GAMMA: Mapping Space Exploration via Optimization

Sheng-Chun (Felix) Kao

Georgia Institute of Technology

S.-C. Kao, T Krishna, “GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm”, ICCAD, 2020

Code available: https://github.com/maestro-project/gamma
For this work, we assume the HW Resources are fixed.
Impact of Mappings

Layer: 2nd layer of VGG16 (N=1, K=64, C=64, Y=224, X=224, R=3, S=3)
HW resources: PEs: 168, SL: 512B, SG:108KB

Different mapping leads to drastically different HW performance
• Up to 4 order of magnitude difference by different mappings

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (cycles)</td>
<td>8.18E+06</td>
<td>1.85E+09</td>
</tr>
<tr>
<td>Energy (nJ)</td>
<td>1.12E+10</td>
<td>8.34E+10</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>1.52E+05</td>
<td>2.24E+09</td>
</tr>
<tr>
<td>Area (um²)</td>
<td>1.77E+09</td>
<td>4.57E+13</td>
</tr>
</tbody>
</table>
Mapping Search as an Optimization Problem

**Optimization of accelerators at design/compile time**

Given Platform/ accel. infrastructure
- Constrained HW resources (PEs/ buffers)

Find optimal mapping strategy
- Tune each knob of the mapping aspects

Component of mapping
- Num. of Parallelism
- Compute order
- Parallelism dim.
- Tiling strategy
- Component of mapping
Constructing the Search Space

GAMMA's genetic encoding

<table>
<thead>
<tr>
<th>P</th>
<th>C</th>
<th>R</th>
<th>S</th>
<th>X</th>
<th>K</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>3</td>
<td>3</td>
<td>15</td>
<td>64</td>
<td>10</td>
</tr>
</tbody>
</table>

Mapping

| Compute order | ✓ |
| Tiling strategy/sizes | ✓ |
| # of Parallelism | ✓ |
| Parallelism dimension | ✓ |

```
for c=[0, C):
    for r=[0, R):
        for s=[0, S):
            for x=[0, X):
                for k=[0, K):
                    for y=[0, Y):
                        output[y,x,k,c] +=
                           weight[r,s,k,c] *
                           Input[y+r,x+s, k,c]
```

……

```
for k=[0, K, kt):
    for y=[0, Y, yt):
        for ct=[0, C, 20):
            for rt=[0, R, 3):
                for st=[0, S, 3):
                    for xt=[0, X, 15):
                        for kt=[0, K, 64):
                            for yt=[0, Y, 10):
                                output[yt,xt,kt,ct] +=
                                   weight[rt,st,kt,ct] *
                                   Input[yt+rt,xt+st, kt,ct]
```

……

```
for k=[0, K, kt):
    for y=[0, Y, yt):
        for ct=[0, C, 20):
            for rt=[0, R, 3):
                for st=[0, S, 3):
                    for xt=[0, X, 15):
                        par_for kt=[0, K, 64):
                            for yt=[0, Y, 10):
                                output[yt,xt,kt,ct] +=
                                   weight[rt,st,kt,ct] *
                                   Input[yt+rt,xt+st, kt,ct]
```
Example: NVDLA-like Mapping

<table>
<thead>
<tr>
<th>P</th>
<th>K</th>
<th>C</th>
<th>R</th>
<th>S</th>
<th>Y</th>
<th>X</th>
<th>P</th>
<th>K</th>
<th>C</th>
<th>R</th>
<th>S</th>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>1</td>
<td>64</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>C,64</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**NVDLA-like**
- 2-level of parallelism
- Parallelize across K, C dimension

**Level of Par.**
- **L2: Par. dim.**
  - SpatialMap(1,1) K;
  - TemporalMap(64,64) C;
  - TemporalMap(3,3) R;
  - TemporalMap(3,3) S;
  - TemporalMap(3,3) Y';
  - TemporalMap(3,3) X';
  - Cluster(64);
- **L1: Par. dim.**
  - SpatialMap(1,1) C;
  - TemporalMap(1,1) K;
  - TemporalMap(3,3) Y';
  - TemporalMap(3,3) X';
  - TemporalMap(3,3) R;
  - TemporalMap(3,3) S;

**Tile sizes**
- $P_{L1} = 64$

**Compute order**
- Decoded. L2-Mapping
- Decoded. L1-Mapping
GAMMA: GA-based Optimization

Evolution
- 3 additional genetic operators

Evaluation
- Evaluate fitness (targeting objective: e.g., Latency)
- HW cost model (MAESTRO)

Conventional Optimization Methods

Supported optimization methods

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>Random Search. We randomly sample design points and keeps the best solution.</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm. We encode the design point into a series of genes. We evolve the genes by mutation and crossover.</td>
</tr>
<tr>
<td>DE</td>
<td>Differential Evolution. In DE, we mutate by the difference vector. When mutation, we sample two parents and extract their difference on each dimension to formulate a difference vector. Next, we sample another parent and mutate it by adding the difference vector to it.</td>
</tr>
<tr>
<td>(1+1)ES</td>
<td>(1+1) Evolution Strategy. For each parent we mutate it to reproduce $\lambda$ number of mutated children. The parent and children compete with each other by their fitness values and the best one will go to the next generation.</td>
</tr>
<tr>
<td>CMAES</td>
<td>Covariance Matrix Adaptation-ES. We use covariance matrix of the elite group and the entire population to estimate the distance to the optimum. We adapt the step size based on covariances.</td>
</tr>
<tr>
<td>TBPSA</td>
<td>Test-based Population-Size Adaptation. We estimate the trend of the population’s fitness values. When the fitness values converge, the algorithm will adapt to have a smaller population size.</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization. We track global best and parent best, which represent the global and local information respectively. We update the parameters by the weighted sum of them.</td>
</tr>
<tr>
<td>pPortfolio</td>
<td>Passive Portfolio. We maximize diversity by spreading the risk broadly to multiple instances (optimizers). We launch K numbers of optimization methods, each experiencing 1/K samples, and take the best performing solution provided by one of them.</td>
</tr>
</tbody>
</table>

https://facebookresearch.github.io/nevergrad/
GAMMA Setup

• Setup
  • git clone https://github.com/maestro-project/gamma.git
  • conda create --name gammaEnv python=3.6
  • conda activate gammaEnv
  • pip install -r requirements.txt
  • python build.py

• Run GAMMA
  • ./run_gamma_with_hwconfig.sh

• Run other optimizations
  • ./run_others_with_hwconfig.sh
GAMMA Code-base

- Soft-link to maestro
- User-defined HW configuration
- Supported model descriptions

Result directory
Main program

Code available: https://github.com/maestro-project/gamma
User Options

```
python main.py --fitness1 latency --fitness2 power --stages 1 --num_pe 168 --l1_size 512 --l2_size 108000 \--NocBW
81920000 --slevel_min 1 --slevel_max 2 --epochs 10 \
--model vgg16 --singlelayer 1
```

```
python main.py --fitness1 latency --fitness2 power --stages 1 --slevel_min 1 --slevel_max 2 --epochs 10 \
--model vgg16 --singlelayer 1 --hwconfig hw_config.m
```

**Objective**

- **fitness1/fitness2**: The first/second fitness objective (latency/ power/ energy)
- **stages**: Number of stages, can choose from [1 or 2]
  - Choose 1, the system will only optimize on fitness1
  - Choose 2, the system will optimize on fitness1 and then fitness2
- **model**: The model to run
- **singlelayer**: The layer index of the selected model, if want to optimize only single layer. If want to optimize all layers, skip this specification.

**Constraint**

- **num_pe**: Number of PEs
- **l1_size/l2_size**: L1/L2 size (Bytes)
- **slevel_min**: The minimum number of parallelism
- **slevel_max**: The maximum number of parallelism, The parallelism goes from [slevel_min, slevel_max]
- **hwconfig**: Read in HW configuration from file

**Hyper parameters**

- **method**: The optimization methods to choose from (PSO/ Portfolio/ OnePlusOne/ CMA/ DE/ TBPSA/ pureGA/ Random/ GA)
- **epochs**: Number of generation for the optimization
- **outdir**: The output result directory
- **Probability**: Probability of each genetic operator can be set at codebase
Add your own HW Configuration

Inside ./data/HWconfigs/hw_config.m

NumPEs: **168**
L1Size: **512**
L2Size: **108000**
NoC_BW: **81920000**

User can also set the HW configuration with hw_config, rather than from argument
Add your own DNN Model

Add new model descriptions here

CONV2D

![CONV2D Diagram]
[Advanced User] GAMMA-specific Tuning

In ./src/GAMMA/gamma_env.py

- **Tune number of population/generation**

```python
def run(self, dimension, stage_idx, prev_stage_value=0, num_population=100, num_generations=100, elite_ratio=0.05, parents_ratio=0.15, ratio_decay=1, num_finetune=1, best_sol_list=None):
```

- **Ratio of elite we kept in the next generation. Here we kept 5% of populations.**

- **Ratio of parents we use to do cross over.**

- **In genetic operator, alpha is the mutation ratio.**

```python
def crossover_tile(self, parents, pop, alpha=0.5):
```
**[Advanced User] Add your own Optimization Algorithm**

In `src/Other_opts/main.py`

Swap to your own algorithm

Define the search space.
The range of your parameter.
(Here we define in nevergrad format.
You can use any format e.g., numpy)

Define your optimizer function here.
(Evolution block)

Evaluate your parameters
(Evaluation block):
Modify this to take in your input parameters.

This will handle the HW performance evaluation for you.
Resources

Paper:
S.-C. Kao, T Krishna, “GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm”, ICCAD, 2020

Repo:
https://github.com/maestro-project/gamma