Qualcomm

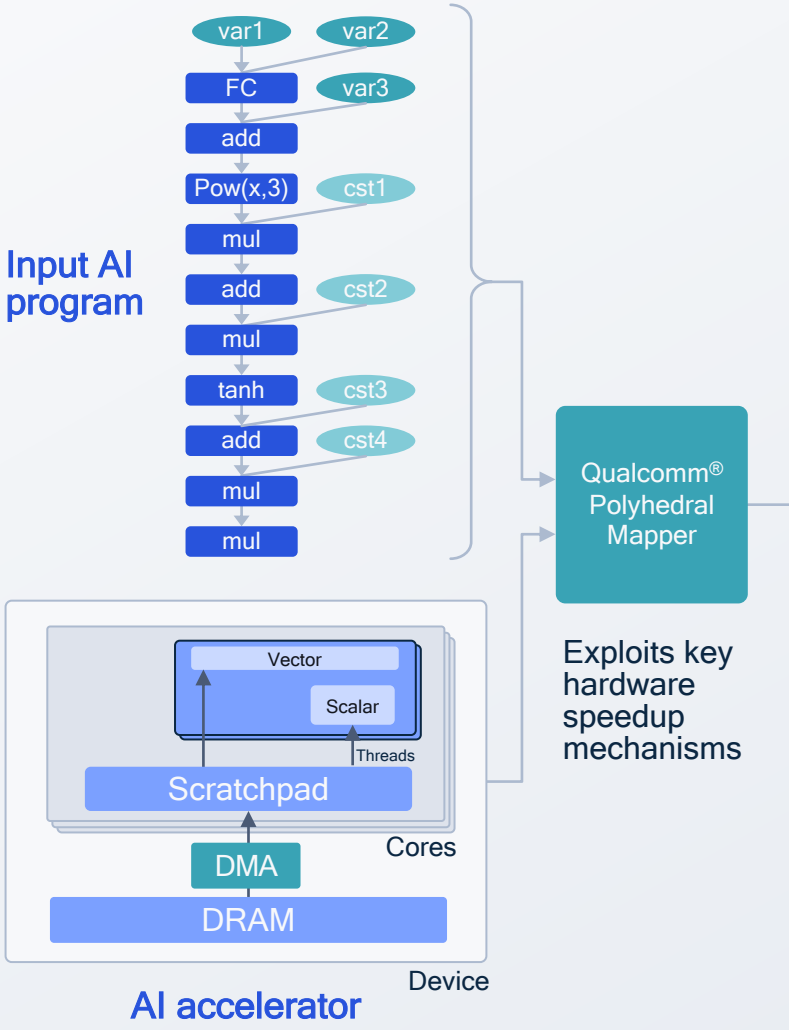
May 9, 2023

Advanced compilation technology for the AI stack

Benoît Meister

Principal Engineer
Qualcomm Technologies, Inc.

@QCOMResearch



Optimized code (sample)

```
void subgraph_3_1(float (*__restrict__ var)[1024],
   float (*__restrict__ var_2_)[1024],
   float*__restrict__ var_3_,
   float (*__restrict__ T_multiply)[4096])
{
   /* [var decls] */
   _t1_1 = rspmd_core_id(0);
   _t2 = rspmd_thd_id(0);
   var_1 = (float (*)[1024])static_alloc(spad, 0, 52ul, 131072ul) /* var_1 */;
   /* [more static allocs] */
   if (_t1_1 <= 11) {
      if (_t2 <= 0) {
         int i;
         for (i = 0; i <= 1; i++) {
            int j;
            int j_1;
            int _t3;
            int _t4;
            rspmd_dma_get((void const volatile*)(var[32 * _t1_1] + 0), (void
               volatile*)(var_1[32 * _t1_1 + -32 * _t1_1] + 0), (unsigned long)
               (unsigned int)4096, (long)4096, (long)4096, (unsigned long)(
               unsigned int)32, 0);
            rspmd_dma_get((void const volatile*)(var_2_[2048 * i] + 0), (void
               volatile*)(var_2__1[-2048 * i + 2048 * i] + 0), (unsigned long)(
               unsigned int)4096, (long)4096, (long)4096, (unsigned long)(
               unsigned int)2048, 0);
            rspmd_dma_get((void const volatile*)(var_3_ + 2048 * i), (void
               volatile*)(var_3__1 + (-2048 * i + 2048 * i)), (unsigned long)(
               unsigned int)8192, (long)0, (long)0, (unsigned long)(
               unsigned int)1, 0);
            rspmd_dma_wait(0);
            rspmd_thd_barrier();
```

```
   for (j = 0; j <= 15; j++) {
      int k;
      for (k = 0; k <= 127; k++) {
         int i1;
         rv_vstore_f32t16(T_dense_comp1[_t2] + 0, rv_bcast_f32t16(0.0f));
         for (i1 = 0; i1 <= 1023; i1++) {
            rv_vstore_f32t16(T_dense_comp1[_t2] + 0, rv_fadd_f32t16(
               rv_vload_f32t16(T_dense_comp1[_t2] + 0), rv_fmul_f32t16
               (rv_vload_s_f32t16(var_1[j + 16 * _t2] + i1, 0l),
                rv_vload_s_f32t16(var_2__1[16 * k + 0] + i1, 1024l))));
         }
         rv_vstore_f32t16(T_add_comp0 + 0, rv_fadd_f32t16(
                  rv_vload_f32t16(T_dense_comp1[_t2] + 0),
                  rv_vload_f32t16(var_3__1 + (16 * k + 0))));
         rv_vstore_f32t16(T_power_comp0 + 0, rv_fpow_f32t16(
                  rv_vload_f32t16(T_add_comp0 + 0),
                  rv_bcast_f32t16(3.0f)));
         rv_vstore_f32t16(T_multiply_2__comp0 + 0, rv_fmul_f32t16(
                  rv_bcast_f32t16(0.04471499845f),
                  rv_vload_f32t16(T_power_comp0 + 0)));
         rv_vstore_f32t16(T_add_2__comp0 + 0, rv_fadd_f32t16(
                  rv_vload_f32t16(T_add_comp0 + 0),
                  rv_vload_f32t16(T_multiply_2__comp0 + 0)));
         rv_vstore_f32t16(T_multiply_3__comp0 + 0, rv_fmul_f32t16(
                  rv_bcast_f32t16(0.7978850007f),
                  rv_vload_f32t16(T_add_2__comp0 + 0)));
         rv_vstore_f32t16(T_tanh_comp0 + 0, rv_ftanh_f32t16(
                  rv_vload_f32t16(T_multiply_3__comp0 + 0)));
         rv_vstore_f32t16(T_add_3__comp0 + 0, rv_fadd_f32t16(
                  rv_bcast_f32t16(1.0f),
                  rv_vload_f32t16(T_tanh_comp0 + 0)));
         rv_vstore_f32t16(T_multiply_4__comp0 + 0, rv_fmul_f32t16(
                  rv_bcast_f32t16(0.5f),
                  rv_vload_f32t16(T_add_3__comp0 + 0)));
         rv_vstore_f32t16(T_multiply_1[j + 16 * _t2] + (16 * k + 0),
                  rv_fmul_f32t16(rv_vload_f32t16(T_add_comp0 + 0),
                  rv_vload_f32t16(T_multiply_4__comp0 + 0)));
      }
   }
   rspmd_thd_barrier();
   /* [ … ] */
```

Features: Core/thread parallelism Explicit data transfers (DMA) SIMD parallelism Other

Taking advantage of parallel processing for power efficiency

Enabling automatic optimization of AI code

Today’s agenda

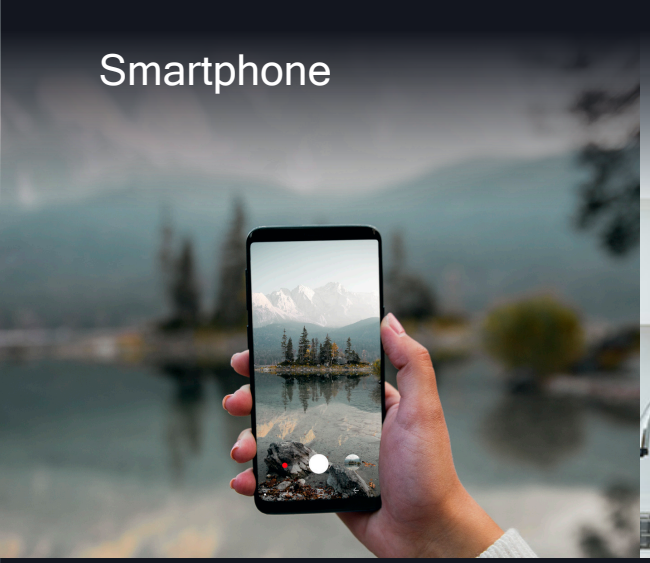
The need for improved AI compilers

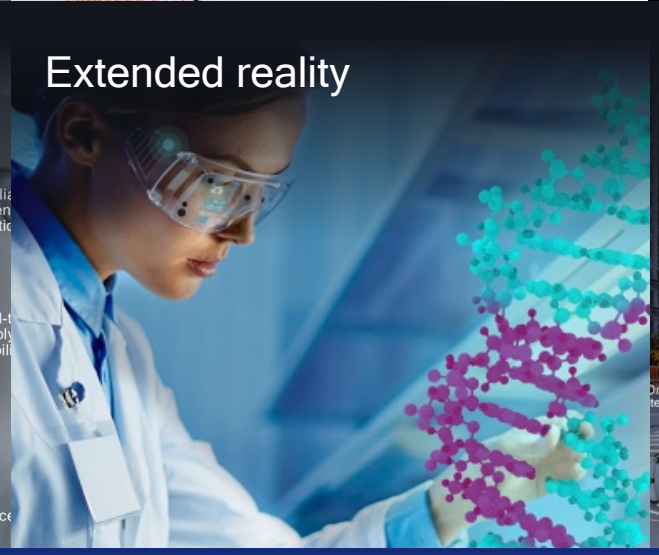
Advancements with polyhedral compilation

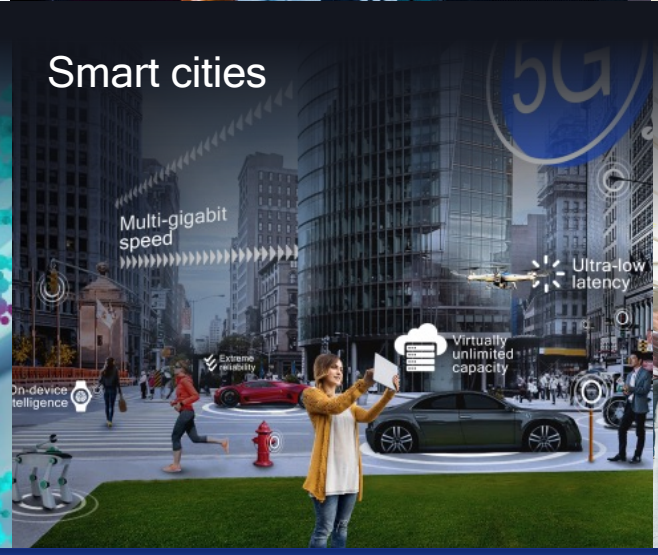
Our leading compiler research

General directions

Q&A





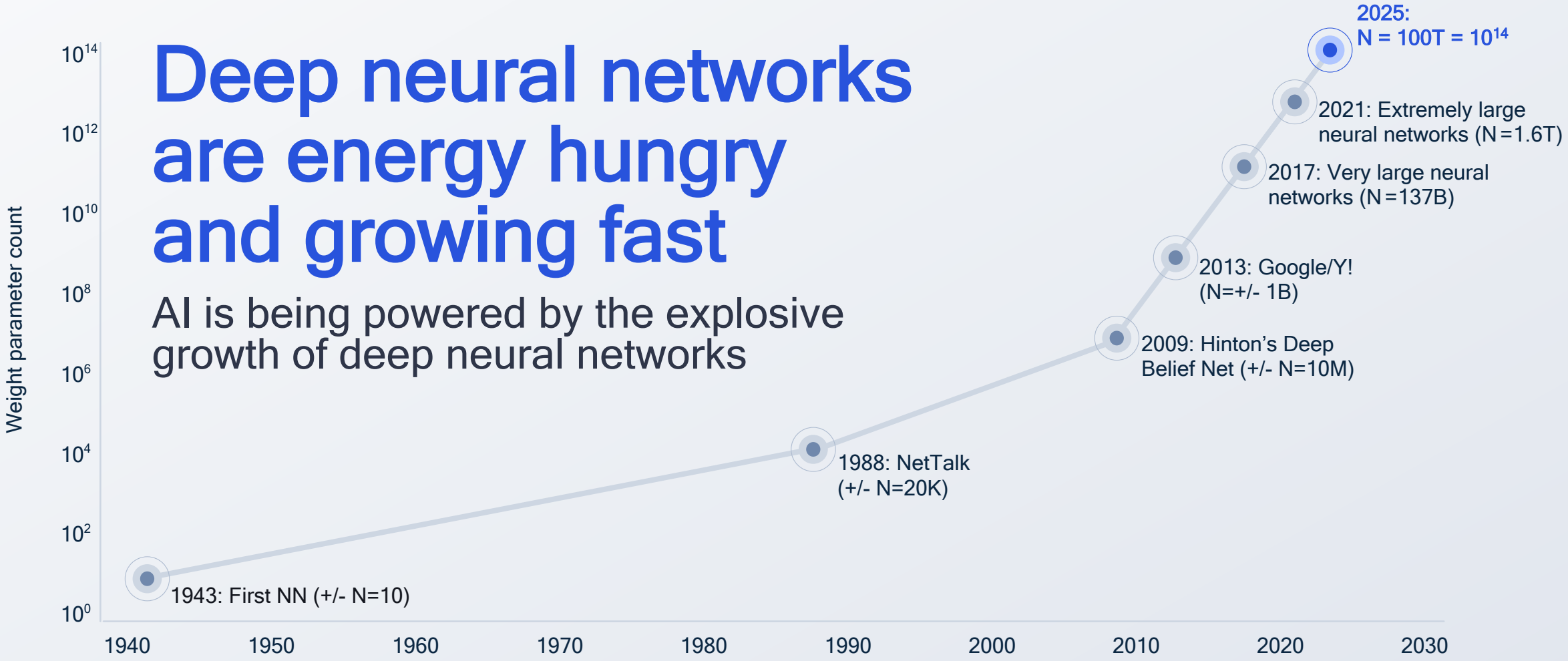


AI is being used all around us

increasing productivity, enhancing collaboration, and transforming industries

Deep neural networks are energy hungry and growing fast

AI is being powered by the explosive growth of deep neural networks



2025

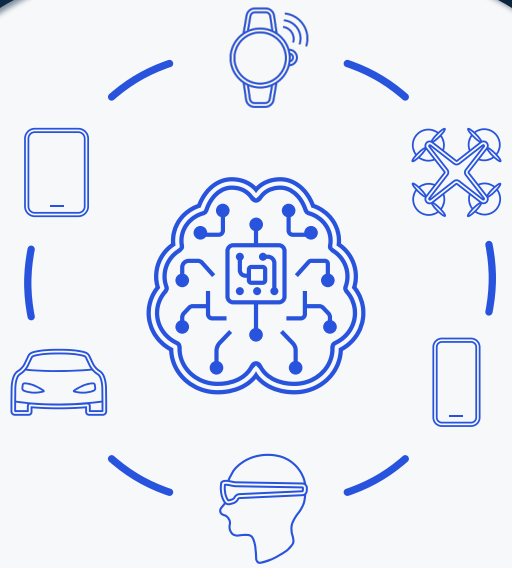
Will we have reached the capacity of the human brain?

Energy efficiency of the human brain is estimated to be 100,000x better than current hardware

Source: Welling

The challenge of AI workloads

- Very compute intensive
- Large, complicated neural network models
- Complex concurrencies
- Real-time
- Always-on



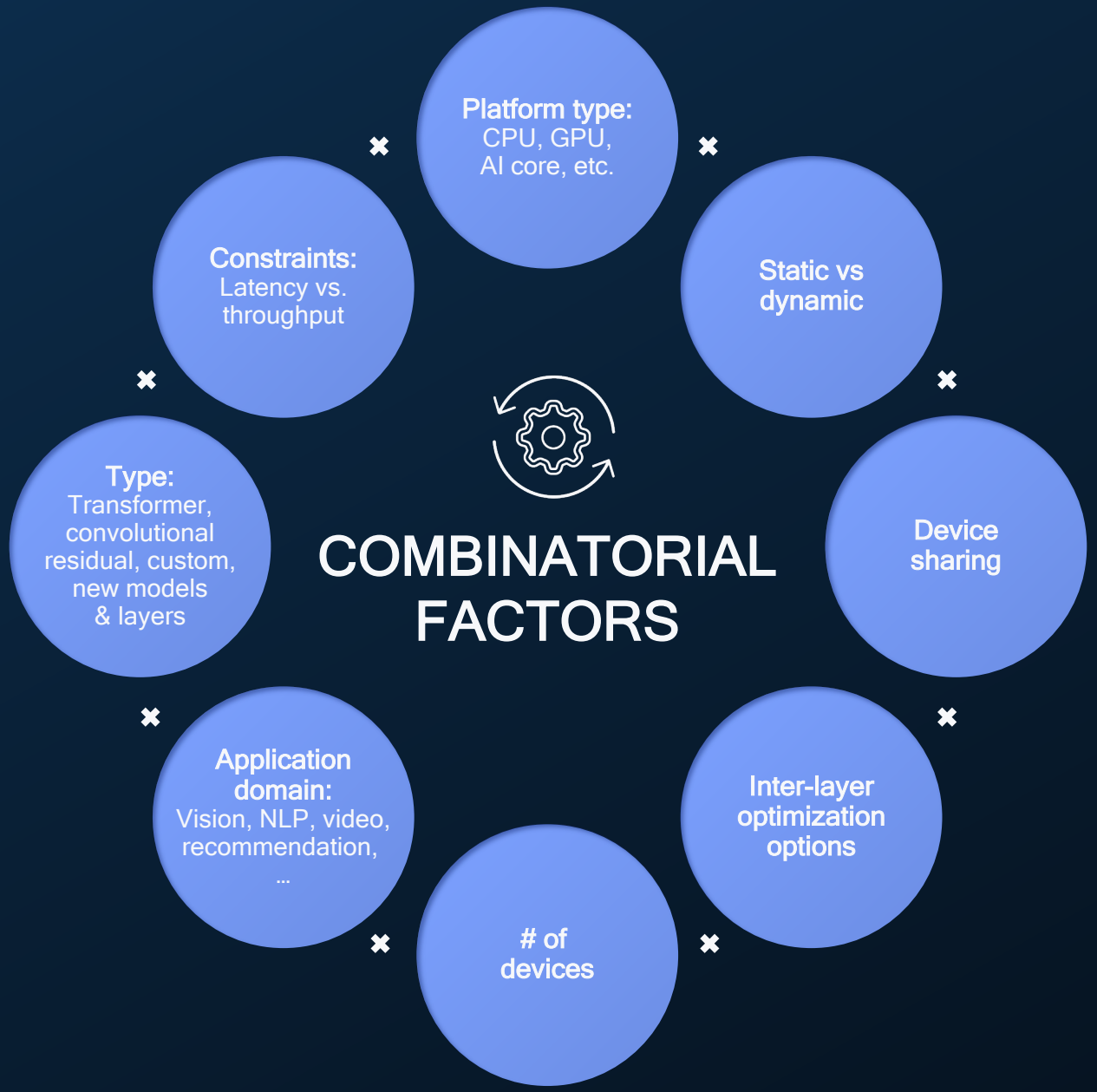
Power and thermal efficiency are essential for on-device AI

Constrained mobile environment

- Must be thermally efficient for sleek, ultra-light designs
- Requires long battery life for all-day use
- Storage/memory bandwidth limitations

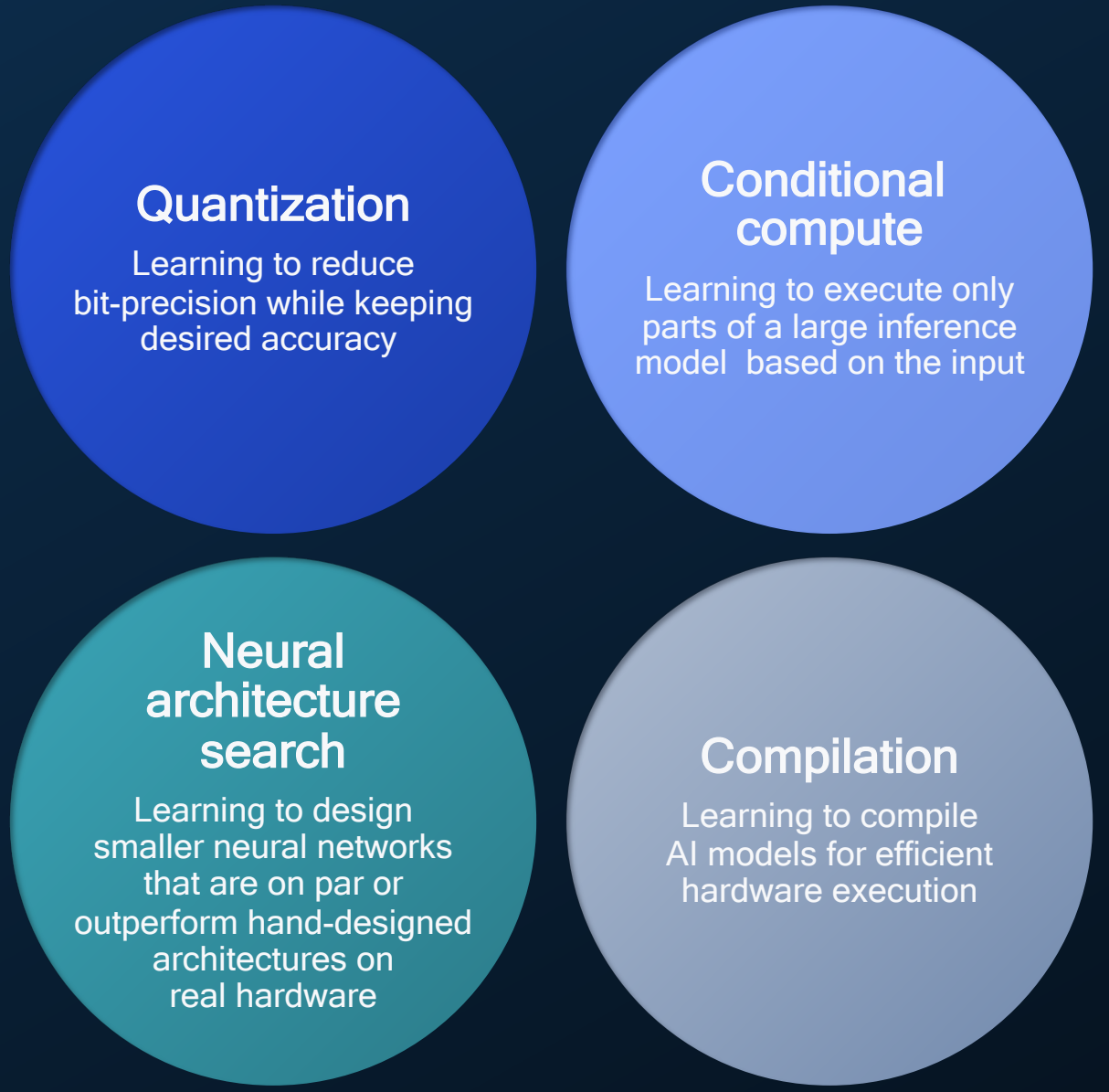
All optimization cannot be done manually

Optimization techniques need to scale with the explosion of AI



Holistic power efficiency research

Multiple axes to shrink AI models and efficiently run them on hardware



What is a compiler?

					Improvement in user coding effort	Main challenges
20th century	C,C++, etc.	Single core processor (e.g. 80286)	Compiler	Executable code	**Before:** Machine code (assembly) **After:** High-level language (C, etc.)	• Match C semantics with core instructions efficiently • Keep ALUs busy • Use registers sparingly
Now	AI model	Highly parallel, energy-constrained processor (e.g. Snapdragon)	AI Compiler	Executable code	**Before:** Complex parallel C++ with intrinsics **After:** Neural net in PyTorch, TensorFlow, etc.	**The same, plus:** • Keep cores, threads, vector units busy • Utilize local memories • Minimize data movement

Compiler priorities have changed with evolving apps, software, and computer architectures

The need for parallelism and data locality

Use all the processing elements, avoid data transfers between remote and local memories

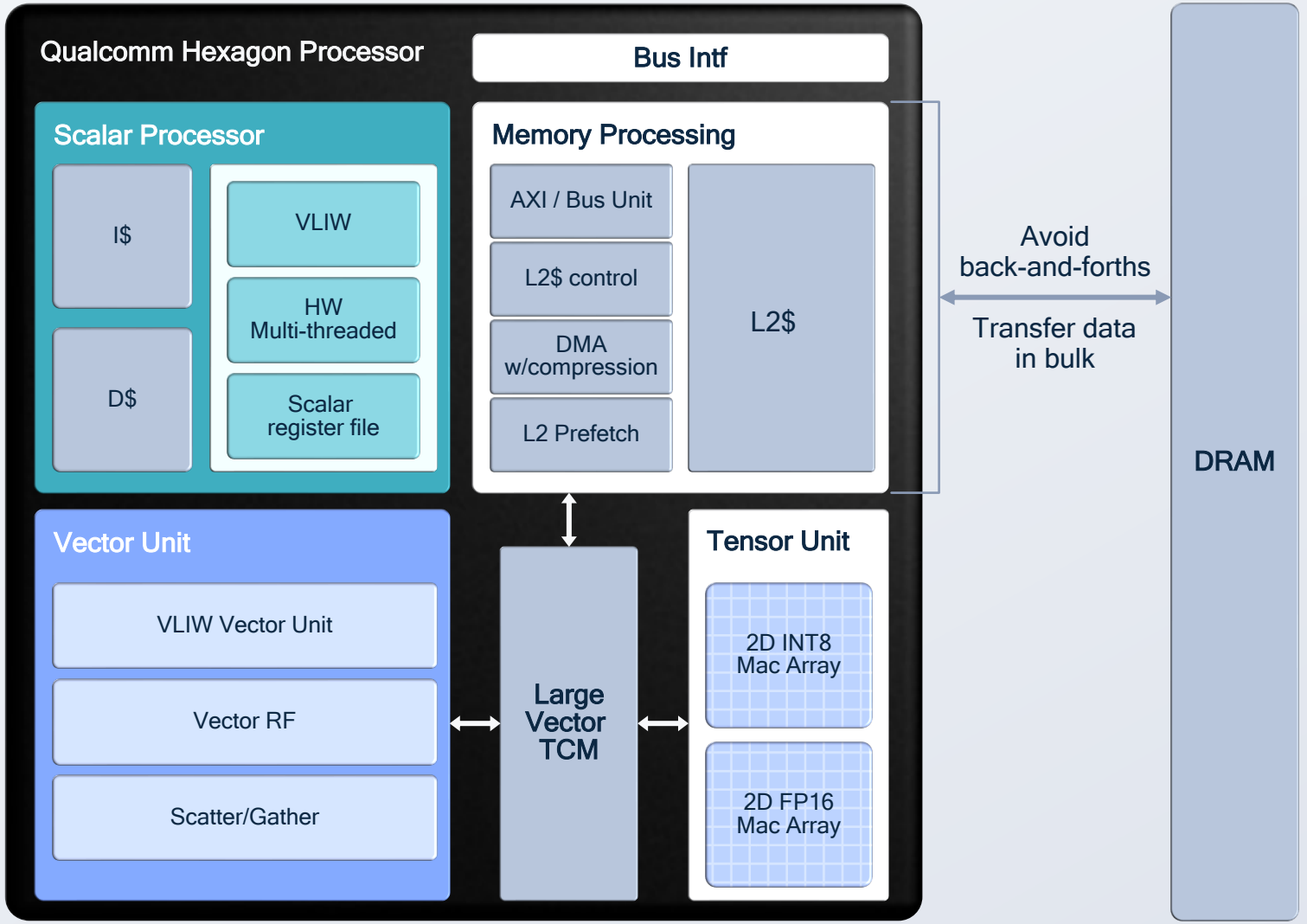
More PEs:
More compute, better power

Parallelism:
Exploit as many PEs as possible

Local memories:
Faster access to data

Data locality:
Use data shortly after it comes to local mem

Transfer useful data in bulk



Parallelism and data locality in code

Illustrated on a BERT language model subgraph optimized by the Qualcomm Polyhedral Mapper

```
void subgraph_3_1(float (*__restrict__ var)[1024],
   float (*__restrict__ var_2_)[1024],
   float*__restrict__ var_3_,
   float (*__restrict__ T_multiply)[4096])
{
   /* [var decls] */
   _t1_1 = rspmd_core_id(0);
   _t2 = rspmd_thd_id(0);
   var_1 = (float (*)[1024])static_alloc(spad, 0, 52ul, 131072ul) /* var_1 */;
   /* [more static allocs] */
   if (_t1_1 <= 11) {
      if (_t2 <= 0) {
         for (int i = 0; i <= 1; i++) {
            int j_1;
            int _t3;
            int _t4;
            rspmd_dma_get((void const volatile*)(var[32 * _t1_1] + 0), (void
               volatile*)(var_1[32 * _t1_1 + -32 * _t1_1] + 0), (unsigned long)
               (unsigned int)4096, (long)4096, (long)4096, (unsigned long)(
               unsigned int)32, 0);
            rspmd_dma_get((void const volatile*)(var_2_[2048 * i] + 0), (void
               volatile*)(var_2__1[-2048 * i + 2048 * i] + 0), (unsigned long)(
               unsigned int)4096, (long)4096, (long)4096, (unsigned long)(
               unsigned int)2048, 0);
            rspmd_dma_get((void const volatile*)(var_3_ + 2048 * i), (void
               volatile*)(var_3__1 + (-2048 * i + 2048 * i)), (unsigned long)(
               unsigned int)8192, (long)0, (long)0, (unsigned long)(
               unsigned int)1, 0);
            rspmd_dma_wait(0);
            rspmd_thd_barrier();
```

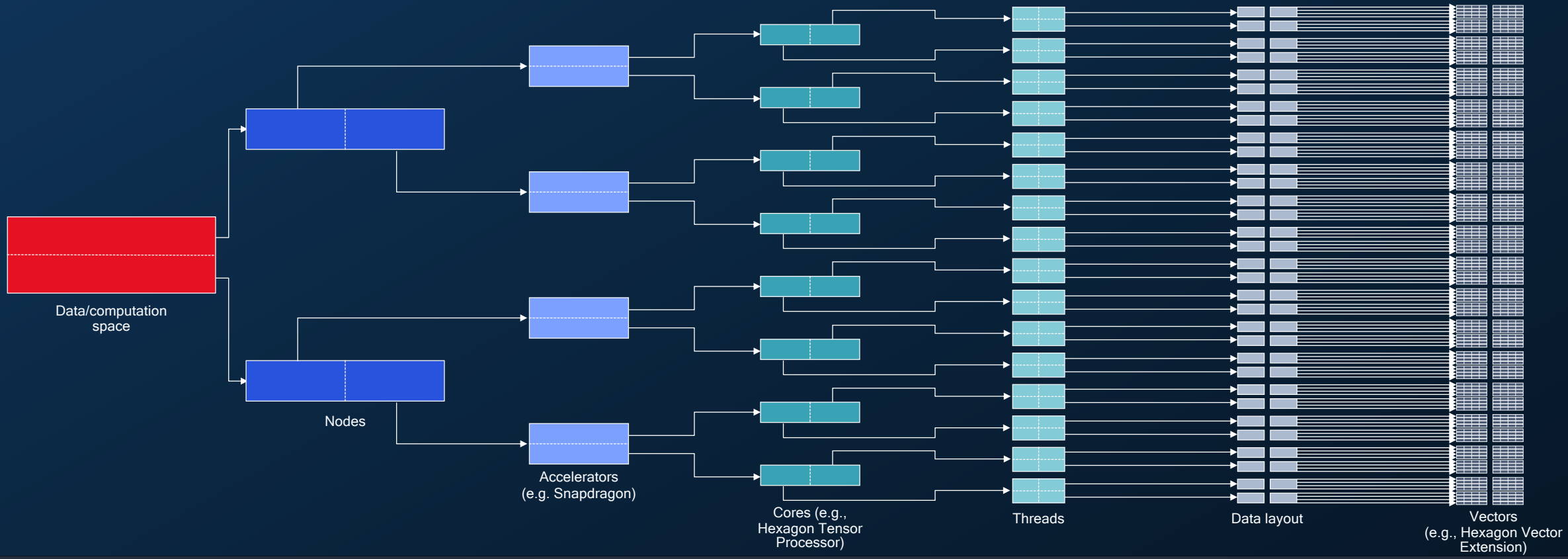
Features:
Core/thread parallelism Explicit data transfers (DMA) SIMD parallelism Other

```
for (int j = 0; j <= 15; j++) {
    for (int k = 0; k <= 127; k++) {
        rv_vstore_f32t16(T_dense_comp1[_t2] + 0, rv_bcast_f32t16(0.0f));
        for (int i1 = 0; i1 <= 1023; i1++) {
            rv_vstore_f32t16(T_dense_comp1[_t2] + 0, rv_fadd_f32t16(
                rv_vload_f32t16(T_dense_comp1[_t2] + 0), rv_fmul_f32t16
                (rv_vload_s_f32t16(var_1[j + 16 * _t2] + i1, 0l),
                 rv_vload_s_f32t16(var_2__1[16 * k + 0] + i1, 1024l))));
        }
        rv_vstore_f32t16(T_add_comp0 + 0, rv_fadd_f32t16(
                    rv_vload_f32t16(T_dense_comp1[_t2] + 0),
                    rv_vload_f32t16(var_3__1 + (16 * k + 0))));
        rv_vstore_f32t16(T_power_comp0 + 0, rv_fpow_f32t16(
                    rv_vload_f32t16(T_add_comp0 + 0),
                    rv_bcast_f32t16(3.0f)));
        rv_vstore_f32t16(T_multiply_2__comp0 + 0, rv_fmul_f32t16(
                    rv_bcast_f32t16(0.04471499845f),
                    rv_vload_f32t16(T_power_comp0 + 0)));
        rv_vstore_f32t16(T_add_2__comp0 + 0, rv_fadd_f32t16(
                    rv_vload_f32t16(T_add_comp0 + 0),
                    rv_vload_f32t16(T_multiply_2__comp0 + 0)));
        rv_vstore_f32t16(T_multiply_3__comp0 + 0, rv_fmul_f32t16(
                    rv_bcast_f32t16(0.7978850007f),
                    rv_vload_f32t16(T_add_2__comp0 + 0)));
        rv_vstore_f32t16(T_tanh_comp0 + 0, rv_ftanh_f32t16(
                    rv_vload_f32t16(T_multiply_3__comp0 + 0)));
        rv_vstore_f32t16(T_add_3__comp0 + 0, rv_fadd_f32t16(
                    rv_bcast_f32t16(1.0f),
                    rv_vload_f32t16(T_tanh_comp0 + 0)));
        rv_vstore_f32t16(T_multiply_4__comp0 + 0, rv_fmul_f32t16(
                    rv_bcast_f32t16(0.5f),
                    rv_vload_f32t16(T_add_3__comp0 + 0)));
        rv_vstore_f32t16(T_multiply_1[j + 16 * _t2] + (16 * k + 0),
                    rv_fmul_f32t16(rv_vload_f32t16(T_add_comp0 + 0),
                    rv_vload_f32t16(T_multiply_4__comp0 + 0)));
} }
rspmd_thd_barrier();
/* [ … ] including _t2 > 0*/
```

The Qualcomm Polyhedral Mapper

Qualcomm Technologies’ polyhedral approach to AI model optimization

How we optimize AI models: the Qualcomm Polyhedral Mapper



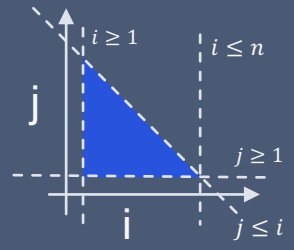
AI code is repeated computations

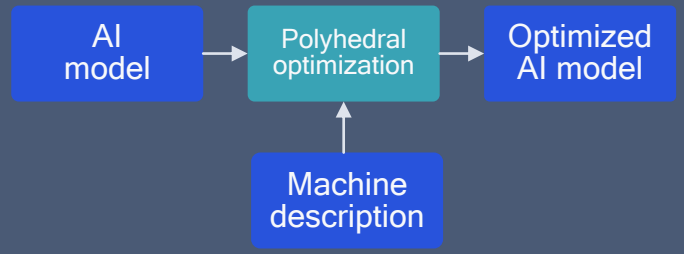
Loops

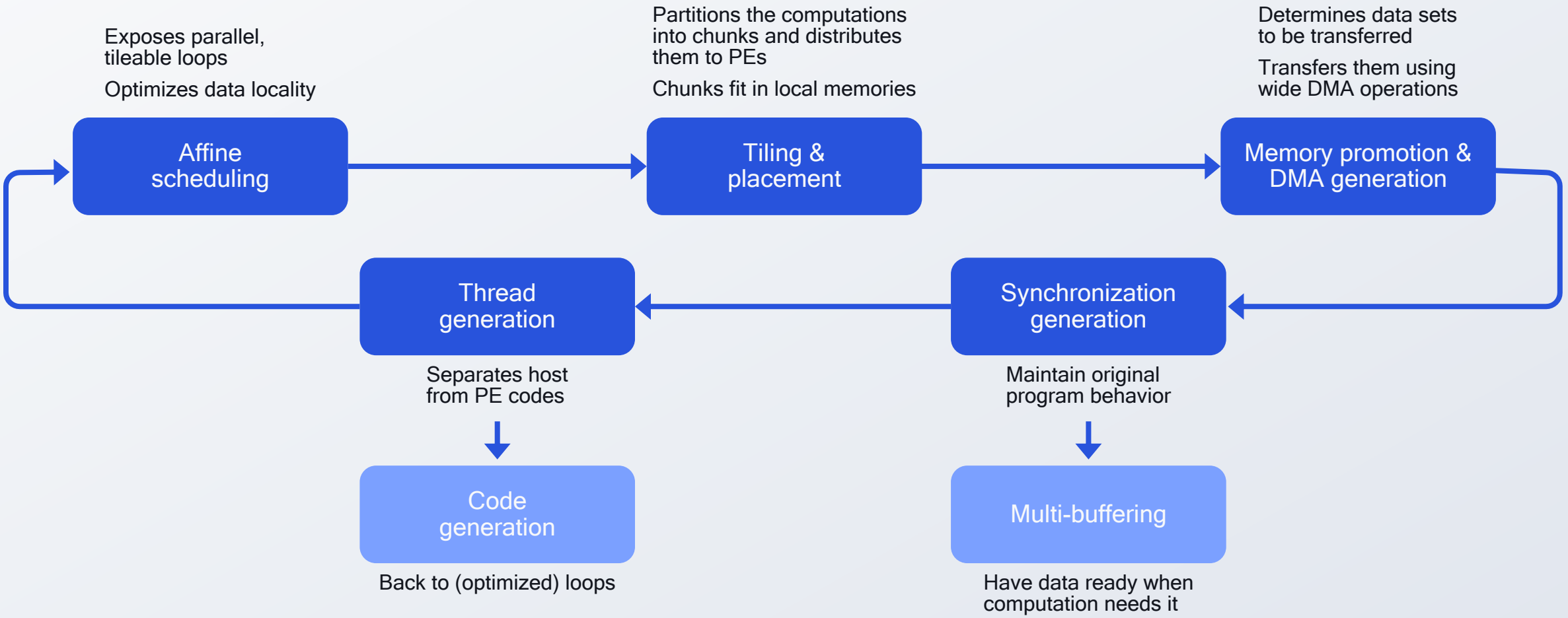
Computation

```
for i in [1, n]
  for j in [1, i]
    Read activation and weights elements
    Use them to compute output elements
```

Model the loops as a polyhedron



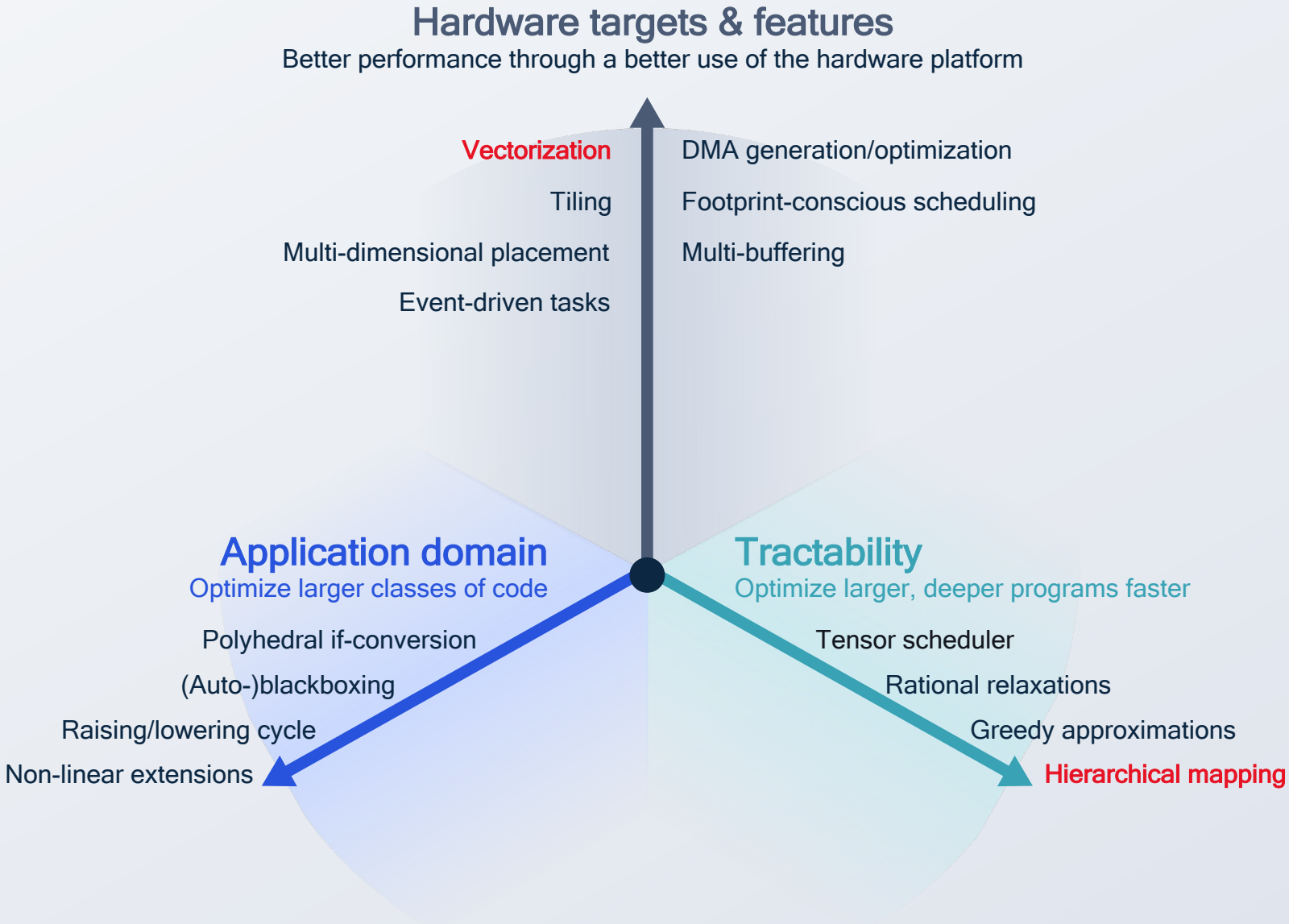




Bird’s eye view of polyhedral AI compiler optimizations

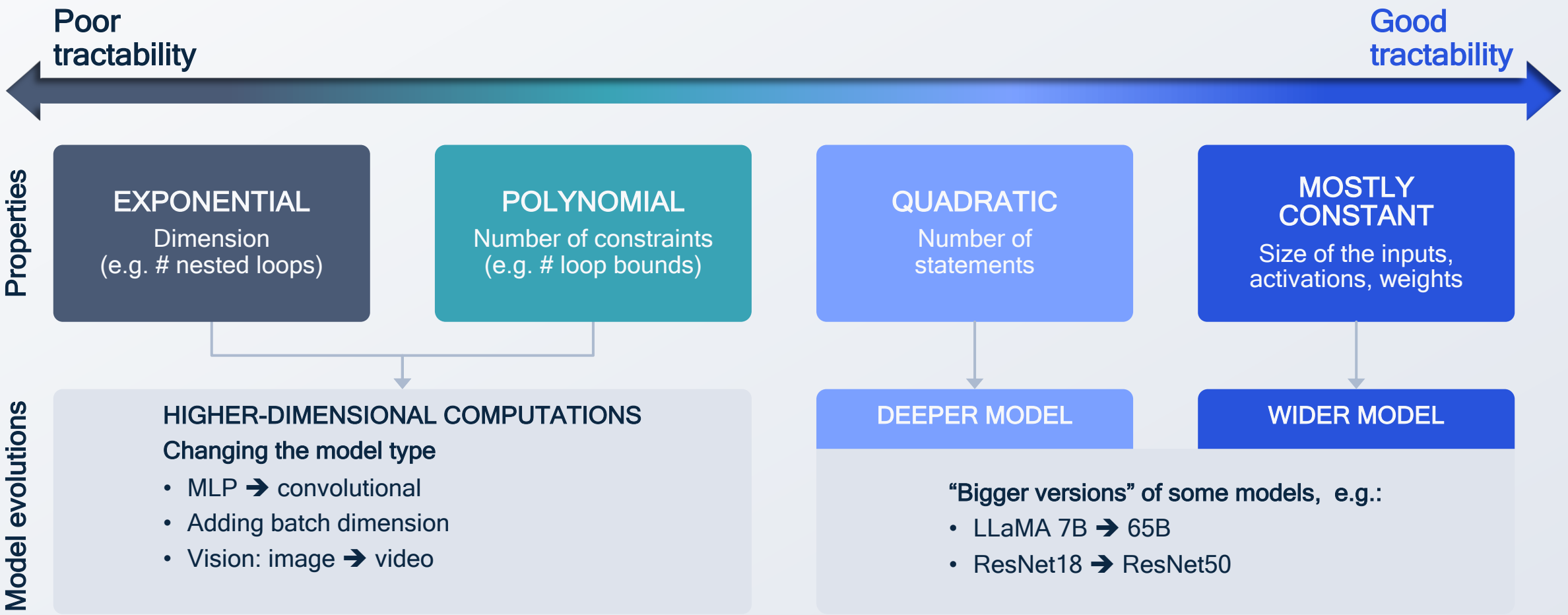
Challenge:

AI compilers need to grow in three different directions



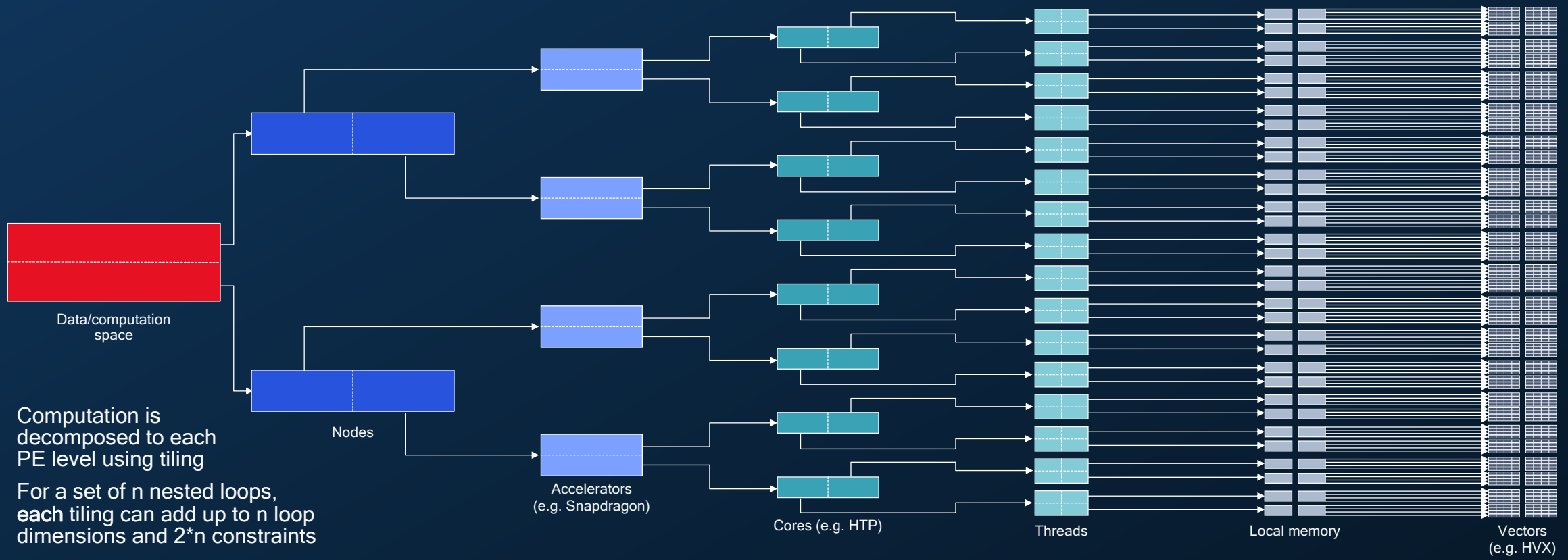
Hierarchical mapping optimization

Improving tractability of polyhedral optimization for deep PE hierarchies



Some evolutions of AI models are tractable in polyhedral AI compiler

We address the difficult ones and leverage the easy ones



Computation is decomposed to each PE level using tiling

For a set of n nested loops, **each** tiling can add up to n loop dimensions and 2*n constraints

for i0 in [1, nti0]
  for j0 in [1, ntj0]

for i1 in [1, nti1]
  for j1 in [1, ntj1]

for i2 in [1, nti2]
  for j2 in [1, ntj2]

for i3 = [1, nti3]
  for j3 = [1, ntj3]

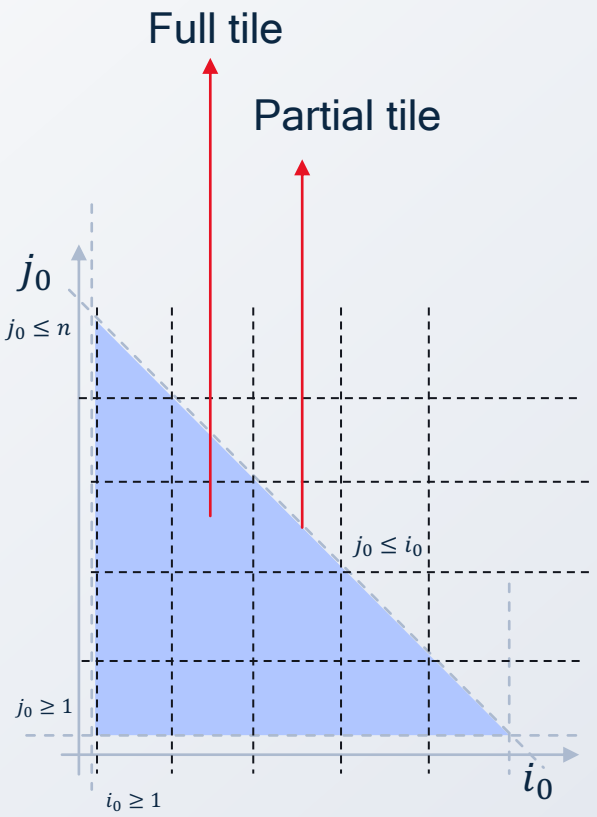
for i4 = [1, nti4]
  for j4 = [1, ntj4]

for i5 = [1, nti5]
  for j5 = [1, ntj5]

f(i0, j0, …, i5, j5)

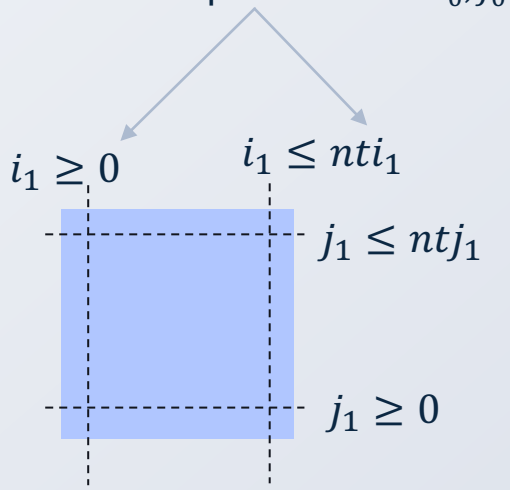
Hierarchical mapping optimization addresses tractability issues to scale polyhedral compilation

Loop transformations



Look at full tiles:
Omit partial-tile boundaries

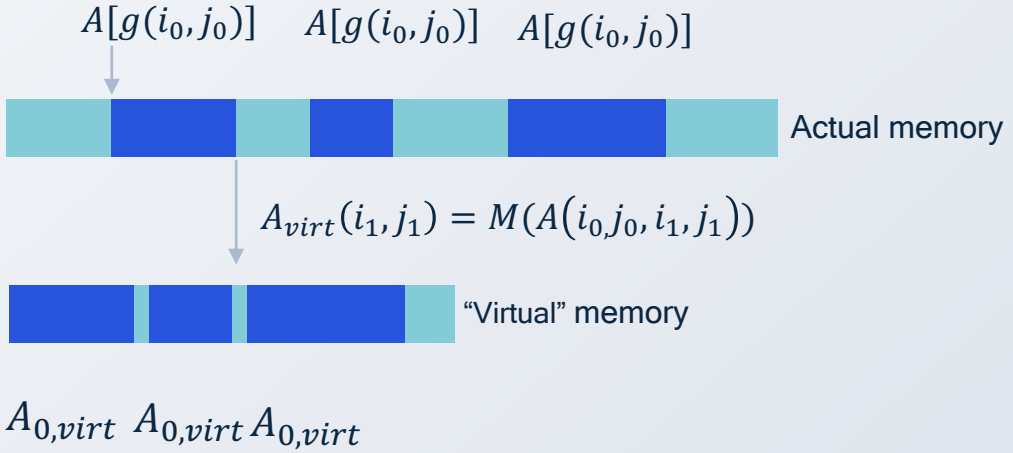
Bounds independent from i_0, j_0



[Pra16] Scalable hierarchical Polyhedral Compilation

Data transformations

Map tile data to a virtual space where accessed data regions are independent from other loop indices

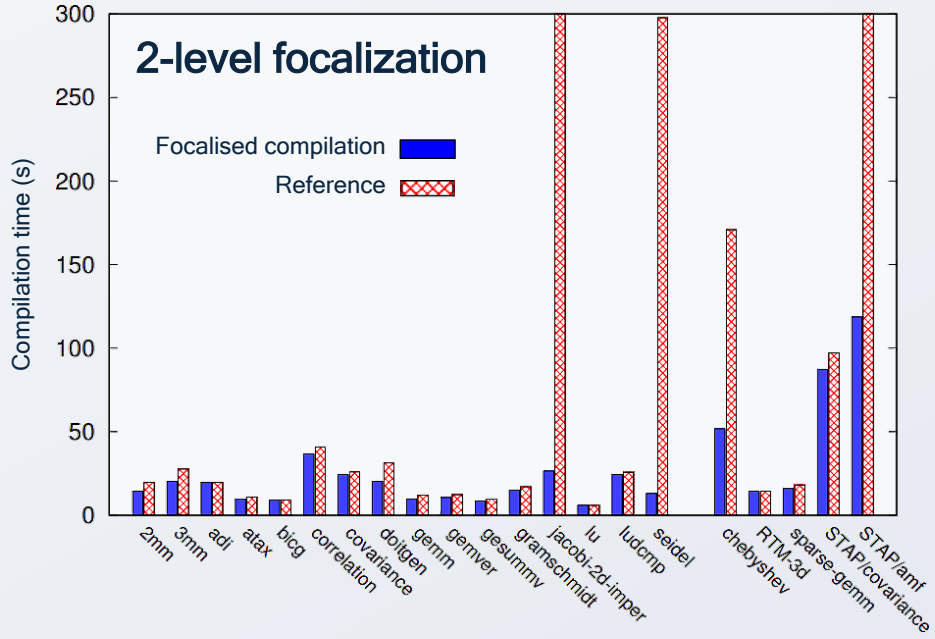


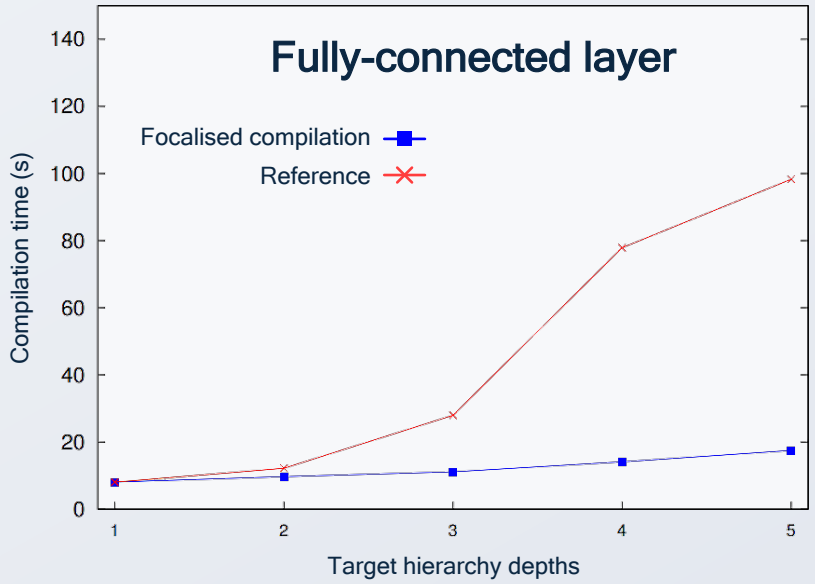
Focalization on a particular tiling level with loop and data transformations

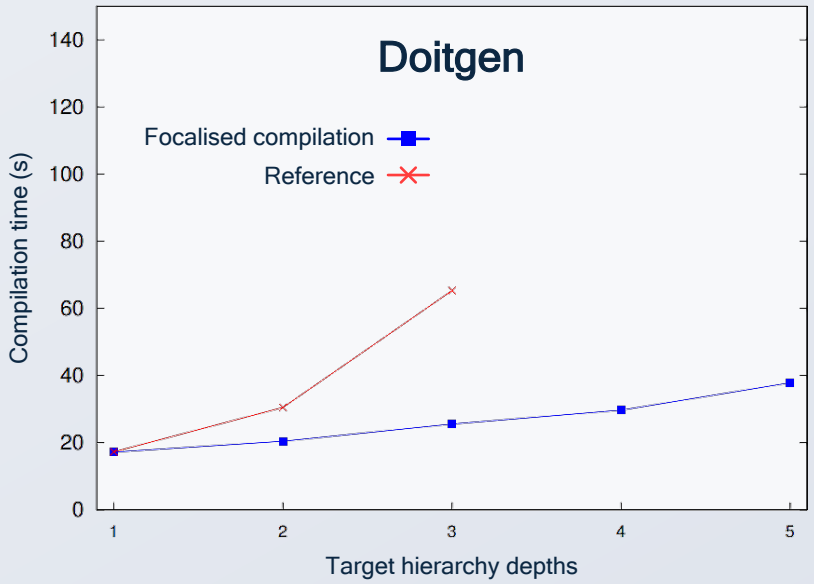
This makes intractable AI model evolutions tractable in AI compiler

Using focalization, AI compile time reduces significantly

Successful on a broad class of programs

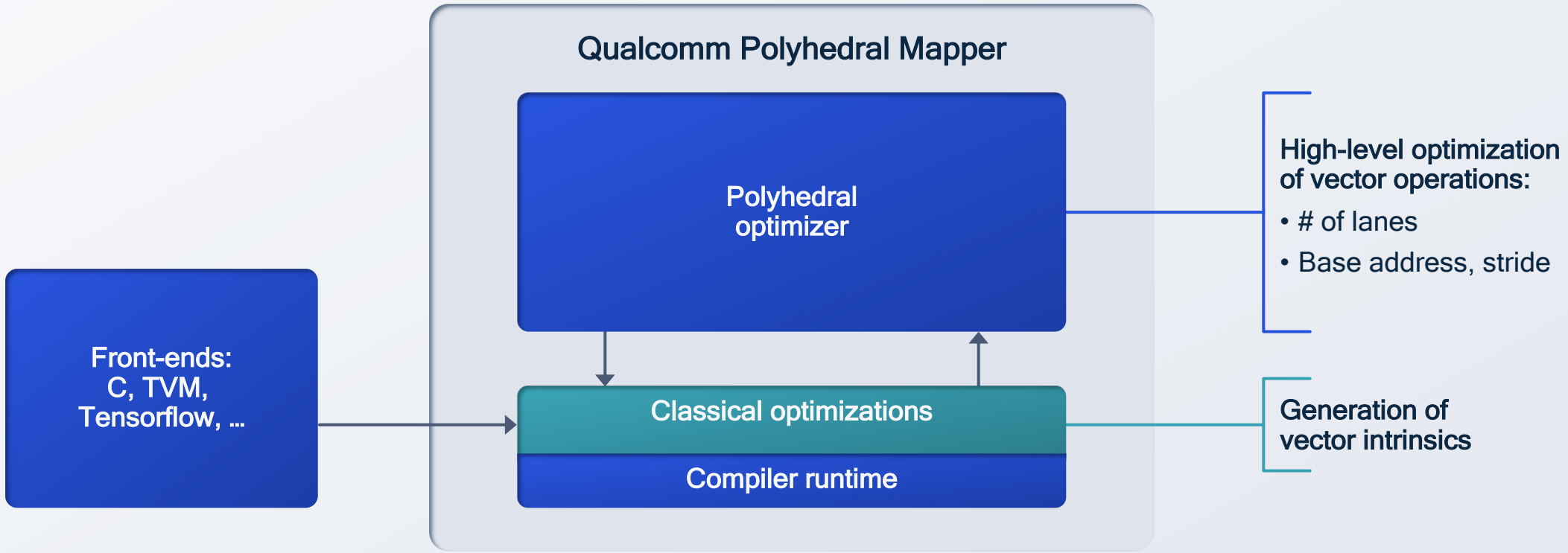






Auto-vectorization

In the Qualcomm Polyhedral Mapper



Separation of concerns:

Polyhedral Mapper finds independent operations suitable for vectorization

Underlying compiler generates optimized calls to vector intrinsics

Cooperative optimization between polyhedral and classical optimizations of the Qualcomm Polyhedral Mapper

By automating vectorization, AI compiler allows programmers to write fewer lines of code and get substantial performance

Auto-optimized Transposed LayerNorm with the Qualcomm Polyhedral Mapper

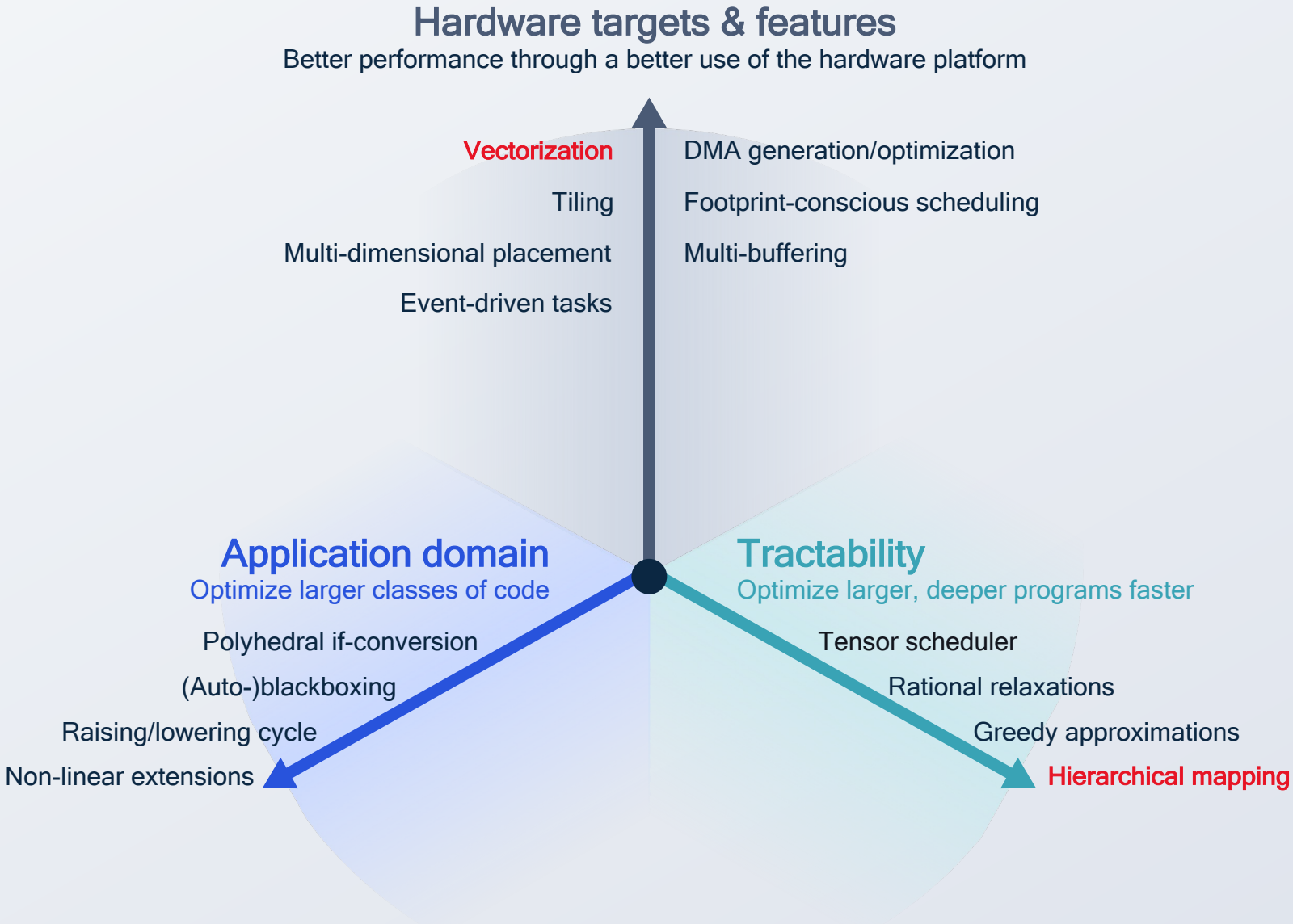
32.2% of hand-tuned performance

620x speedup over sequential performance

55.4 #optimized LOCs / #original LOCs ratio

LOC: lines of code

We are solving these challenges with polyhedral AI compilers



Broad range of research directions for AI compiler optimization

User empowerment & productivity

Tools to optimize time to market of user applications

AI-augmentation of compilers

Train models to make program optimization decisions

Scale AI up to larger graphs

Efficiently exploit global graph structure

Heterogeneous computing

Fully utilize the Snapdragon computing capabilities

High-level optimizations for generative models

Automate algorithmic optimizations, e.g., in large language models

Global optimality

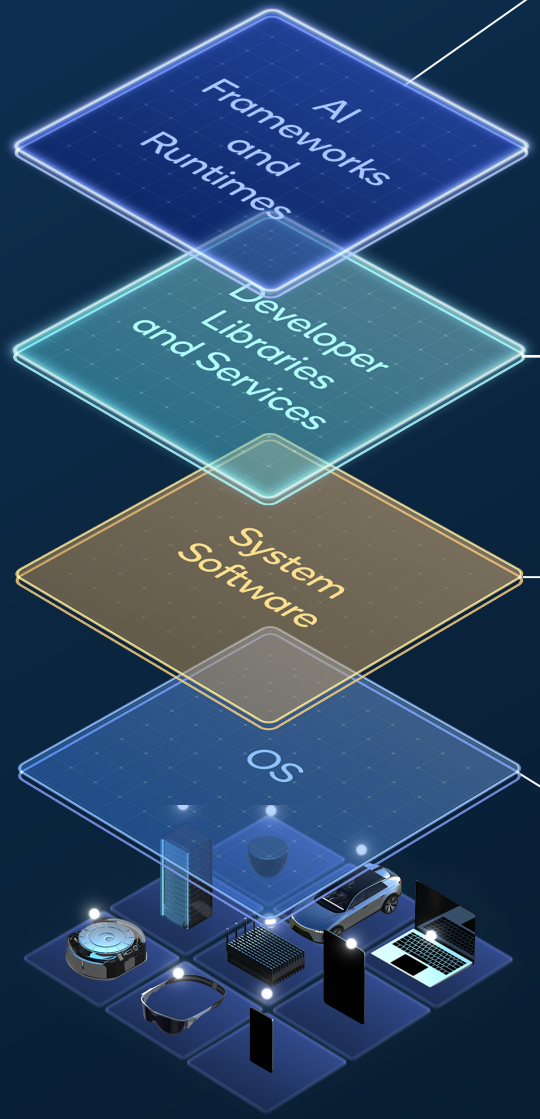
Going from local optima to the global optimum: across PE levels, across whole model

Dynamical systems and AI

Combine efficiency of dynamic models with AI learning

Qualcomm AI Stack

Qualcomm AI Studio



AI Frameworks

TensorFlow PyTorch ONNX Keras

AI Runtimes

Qualcomm® Neural Processing SDK ONNX RUNTIME TF Lite Micro Direct ML TF Lite

Qualcomm® AI Engine Direct

Math Libraries Compilers Virtual platforms

Profilers & Debuggers Programming Languages Core Libraries

System Interface SoC, accelerator drivers Emulation Support

android Zephyr ubuntu® CentOS QNX

Improved AI compilers are key to making machine learning ubiquitous

Qualcomm AI Research has achieved state-of-the-art results in AI compiler

We are enabling AI models to scale across a variety of use cases and run efficiently

Questions

Connect with us

www.qualcomm.com/research/artificial-intelligence

www.qualcomm.com/news/onq

@QCOMResearch

www.youtube.com/c/QualcommResearch

www.slideshare.net/qualcommwirelessevolution

Thank you

Qualcomm

Follow us on:

For more information, visit us at:

qualcomm.com & qualcomm.com/blog