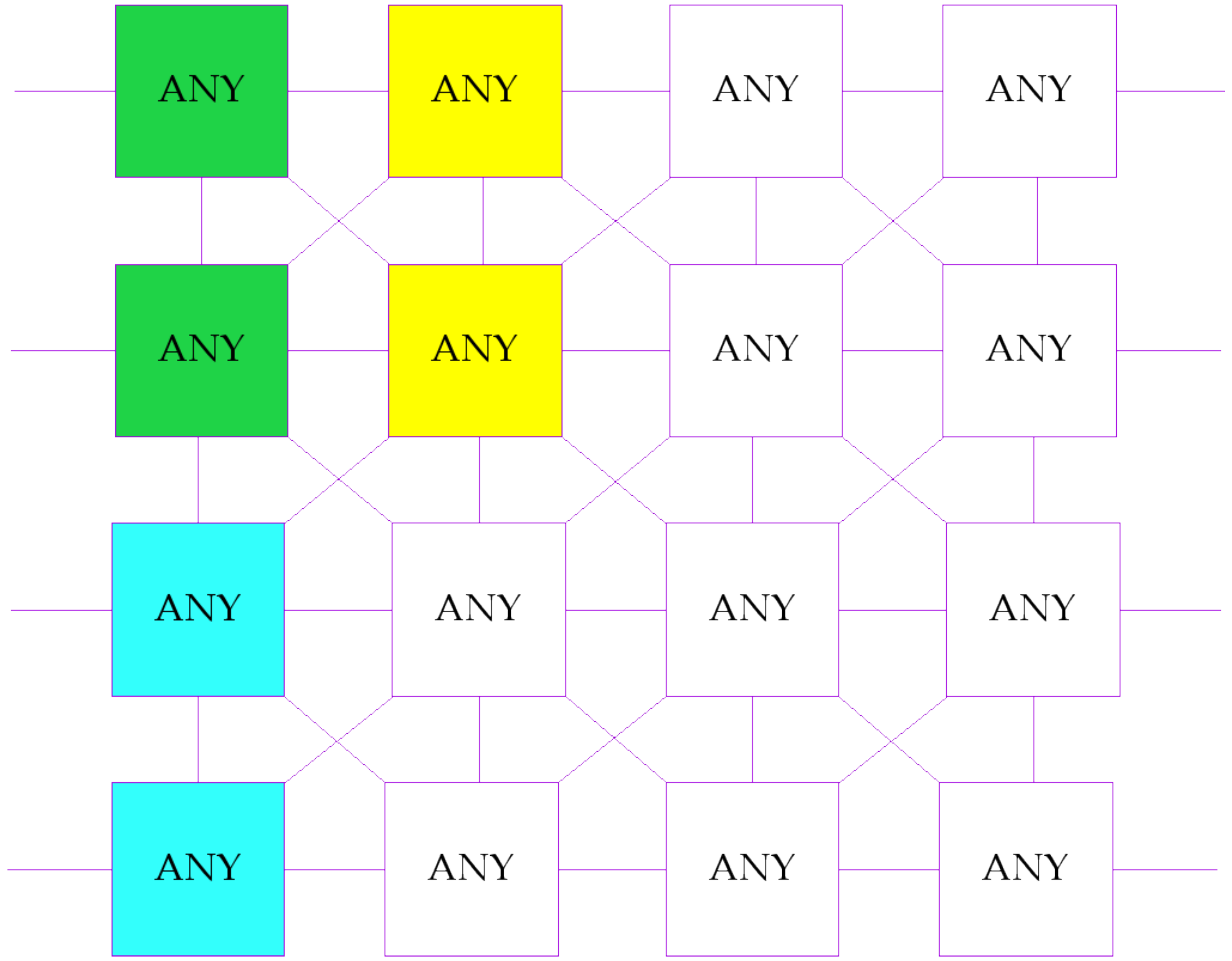
Node Attributes Don’t Matter





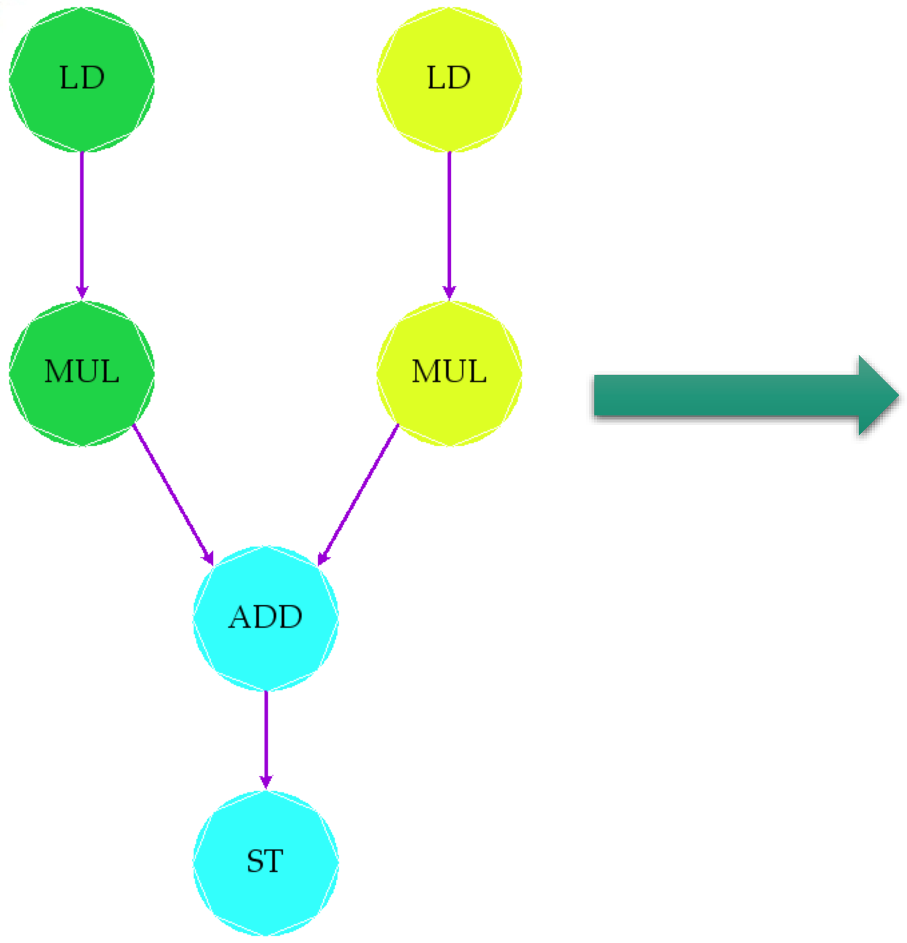
Node Attributes Don't Matter

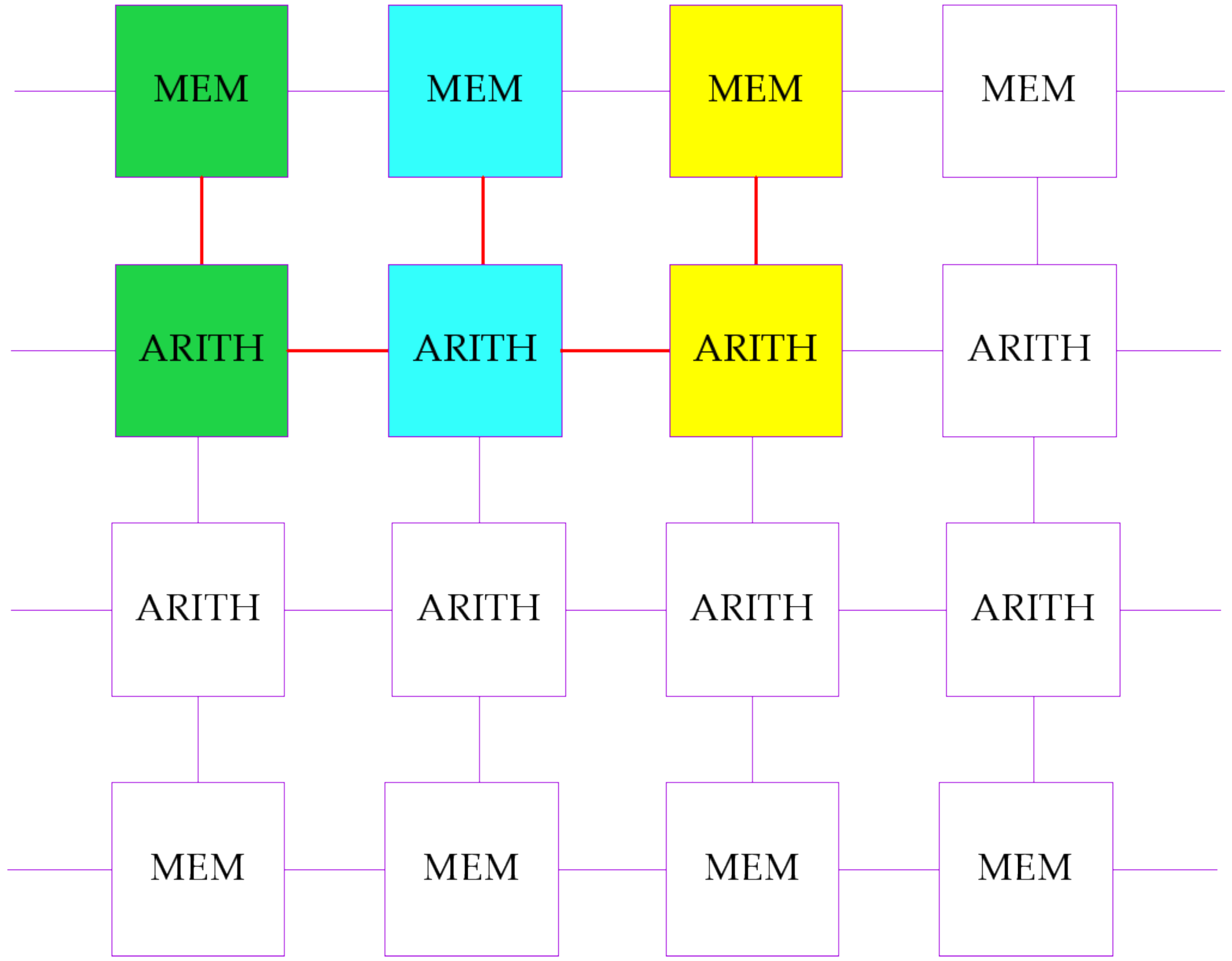




13720 possible mappings

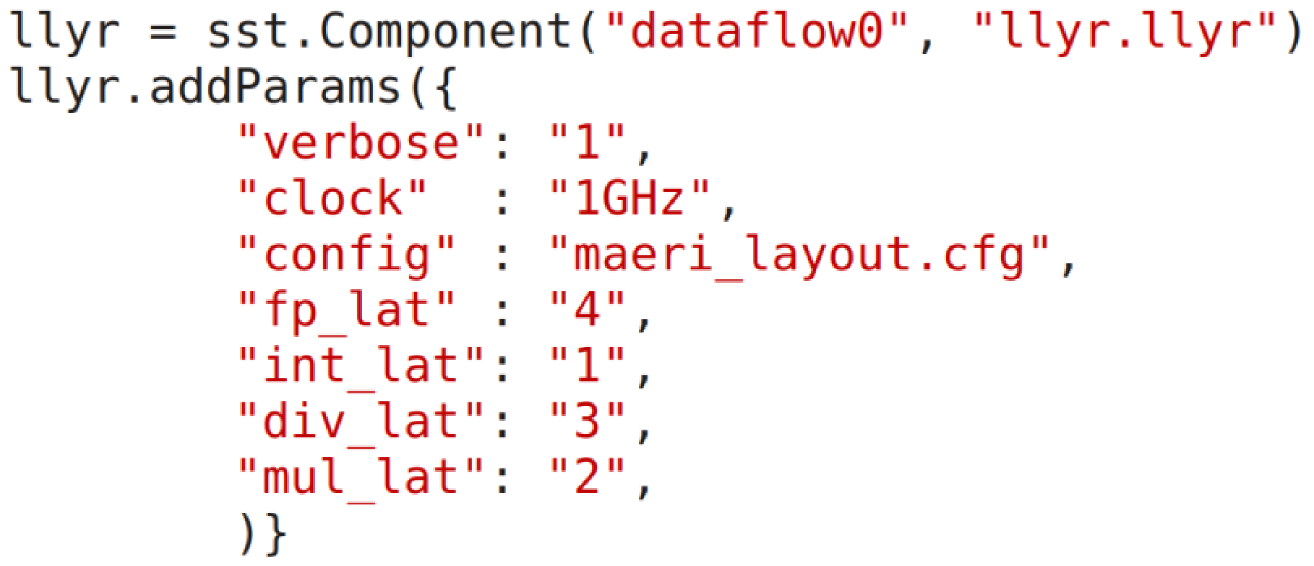
Node Attributes Matter (Kind'a Sort'a)





32 possible mappings

SST llyr Component Sample Configuration



```
llyr = sst.Component("dataflow0", "llyr.llyr")
llyr.addParams({
        "verbose": "1",
        "clock"  : "1GHz",
        "config" : "maeri_layout.cfg",
        "fp_lat" : "4",
        "int_lat": "1",
        "div_lat": "3",
        "mul_lat": "2",
        )}
```

Parameters

- clock: Operating frequency for entire device
- config: Input hardware layout
- xxx_lat: Number of cycles to complete the operation
- queue_depth: number of buffer entries
- ls_entries: number of L/S entries to process each cycle
- mapper: app_graph → hw_graph embedding

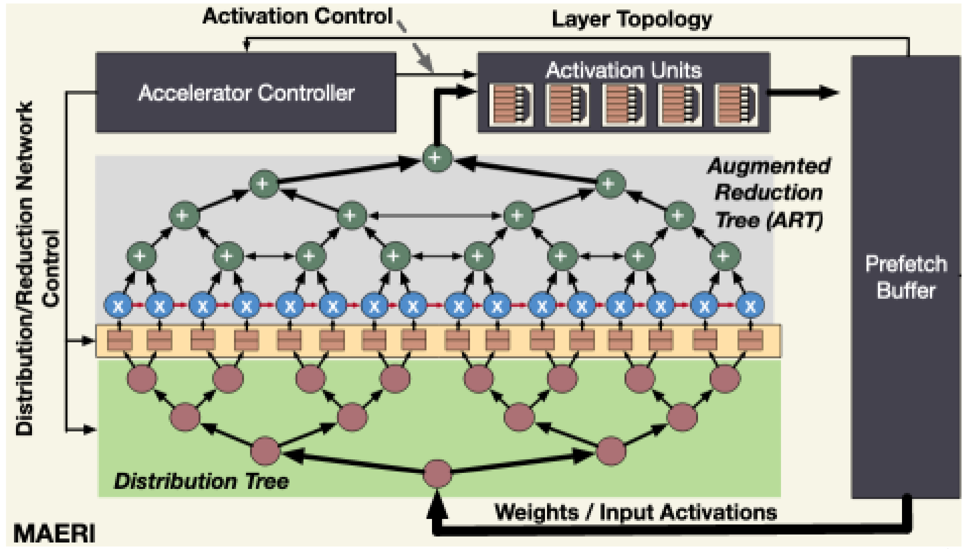
Additional parameters for standalone/testing

- application: application in LLVM IR
- mem_init: memory initialization file

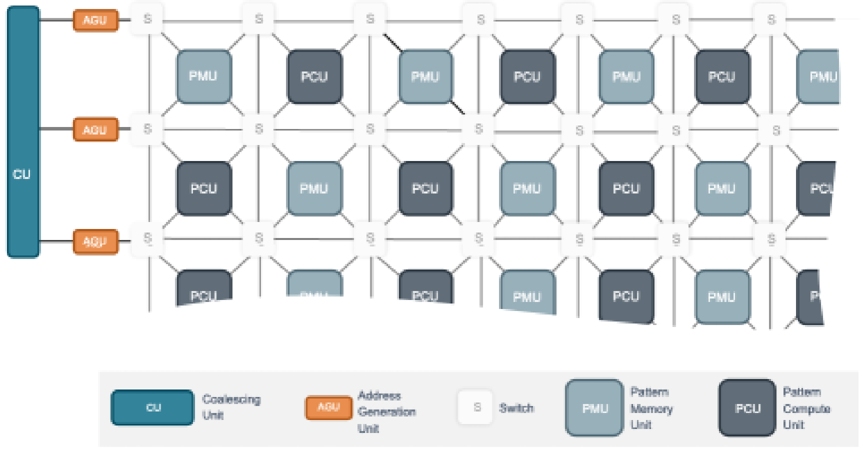
Modeling ASAs

The hardware description should allow users to emulate more constrained dataflow devices

- MAERI
  - Programmable DNN accelerator
- SambaNova?
  - Reconfigurable dataflow
- Others?
  - ASA?
  - Crossbar?



[Kwon, 2019]



[SambaNova, 2021]

Summary

Advances in material science and fabrication technologies are opening the door to the integration of exotic computing devices with traditional architectures

Dataflow and application-specific accelerators are the most promising near-term candidates under consideration

In simulation, these models can emulate a broad range of computational devices

Thank You’s…

Siva Rajamanickam, Jim Laros, and Kevin Pedretti

Thanks! Questions?

Exceptional Service In The National Interest

Sandia National Laboratories

References

M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10-14, doi: 10.1109/ISSCC.2014.6757323.

J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah and T. Nowatzki, "DSAGEN: Synthesizing Programmable Spatial Accelerators," *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 268-281, doi: 10.1109/ISCA45697.2020.00032.

T. Nowatzki, V. Gangadhan, K. Sankaralingam and G. Wright, "Pushing the limits of accelerator efficiency while retaining programmability," *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 27-39, doi: 10.1109/HPCA.2016.7446051.

Devices & Architectures

Transistor density has continued to scale well despite decreasing clock frequency
- Vendors provide more computation on a single chip
- Scaling will probably end sometime before 2030
  - 2D lithography approaching atomic scale

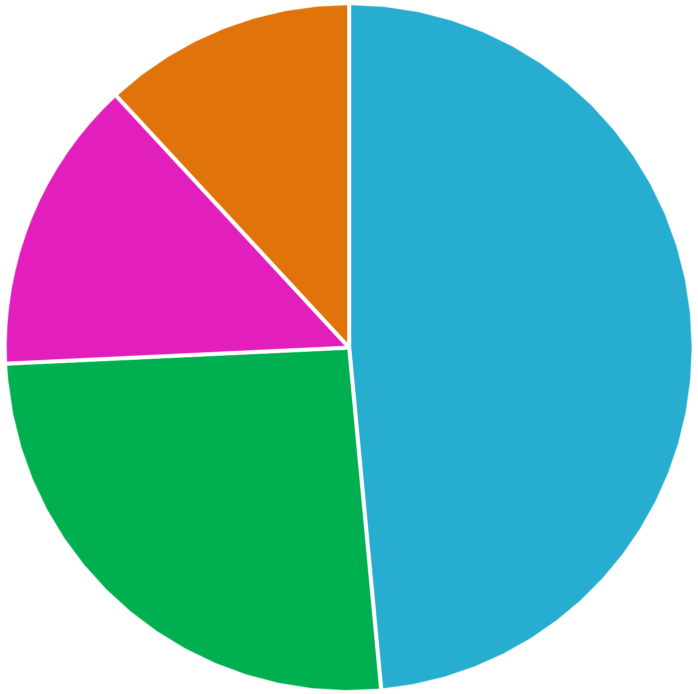
Regardless of transistor scaling, there is still the problem of resistance
- $E \propto bitRate * d$
  - Memory becomes a major performance and energy bottleneck
  - Fetching data for the cores becomes the dominant activity in terms of energy
  - Pushes programming models toward more localized data movement

Operation	Energy (nJ)
ADD	0.64
L1 → REG	1.11
L2 → REG	2.21
L3 → REG	9.80
MEM → REG	63.64
Prefetch	65.08

[Kestor, 2014]

The Power Problem

Processor Energy Breakdown



$$P = \alpha C V_{dd}^2 f \rightarrow P = \left(\frac{Energy}{op}\right)\left(\frac{op}{s}\right)$$

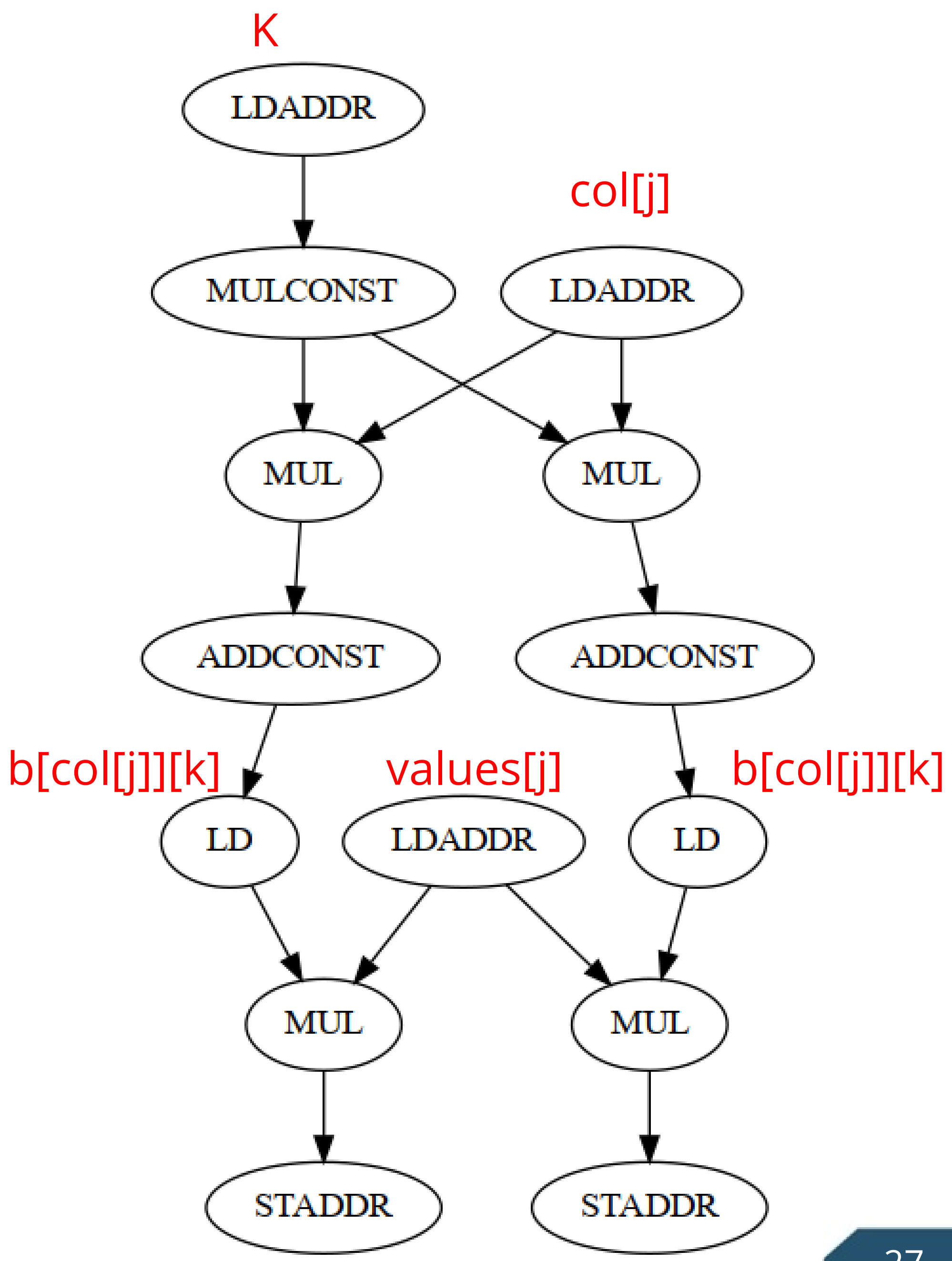
Computation does not represent the bulk of the energy!

25pJ	6pJ	39pJ	0.1pJ	70pJ
Ins Cache	Register	Control	Add	

[Horowitz, 2014]

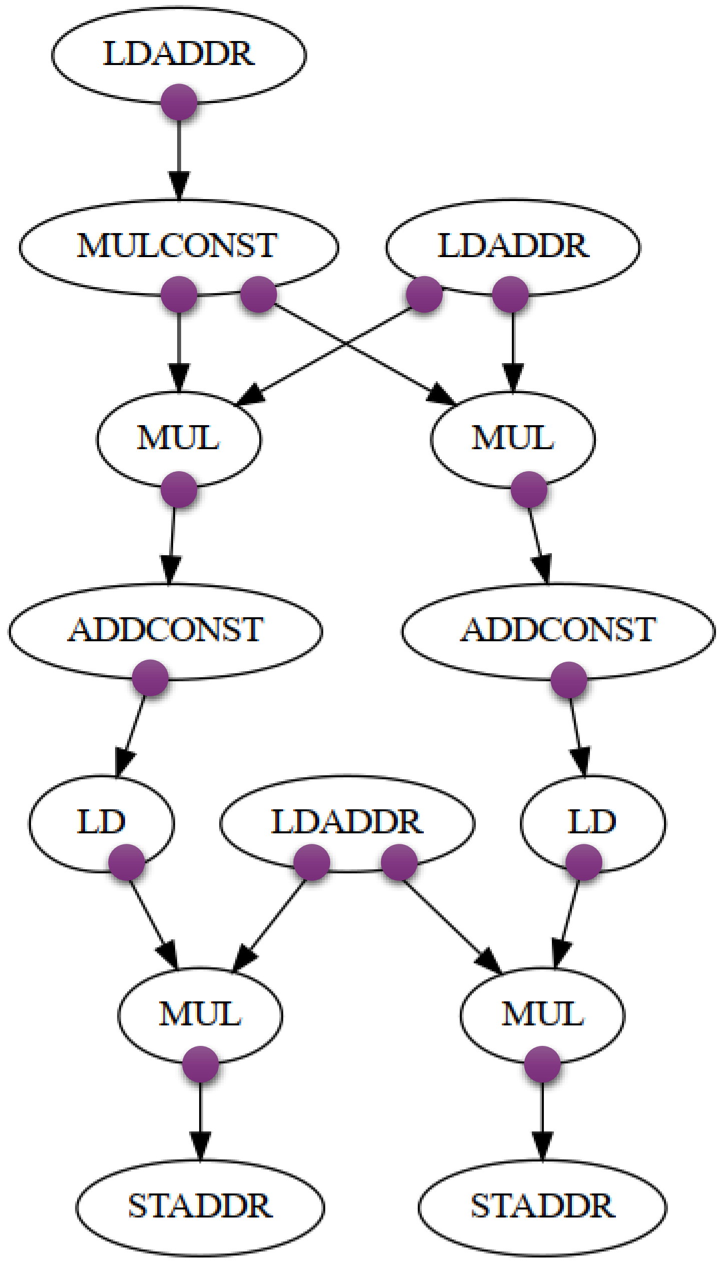
SpMM - CSR $(1 \times 3 \cdot 3 \times 2)$

```
for( uint32_t i = 0; i < M; ++i )
    for( uint32_t j = row_ptr[i]; j < row_ptr[i+1]; ++j )
      for( uint32_t k = 0; k < K; ++k )
            result[i][k] = result[i][k] + values[j] * b[column[j]][k];
```



SpMM - CSR $(1 \times 3 \cdot 3 \times 2)$

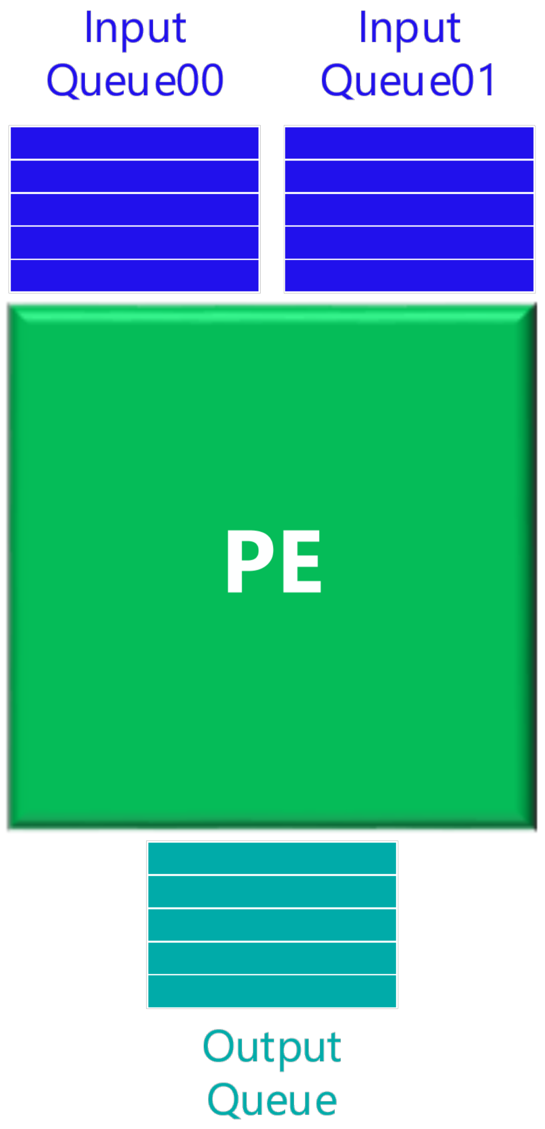
```
for( uint32_t i = 0; i < M; ++i )
    for( uint32_t j = row_ptr[i]; j < row_ptr[i+1]; ++j )
      for( uint32_t k = 0; k < K; ++k )
          result[i][k] = result[i][k] + values[j] * b[column[j]][k];
```



SST llyr Component

PEs have a compute unit, input buffers, and output buffers
- Number of buffers bounded by connectivity
- Buffer depth is configurable but is uniform for all PEs

Current PE list
- LD, ST, LD_ST
- SLL, SLR, ROL, ROR
- ADD, SUB, MUL, DIV
- FPADD, FPSUB, FPMUL, FPDIV, FPMATMUL
- BUFFER
- ANY, ANYMEM, ANYLOGIC, ANYINT, ANYARITH, ANYFP



SST llyr Component

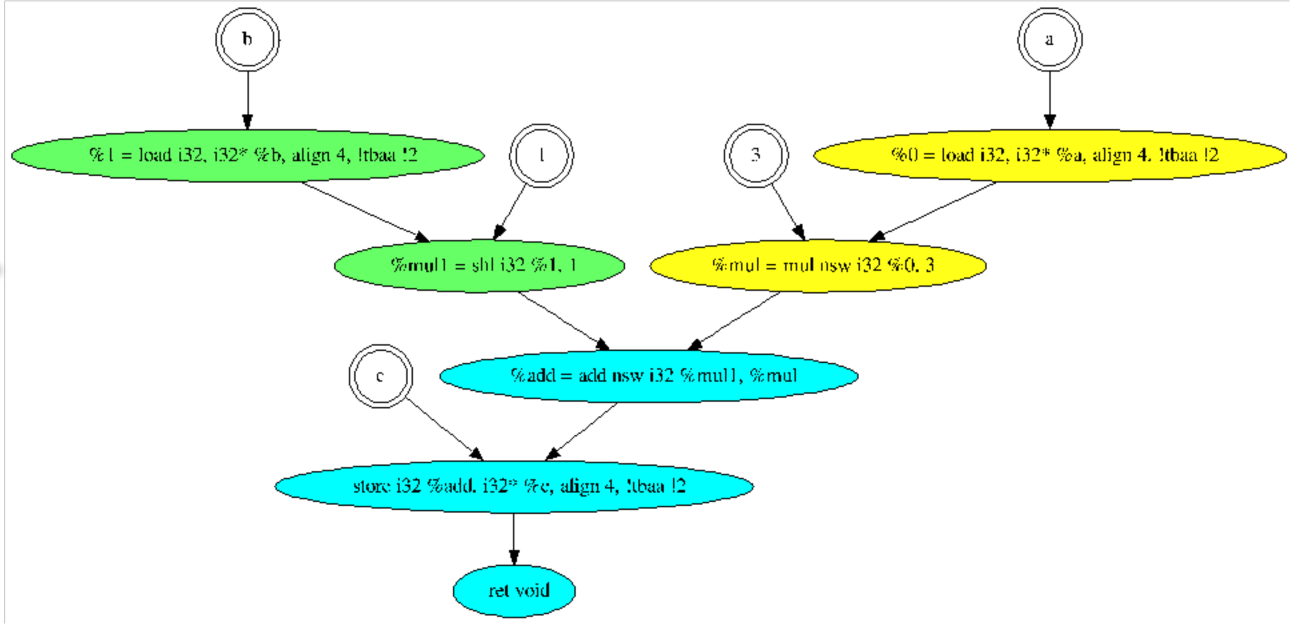
The hardware configuration controls the potential computation, but the application graph controls the actual computation

LLVM IR will be extracted from the ELF executable and passed to the llyr parser

Parser will construct a dataflow graph with control information (backward edges) from the LLVM IR

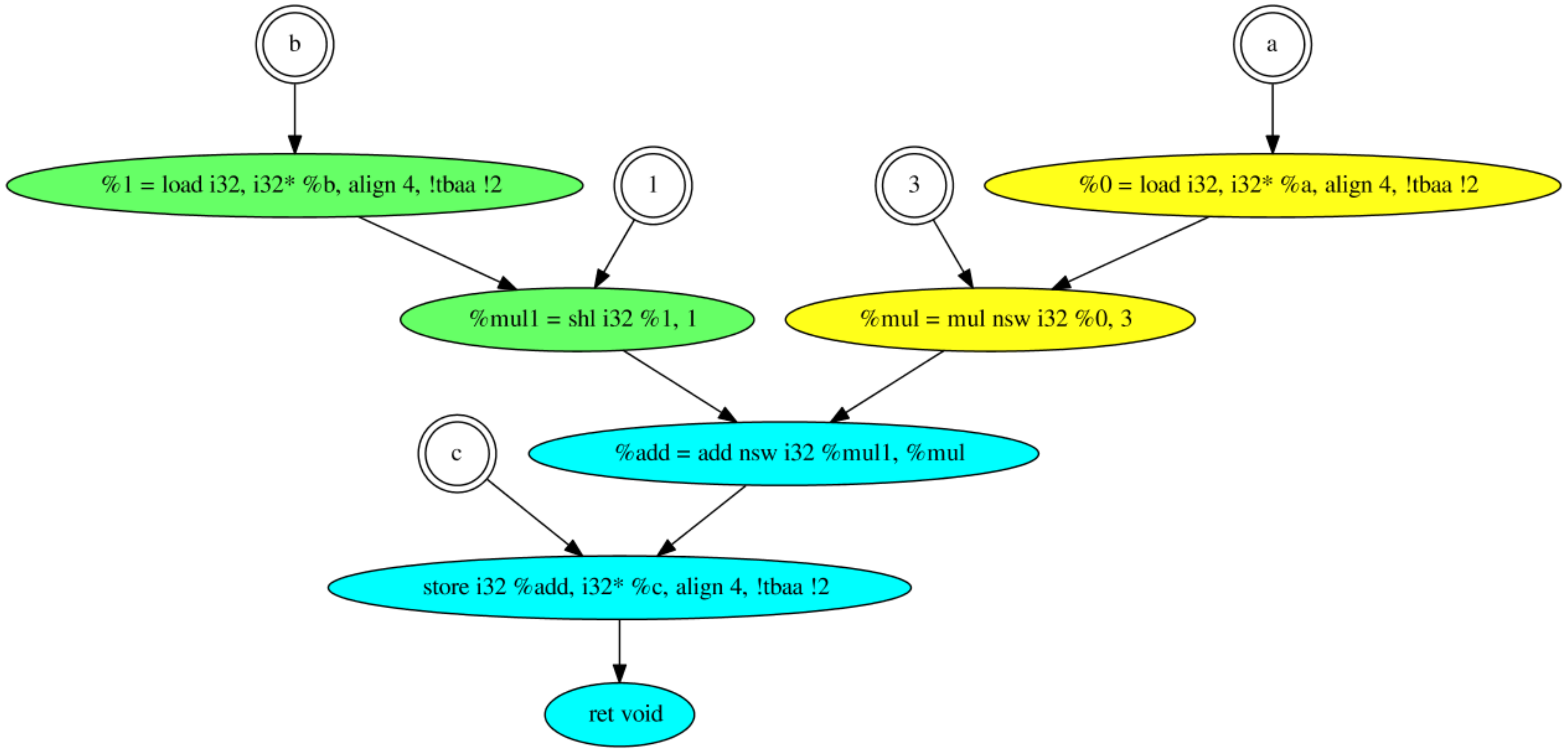
```
void  multiply_test(int* a, int *b, int* const c)
{
   int d = *a;
   int e = *b;

   int f = 3 * d;
   int g = 2 * e;

   *c = f + g;
}
```

Parser



b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void

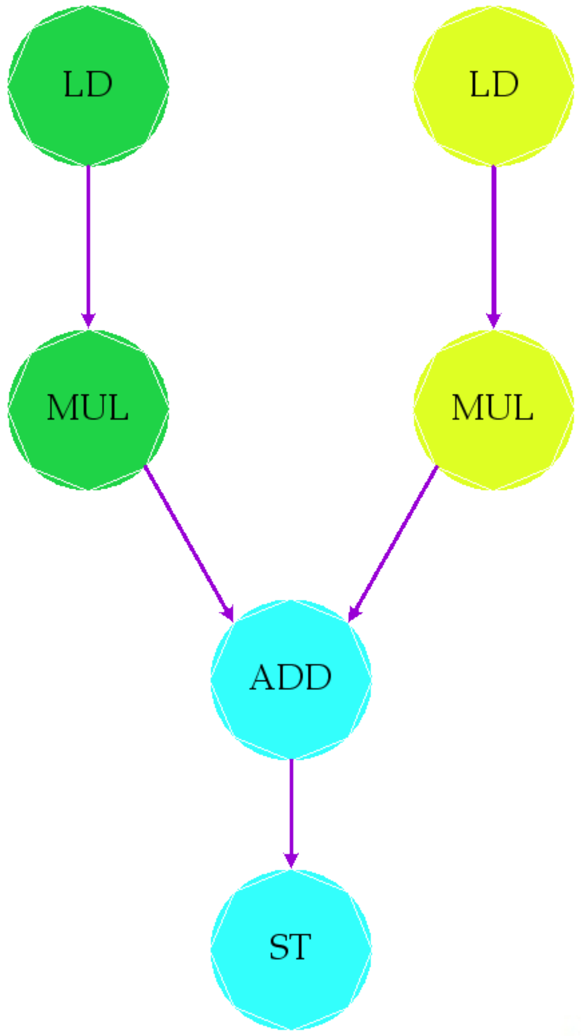
LLVM IR to AppGraph



b
a
%1 = load i32, i32* %b, align 4, !tbaa !2
1
3
%0 = load i32, i32* %a, align 4, !tbaa !2
%mul1 = shl i32 %1, 1
%mul = mul nsw i32 %0, 3
c
%add = add nsw i32 %mul1, %mul
store i32 %add, i32* %c, align 4, !tbaa !2
ret void

LLVM IR

Application Graph



LD
LD
MUL
MUL
ADD
ST