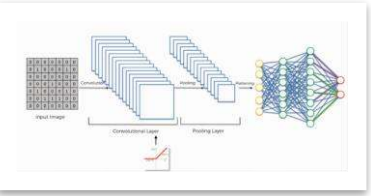
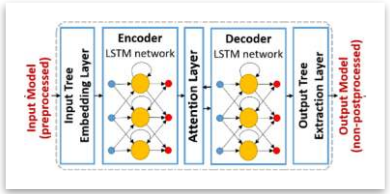
Mozart: Designing for Software Maturity and the Next Paradigm for Chip Architectures

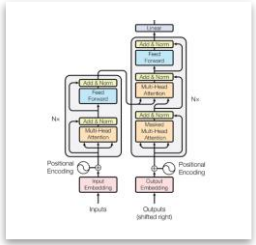
Karu Sankaralingam | UW-Madison and SimpleMachines
Tony Nowatzki | UCLA
Greg Wright, Poly Palamuttam, Jitu Khare, Vinay Gangadhar, Preyas Shah | SimpleMachines

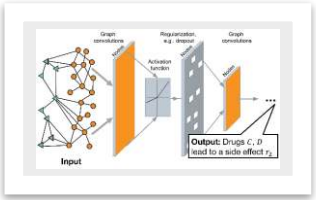
Where does AI hardware/software stand today?

1. The computational diversity needed to support AI is increasing
2. The software user experience expectations is increasing
3. GPU software maturity* is unrivalled in completeness and hence allows near complete dominance among AI industry deployment and researchers.
4. This support for model diversity is fuelling these trends and increasing GPU adoption!









Algorithm CNN LSTM Transformer Graph Neural Net ???

Time in months

Users

ebay, Airbnb, Target, aetna, KOHL'S, J.P.Morgan, NUANCE, CONDÉ NAST, J.P.Morgan, Capital One, TWO SIGMA, feedzai, 淘宝网 Taobao.com, expero

* *NVIDIA DL stack - cuDNN, TensorRT, etc.*

Is there a problem?

1. GPU chip utilization is quite low (10% of peak)
2. The software stack is turgidly tied to hand-tuned/auto-tuned libraries, resulting in ML systems getting caught in the rut.
3. Tail wagging the dog: Because only some styles of kernels achieve high performance on GPUs, ML practitioners are forced to create ridiculously large models to get new capability, rather than use better algorithms.
4. Simpler models exist but don't run fast on existing HW or SW
    a. Lottery Hypothesis shows simpler models have the same/superior learning ability.
    b. Depth-wise separable convolutions have 10X fewer ops, but provide no speedups on today's HW.

New chip architectures must have software maturity to challenge GPU dominance and serve as an alternative

Designing for Software Maturity

Goal: automated realization of the compiler and software stack utilizing program-synthesis vs *human coding of high-perf libraries or non-automated/imperative compilers*.

Benefits:

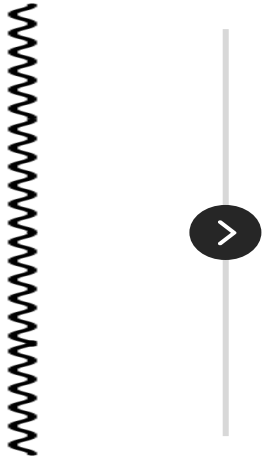
1. Allows day-0 SW maturity
2. Avoiding the pit-fall of building software after silicon arrives
3. Overcome the problem of SW dev cycle that far exceeds the HW dev cycle and cannot match the rate of application change
4. Allows co-design of the architecture to match needs of software, because software stack exposes what applications are doing

Other IRs and Compiler:

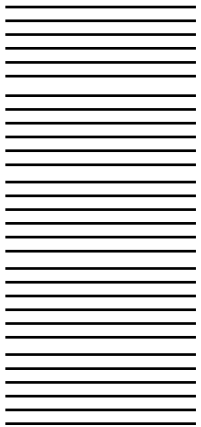
1. TVM, GLOW and other IRs solve part of the problem
2. Don’t get sufficiently down to object-code or sufficiently up the DL-stack to serve as turn-key deployment

Composable Computing: An Architecture Paradigm That Allows SW Maturity

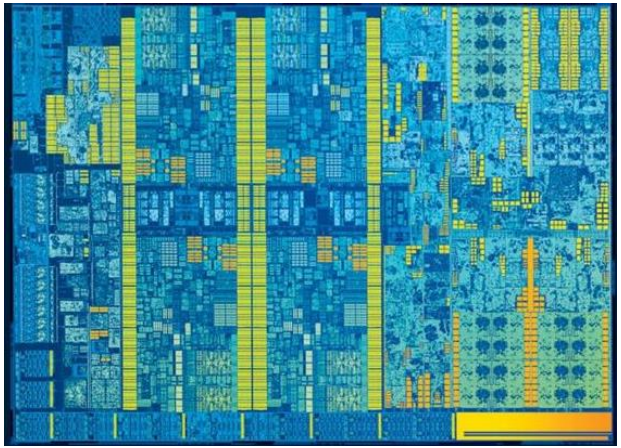
Overview of CPU Execution



Source code has rich semantic information



Machine code



Compute resources devoted to overhead instead of app perf.

Composable Computing Paradigm

Source code has rich semantic information

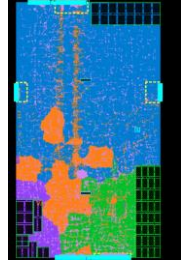
Behavior Decomposition Compiler

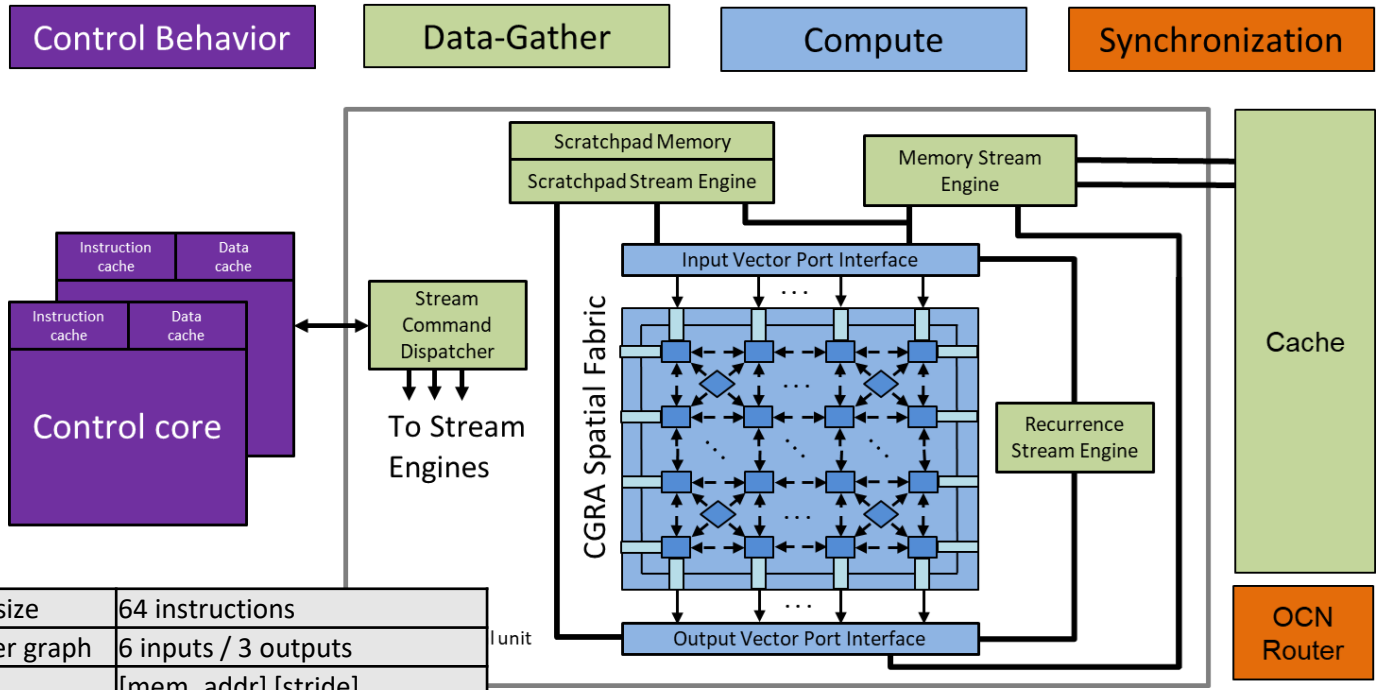
Machine code

Composable Architecture

Application Compute: 85%
Overhead: 15%

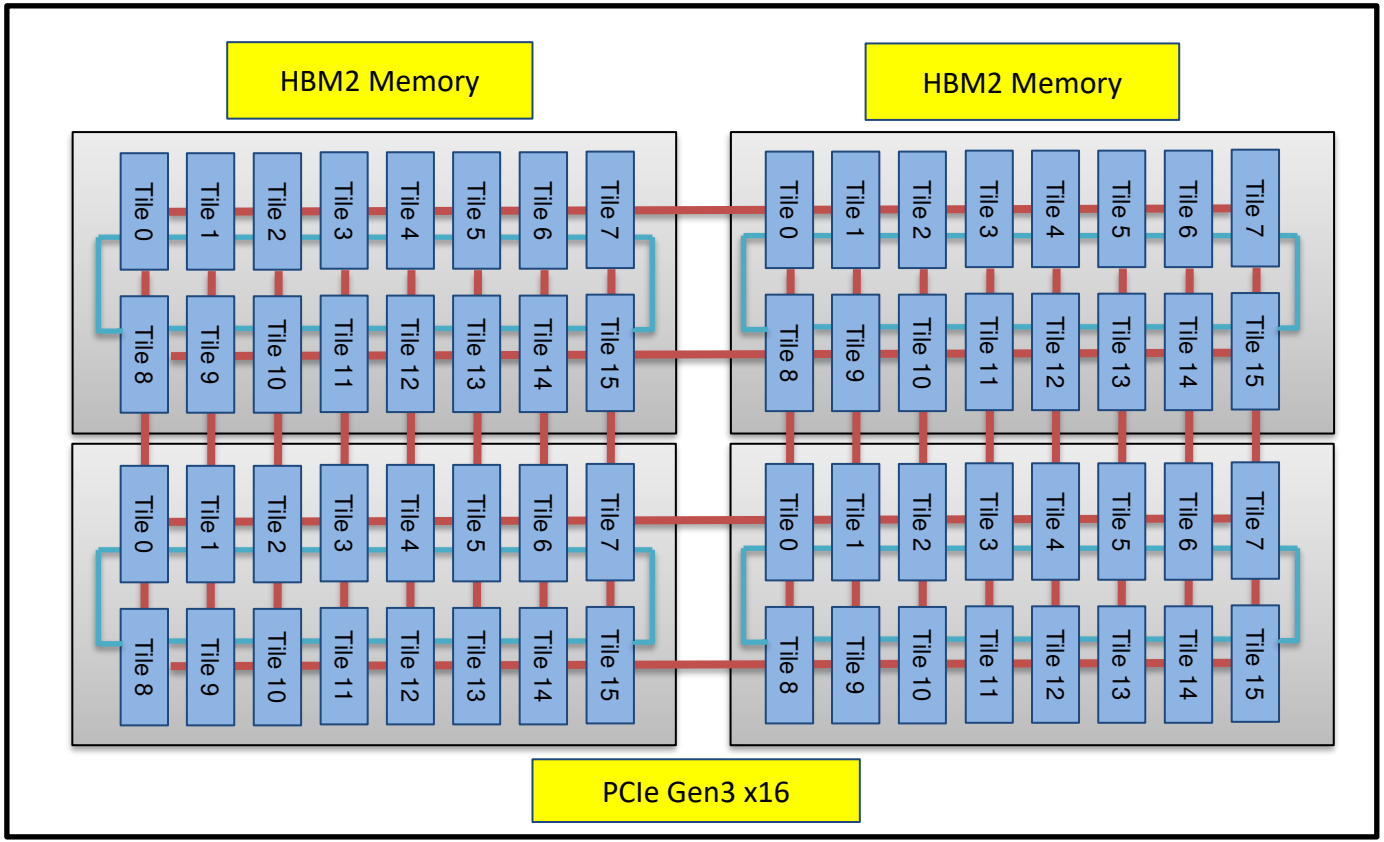
Architecture Overview and Basic Tile



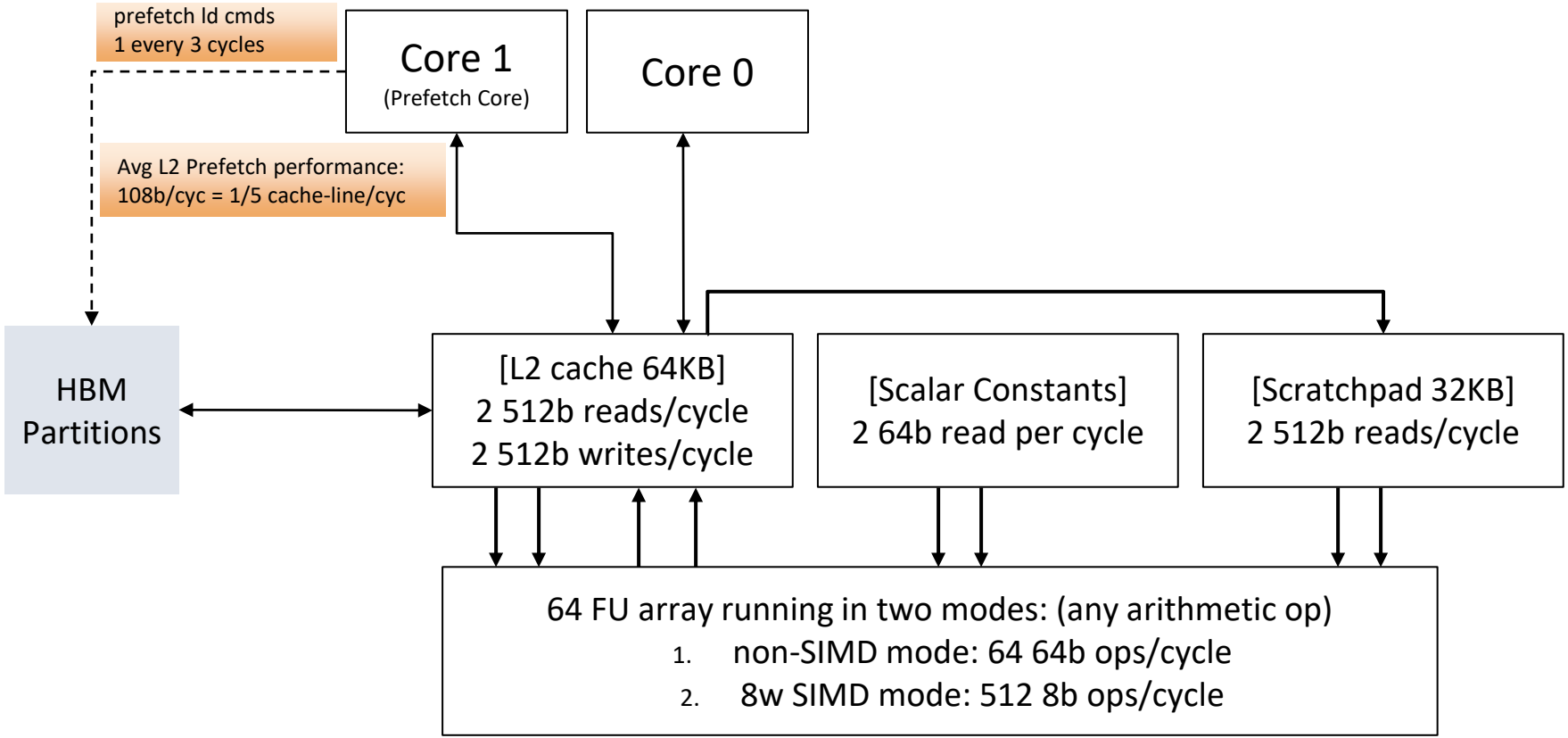


Maximum graph size	64 instructions
inputs/outputs per graph	6 inputs / 3 outputs
Stream Read	[mem_addr] [stride] [access_size] [num_strides] [input_port]
Stream Write	[mem_addr] [stride] [access_size] [num_strides] [output_port]

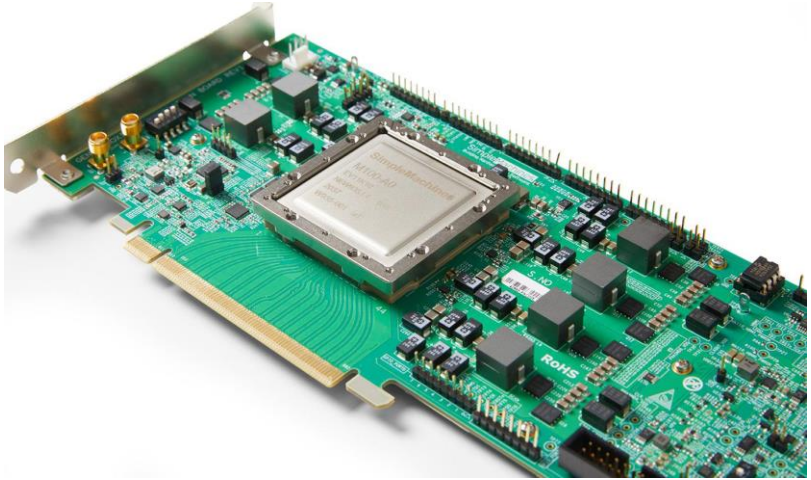
SoC Organization



Architecture Specification: Performance View



Mozart Chip, Board, System



Technology Node	TSMC 16FFC
Operating Frequency	1 GHz
Die Area	404 mm^2
Package	45mmx45mm FCBGA
Peak INT8 performance	48 TOPS
DRAM Channels	2x HBM2
DRAM Bandwidth	512 GB/s
Host Interface	PCIe Gen3 x16

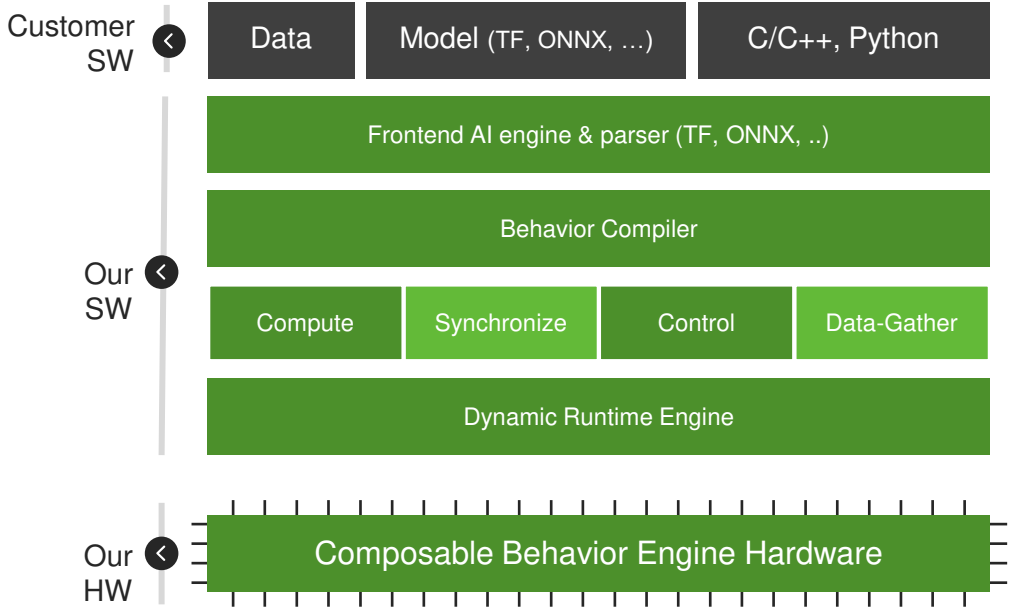
Deep Learning Performance

	Mozart @1GHz, 16nm, 75W PCIe Card	Bach (7nm) (projection)
Resnet	2.6X	6.6X
BERT	1.2X	3.1X
SSD-Resnet	1.0X	2.7X
RNN-T*	5.8X	15.1X
DLRM	0.3X	1.8X

** RNN-T includes many subtelities and optimizatin opportunities on batch-size in a sample*

Relative Inf/Sec/Watt to NVIDIA A100

HW/SW Stack



High-level Attributes

Programmability

Algorithm independence

Physical attributes

Physical scalability

Modularity

Efficiency

Software

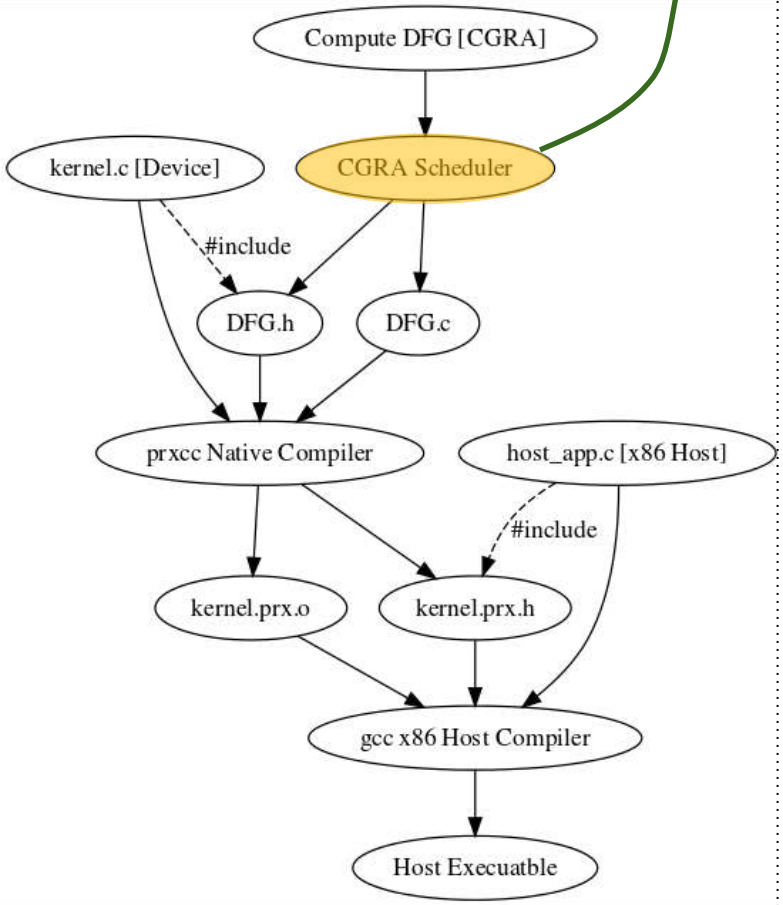
C/C++ and Python SDK

Resnet, BERT, RNN-T, DLRM

Universal AI/ML accelerator

Program Synthesis and SW Maturity in Mozart

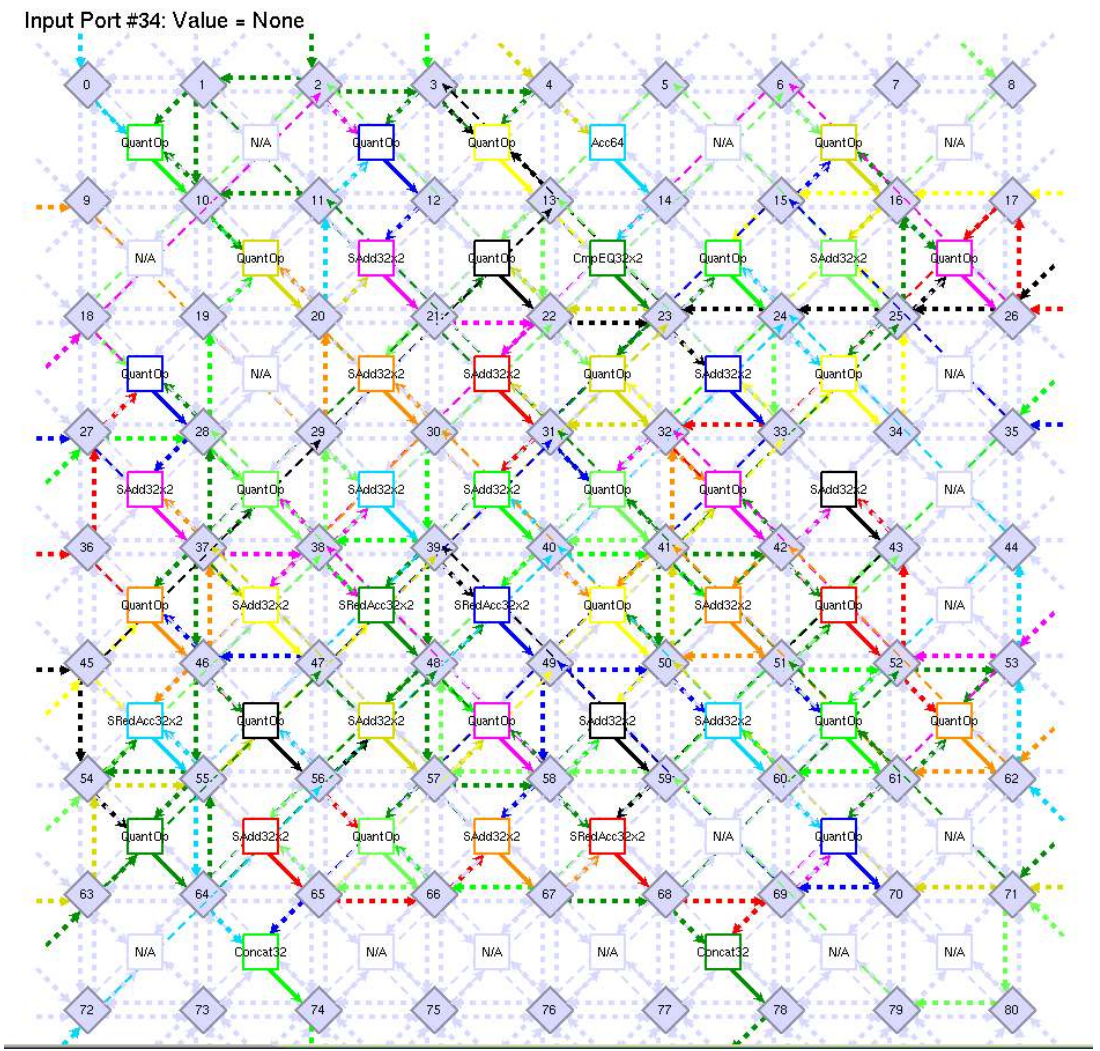
Compiler Flow




```
if in("MRT", stages)
    @constraint(m, opposite_calc_l_used[v_i=v, l_i=l], sum(Mel[e_i, l_i]
                   for e_i=e if Gve[v_i, e_i]) >= Mvl[v_i, l_i])

    @constraint(m, calc_l_used2[v_i=v, e_i=e, l_i=l; Gve[v_i, e_i]],
                   Mel[e_i, l_i] <= Mvl[v_i, l_i])

    @constraint(m, oneEperL[l_i=l], sum(Mvl[v_i, l_i] for v_i=v) <= 1);

    @constraint(m, source_mapping_p[v1_i=v, e_i=e, n_i=n; Gve[v1_i, e_i]],
        sum(Mel[e_i, l_i] for l_i=l if Hnl[n_i, l_i]) == Mn[v1_i, n_i] + PTen[e_i, n_i])

    @constraint(m, dest_mapping_p[e_i=e, v2_i=v, n_i=n; Gev[e_i, v2_i]],
        sum(Mel[e_i, l_i] for l_i=l if Hln[l_i, n_i]) == Mn[v2_i, n_i] + PTen[e_i, n_i]);

    function no_fu_router_loop(nf, rf, lf, ef, Hnlf, Hlrf, Hrlf, Hlnf, Melf)
        for l1_i=lf, l2_i=lf, e_i=ef
            for n_i=nf, r_i=rf
                if Hnlf[n_i, l1_i] && Hlrf[l1_i, r_i] && Hrlf[r_i, l2_i] && Hlnf[l2_i, n_i]
                    @constraint(m, Melf[e_i, l1_i] + Melf[e_i, l2_i] <= 1)
                end
                break
            end
        end
    end

    no_fu_router_loop(1:0, 1:0, 1:0, 1:0, Hnl, Hlr, Hrl, Hln, Mel)

    no_fu_router_loop(n, r, l, e, Hnl, Hlr, Hrl, Hln, Mel)

    @constraint(m, incoming_links[e_i=e, r_i=r],
        sum(Mel[e_i, l_i] for l_i=l if Hlr[l_i, r_i]) == sum(Mel[e_i, l_i]
                    for l_i=l if Hrl[r_i, l_i]));

    @constraint(m, outgoing_links[e_i=e, r_i=r],
        sum(Mel[e_i, l_i] for l_i=l if Hlr[l_i, r_i]) <= 1);
```

DFG MATMUL

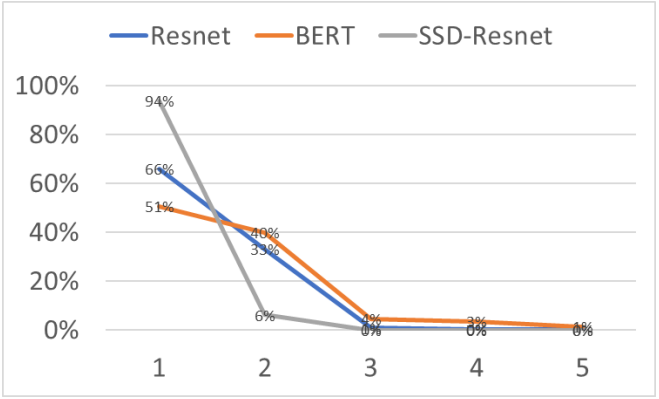
```
quant6x6x64(N0[6], W0[6], N1[6], W1[6], reset, Z -> out0, out1) {

  reset_counter = Acc64(0x04000000, reset, reset);
  reset_delayed = CmpEQ32x2(reset_counter, 0xffffffff04000001);

  NW000 = QuantOp(N0[0], W0[0], Z);
  NW001 = QuantOp(N0[1], W0[1], Z);
  NW002 = QuantOp(N0[2], W0[2], Z);
  NW003 = QuantOp(N0[3], W0[3], Z);
  NW004 = QuantOp(N0[4], W0[4], Z);
  NW005 = QuantOp(N0[5], W0[5], Z);
  AS0 = SAdd32x2(NW000, NW001);
  AS1 = SAdd32x2(NW002, NW003);
  AS2 = SAdd32x2(NW004, NW005);
  AT0 = SAdd32x2(AS0, AS1);
  AU0 = SRedAcc32x2(AT0, AS2, reset_delayed);

  NW100 = QuantOp(N0[0], W1[0], Z);
  NW101 = QuantOp(N0[1], W1[1], Z);
  NW102 = QuantOp(N0[2], W1[2], Z);
  NW103 = QuantOp(N0[3], W1[3], Z);
  NW104 = QuantOp(N0[4], W1[4], Z);
  NW105 = QuantOp(N0[5], W1[5], Z);
  AS3 = SAdd32x2(NW100, NW101);
  AS4 = SAdd32x2(NW102, NW103);
  AS5 = SAdd32x2(NW104, NW105);
  AT1 = SAdd32x2(AS3, AS4);
  AU1 = SRedAcc32x2(AT1, AS5, reset_delayed);

  out0 = Concat32(AU0, AU1);
```



Software Capability

Complete deep-learning compiler including model characterization, graph optimization (fusion, splitting, rewriting), quantization (including quantization on non-linear operators like Gelu, Softmax, Erf), tensor-mapping to memory, backend optimized object code generation, and lightweight high-speed custom software runtime.

Application	Original trained FP32 model		Optimized & quantized Mozart model	
	Total Ops	#uniq Ops	Total Ops	#uniq Ops
Resnet50	461	14	84	12
SSD-Resnet34	3320	46	111	10
BERT	757	24	322	17
SSD-Mobilenet	4104	52	91	7
3D-Unet	587	19	100	15
DLRM	47	10	19	11



Application	Compiler runtime to transform FP32 model
Resnet50	8 seconds
SSD-Resnet34	19 seconds
BERT	29 seconds
DLRM	3 seconds

Day-0 Software Maturity is Necessary for all Future Chips

1. Architecture **definition** is the key challenge for future chips and new architectures **must allow day-0 SW maturity**
    a. Can you bootstrap software for that architecture (before/while) defining it?
    b. Can you define an architecture that is correct and meets some figures-of-merit requirements?
2. Chip implementation of a defined architecture is deterministic
    a. But includes about 8 months of exposure: 4 months for “final” physical design, 4 months manufacture, 1.5 months bringup
3. Program Synthesis and Auto-generation of SW stacks is necessary for all future chips
4. Mozart and Composable Computing Paradigm is a compelling path forward

The Key Research Ideas

1) Spatial compiler that was retargetable to “any” spatial hardware. PLDI 2013 paper. Distinguished Paper award, CACM Research Highlights Nomination. “A General Constraint-centric Scheduling Framework for Spatial Architectures”
2) Five behaviors that capture hardware and software interactions. HPCA 2016 paper. IEEE Micro Top Picks. “Pushing the Limits of Accelerator Efficiency While Retaining General-Purpose Programmability”
3) Hybrid Von-Neuman Dataflow and Stream-Dataflow: Practical blend of above two ideas with a concrete arch, microarchitecture and small enough design. ISCA-2015, ISCA-2017. Top Picks, CACM Research Highlights.
“Heterogeneous Von Neumann/Dataflow Microprocessors”
“Stream-Dataflow Acceleration”

Thank You