ACPO: AI-Enabled Compiler-Driven Program Optimization

www.huawei.com

A. Ashouri, M. Asif, D. Minh, R. Zhang, Z. Wang, A. Zhang, B. Chan, T. S. Czajkowski, Y. Gao

Toronto Heterogeneous Compiler Lab
Huawei Canada

HUAWEI

Motivation

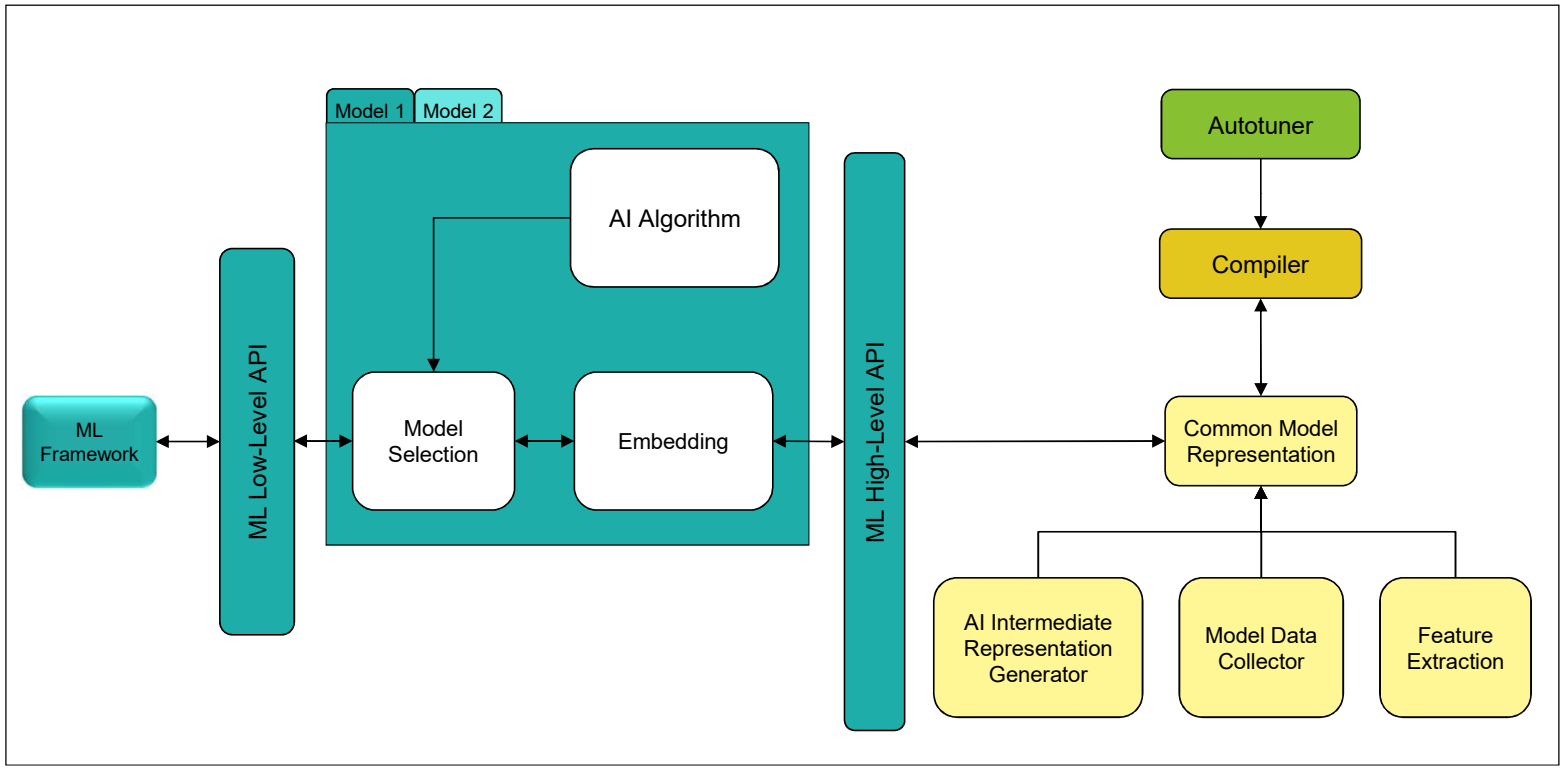
- The key to performance optimization of a program is to decide correctly when a certain transformation should be applied by a compiler.
- Traditionally, such profitability decisions are made by hand-coded algorithms
  - Tuned for a very small number of benchmarks
  - Requiring a great deal of effort to be retuned when the benchmark suite changes
- ML to speed up the tuning process (late 90s and onward)
  - Automatically Constructing optimization heuristics using ML:
    - Optimization selection problem
    - Phase-ordering problem
  - End-to-end frameworks (2018 and onward)
    - Built as a wrapper around compiler (OpenTuner, CompilerGym, NeuroVectorizer)
    - Built-into optimization pass(es) (MLGO/MLGOPerf)
    - Built with Modular design (ACPO)
      - Extensible framework to instantiate ML-Enabled passes
      - Separation of compiler and ML APIs

ACPO Contribution

1. It provides a comprehensive set of *tools*, *libraries*, and *algorithmic* methods
   - Enabling **compiler engineers** with a straightforward and user-friendly interface
   - Instantiate *ACPOModel* classes to replace LLVM's existing hard-coded heuristics.
2. ML APIs and the compiler are seamlessly connected, but at the same time, they are not interdependent
   - changing compiler versions, ML models, or ML frameworks won't break the functionality of ACPO.
   - As long as the inputs/outputs match, users can easily revise the ML side without the need to rebuild LLVM after every change.
3. Showcasing the benefits of ACPO, we demonstrate two different scenarios:
   - Loop Unroll Pass — Building both the interface and the ML model.
   - Function Inlining Pass — Building the interface and leveraging an existing ML model[1].

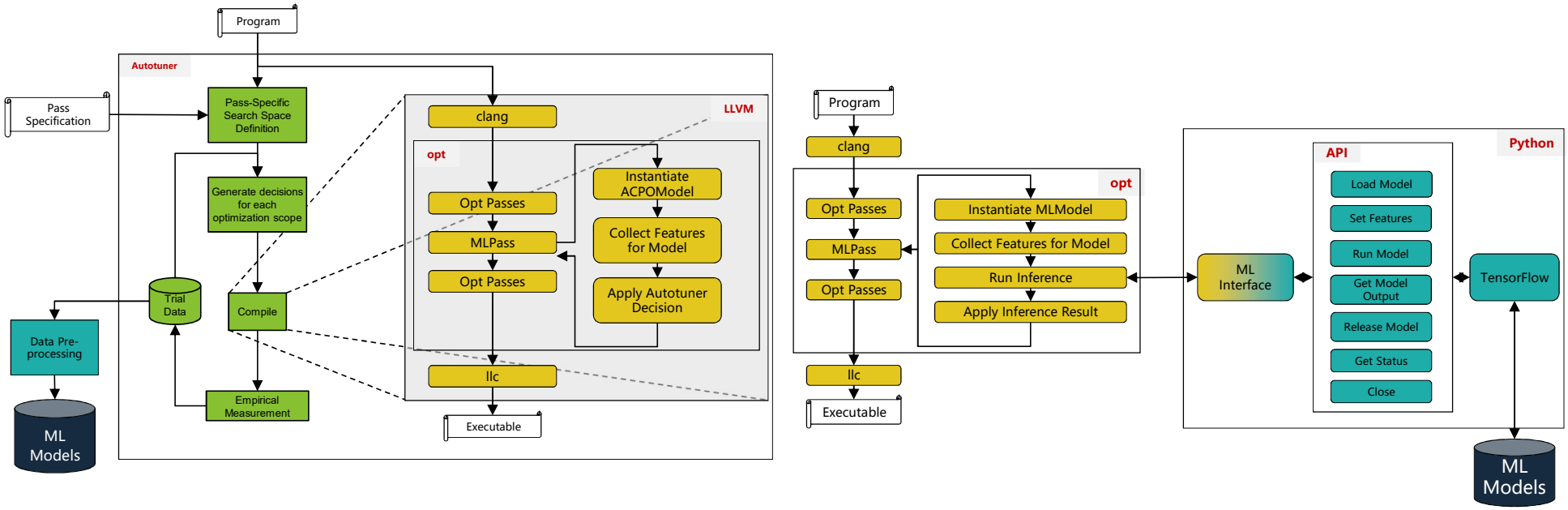
[1] A. Ashouri, M. Elhoushi, Y. Hua, X. Wang, M. Manzoor, B. Chan, and Y. Gao. "MLGOPerf: An ML Guided Inliner to Optimize Performance". arXiv preprint arXiv:2207.08389 (2022).

ACPO Framework



ACPO Infrastructure Architecture

Training & Inference Flows



(a) Training flow

(b) Inference flow

ACPO Model

```
1  class ACPOModel {
2  public:
3    ACPOModel(...);
4    ~ACPOModel();
5    void setMLIF(std::shared_ptr<ACPOMLInterface> ML);
6    std::shared_ptr<ACPOMLInterface> getMLIF();
7    void addRequiredResultField(std::string Name, Type::TypeID &ID);
8    std::unique_ptr<ACPOAdvice> getAdvice();
9    void addFeature(int64_t ID, Constant *Val);
10   void setCustomFeatures(const std::vector<std::pair<std::string, Constant *>> &FeatureValues)
       ;
11
12 protected:
13   virtual void prepareModelInput();
14   virtual bool runModel(std::unique_ptr<ACPOAdvice> &Result);
15 };
```

Listing 1. ACPOModel Header

ACPO ML Interface

```
class ACPOMLInterface {
public:
  ACPOMLInterface();
  virtual ~ACPOMLInterface();
  virtual bool loadModel(std::string ModelSpecFile);
  virtual bool freeModel(std::string ModelName);
  virtual bool registerModel(std::string ModelName, int NumFeatures, int NumOutputs);
  virtual bool registerFeature(std::string ModelName, std::string FeatureName, int Index);
  virtual bool registerOutput(std::string ModelName, std::string OutputName, std::string
     OutputType);
  virtual bool setCustomFeature(std::string ModelName, uint64_t FeatureID, int FeatureValue);
  ...
  virtual bool setCustomFeature(std::string ModelName, std::string FeatureName, bool
     FeatureValue);
  virtual bool setCustomFeatures(std::string ModelName, const std::vector<std::pair<uint64_t,
     std::string>> &FeatureValues);
  virtual bool setCustomFeatures(std::string ModelName, const std::vector<std::pair<std::
     string, std::string>> &FeatureValues);
  ...
  virtual bool runModel(std::string ModelName);
  virtual std::string getOutputType(std::string ModelName, std::string OutputName);
  virtual int getModelResultI(std::string OutputName);
  ...
  virtual bool getModelResultB(std::string OutputName);
  ...
  virtual int getStatus();
  virtual bool closeMLInterface();
};
```

Listing 2. ACPOMLInterface Header

Example (1/4)
ACPO Model for Loop Unroll

```
class ACPOLUModel : public ACPOModel {
public:
  ACPOLUModel(Loop *L, OptimizationRemarkEmitter *ORE, bool UseML = true)
      : ACPOModel(ORE, UseML), CurrentLoop(L), UnrollingFactor(0), UnrollingType(0) {
        std::vector<BasicBlock *> BBVec = L->getBlocksVector();
    ...
    setMLIF(createPersistentPythonMLIF());
  }
  ...
protected:
  // Get an Advice inferred from the ML Model
  virtual std::unique_ptr<ACPOAdvice> getAdviceML() override {
    std::shared_ptr<ACPOMLInterface> MLIF = getMLIF();
    MLIF->loadModel("model-lu.acpo");
    MLIF->setCustomFeatures("LU", CustomFeatureValues);
    bool ModelRunOK = MLIF->runModel("LU");
    ...
    UnrollingType = MLIF->getModelResultI("LU-Type");
    UnrollingFactor = MLIF->getModelResultI("LU-Count");
    ...
    // Form Advice object to return to caller
    std::unique_ptr<ACPOAdvice> Advice = std::make_unique<ACPOAdvice>();
    Advice->addField("LU-Count", ConstantInt::get(Type::getInt64Ty(*(getContextPtr())),(
     int64_t)UnrollingFactor));
    Advice->addField("LU-Type", ConstantInt::get(Type::getInt64Ty(*(getContextPtr())),(int64_t
     )UnrollingType));
    return Advice;
  }
  ...
};
```

Listing 3. ACPOLUModel Instantiation

Example (2/4)
ACPO Model for Loop Unroll :: Invocation

```
1  if (EnableACPOLU.getNumOccurrences() > 0 && LAA) {
2    std::unique_ptr<ACPOLUModel> LU = std::make_unique<ACPOLUModel>(L, ORE);
3    ...
4    LU->setCustomFeatures(...);
5    ...
6    std::unique_ptr<ACPOAdvice> Advice = LU->getAdvice();
7    Constant *ValType = Advice->getField("LU-Type");
8    Constant *Val = Advice->getField("LU-Count");
9    ConstantInt *IntVal = dyn_cast<ConstantInt>(Val);
10   ConstantInt *LUType = dyn_cast<ConstantInt>(ValType);
11   ...
12 }
```

Listing 4. ACPOLUModel Invocation

Example (3/4)
ACPO Model for Loop Unroll :: Architecture

Table 1. ACPO Loop Unrolling Model Architecture

Layer No	Layer (type)	Output Shape
0	Linear-FC1 Relu-0	[Features, 1, 32] [Features, 1, 32]
1	Linear-FC1 Relu-1	[-1, 1, 128] [-1, 1, 128]
2	Linear-FC2 Relu-2	[-1, 1, 256] [-1, 1, 256]
3	Linear-FC3 Relu-3	[-1, 1, 64] [-1, 1, 64]
4	Linear-FC4 Softmax	[-1, 1, Classes] [-1, 1, Classes]

Table 2. ACPOLUModel Features

No	Static Features	No	Static Features
1	PartialOptSizeThreshold	16	NumLoadInstPerLoopNest
2	AllowRemainder	17	NumStoreInstPerLoopNest
3	UnrollRemainder	18	TotLoopNestInstCount
4	AllowExpensiveTripCount	19	AvgNumLoadInstPerLoopNest
5	Force	20	AvgNumStoreInstPerLoopNest
6	TripCount	21	NumLoadInstPerLoop
7	MaxTripCount	22	NumStoreInstPerLoop
8	Size	23	TotLoopInstCount
9	InitialIVValueInt	24	AvgNumLoadInstPerLoop
10	FinalIVValueInt	25	AvgNumStoreInstPerLoop
11	StepValueInt	26	IsInnerMostLoop
12	NumPartitions	27	IsOuterMostLoop
13	IndVarSetSize	28	MaxLoopHeight
14	AvgStoreSetSize	29	TotBlocksPerLoop
15	AvgNumInsts	30	IsFixedTripCount

Table 3. Compile-time Overhead Breakdown

Task	Each Module	Each Loop
Feature Collection	0.000135	0.000135
Setting Features for ML Model	0.000029	0.000029
Total ML Inference Time	0.396148	0.00699
Unrolling Factor Assignment	0.000001	0.000001
Total Time (s)	**0.396313**	**0.007155**

Example (4/4)
ACPO Model for Loop Unroll :: Results

Table 5. ACPOLUModel Speedup on Polybench

Benchmark	Time O3	Var O3	Size O3	Time ACPO	Var ACPO	Size ACPO	**Speedup**	Size Bloat
correlation	2.623	0.29	72728	2.739	0.984	72768	0.957	1.005
covariance	2.743	0.81	72648	2.734	0.927	72728	**1.003**	1.001
gramschmidt	3.98	0.94	72752	4.371	0.972	72752	0.910	1
cholesky	2.477	0.03	72752	2.248	0.027	72832	**1.102**	1.001
atax	0.739	0.26	72720	0.749	0.169	72696	0.985	0.999
2mm	3.478	0.97	72608	3.533	0.385	138216	0.984	1.903
3mm	5.182	0.078	72608	5.267	0.752	138248	0.983	1.9040
bicg	1.371	0.025	72664	1.343	0.85	72664	**1.020**	1
doitgen	0.897	0.044	72680	0.887	0.015	72688	**1.011**	1.0001
mvt	2.050	0.077	72608	1.959	0.085	72608	**1.046**	1
gesummv	1.499	0.003	72616	1.311	0.026	72616	**1.143**	1
gemver	2.699	0.15	72616	2.816	0.151	72648	0.958	1.0004
symm	2.728	0.33	72648	2.572	0.140	72680	**1.060**	1.0004
syr2k	3.150	0.91	72608	3.176	0.667	72680	0.991	1.0009
syrk	1.500	0.972	72608	1.496	0.879	72648	**1.002**	1.0005
trmm	1.378	0.92	72648	1.456	0.930	72680	0.946	1.0004
ludcmp	5.040	0.83	72720	4.812	0.6637	72784	**1.047**	1.0008
lu	5.186	0.31	72696	5.299	0.9422	72816	0.978	1.001
trisolv	1.146	0.013	72648	0.937	0.0171	72648	**1.223**	1
adi	18.70	0.017	72664	18.478	0.0012	72680	**1.012**	1.0002
seidel-2d	32.346	0.08	72648	32.318	0.005	72688	**1.0008**	1.0005
deriche	0.271	0.58	72648	0.251	0.978	72784	**1.078**	1.0018
floyd-Warshall	28.303	0.06	72624	18.946	0.0217	72656	**1.493**	1.0004
nussinov	3.839	0.08	72696	3.277	0.12	203808	**1.171**	2.803
Geomean	-	**0.15**	-	-	**0.237**	-	**1.040725**	**1.10193**

Table 6. ACPOLUModel Generalization Performance

Benchmarks		O3				ACPO				**SpUp**
		Time	Var	Br Miss	Size	Time	Var	Br Miss	Size	
CoreMark		20.783	0.08%	242×10^6	74328	19.723	0.06%	90×10^6	74328	**1.054**
Coral-2	CLOMP	142.267	0.04%	1.2×10^9	78952	140.835	0.06%	1.1×10^9	78952	**1.010**
	Quicksilver	11.647	1.31%	121×10^6	256072	11.657	0.54%	82×10^6	527328	0.999
	STRIDE cac.	19.707	0.38%	23×10^6	72112	18.83	0.40%	3×10^6	72112	**1.047**
	STRIDE str.	44.842	0.04%	105×10^6	71928	44.792	0.06%	6×10^6	71928	**1.001**
	STREAM	18.899	**2.35%**	74×10^6	72992	18.786	0.91%	72×10^6	72992	**1.006***
	Phloem pre.	2.609	**6.21%**	50×10^6	131752	2.668	**7.83%**	50×10^6	182392	0.978*
	AMG	3.2922	0.44%	48×10^6	564192	2.785	0.36%	43×10^6	1292568	**1.182**
	LAMMPS	2.92	0.55%	7.4×10^9	3802424	2.826	0.57%	51×10^6	7082568	**1.033**
Graph500	BFS	7.275	**2.42%**	120×10^6	539264	7.33	**2.76%**	119×10^6	539240	0.992*
	SSSP	15.508	1.64%	313×10^6	539832	15.308	1.43%	316×10^6	605344	**1.013**

Example (1/2)
ACPO Model for Inline[1]

```
class ACPOFIModel : public ACPOModel {
public:
  ACPOLFIModel(CallBase *CB, InlineAdvisor *IA, OptimizationRemarkEmitter *ORE,
               bool OnlyMandatory, bool UseML = true)
      : ACPOModel(ORE, UseML), CurrentCB(CB), NotACPOAdvisor(IA), OnlyMandatory(OnlyMandatory)
      {
    Function *Caller = CB->getCaller();     ...
    setMLIF(createPersistentPythonMLIF());
  }
  ...
protected:
  // Get an Advice inferred from the ML Model
  virtual std::unique_ptr<ACPOAdvice> getAdviceML() override {
    std::shared_ptr<ACPOMLInterface> MLIF = getMLIF();
    MLIF->loadModel("model-fi.acpo");
    MLIF->setCustomFeatures("FI", CustomFeatureValues);
    bool ModelRunOK = MLIF->runModel("FI");
    ...
    ShouldInline = MLIF->getModelResultI("FI-ShouldInline");
    ...
    // Form Advice object to return to caller
    Advice->addField("FI-ShouldInline",
                     ConstantInt::get(Type::getInt64Ty(*(getContextPtr())),
                                      (int64_t)ShouldInline));
    return Advice;
  }
  ...
};
```

Listing 5. ACPOFIModel Instantiation

[1] A. Ashouri, M. Elhoushi, Y. Hua, X. Wang, M. Manzoor, B. Chan, and Y. Gao. “MLGOPerf: An ML Guided Inliner to Optimize Performance”. arXiv preprint arXiv:2207.08389 (2022).

Example (2/2)
ACPO Model for Inline :: invocation

```
1  /// helper function for getting advice with acpo infrastructure
2  bool getACPOAdvice(CallBase *CB, std::unique_ptr<ACPOFIModel> &FI,
3                     ModelDataFICollector *MDC, InlineAdvisor *Advisor) {
4   bool ShouldInline = false;
5   if (EnableACPOFI.getNumOccurrences() > 0) {
6     std::unique_ptr<ACPOFIModel> FI = std::make_unique<ACPOFIModel>(*CB, ORE);
7     ...
8     FI->setCustomFeatures(...);
9     ...
10    std::unique_ptr<ACPOAdvice> Advice = FI->getAdvice();
11    Constant *ValType = Advice->getField("FI-ShouldInline");
12    ConstantInt *ACPOInline = dyn_cast<ConstantInt>(Val);
13    ShouldInline = ACPOInline->getSExtValue();
14    return ShouldInline;
15  }
```

Listing 6. ACPOFIModel Invocation

ACPO Model Combined Results (Inline + Loop Unroll)

Table 8. ACPO Combined Function Inlining & Loop Unrolling Performance (Cbench)

Name	ACPO-FI		ACPO-LU		ACPO Combined	
	Size Bloat	Speedup	Size Bloat	Speedup	Size Bloat	**Speedup**
automotive_bitcount	1.000	0.999	1.001	1.003	1	**1.001**
automotive_qsort1	1.000	0.994	1.000	0.982	1.91	**0.989**
automotive_susan_c	1.000	0.999	3.685	1.035	19.797	**1.066**
automotive_susan_e	1.000	1.023	3.685	1.019	19.797	**1.026**
automotive_susan_s	1.000	0.980	3.685	1.038	19.797	**0.965**
bzip2d	1.441	0.991	2.224	1.010	3.541	**1.014**
bzip2e	1.441	1.007	2.350	1.001	3.546	**1.015**
consumer_jpeg_c	1.288	1.017	5.385	0.986	5.086	**1.027**
consumer_jpeg_d	1.289	0.923	5.400	1.073	4.807	**1.026**
consumer_lame	1.437	1.027	4.501	1.030	4.729	**1.034**
consumer_mad	1.217	1.071	4.321	0.990	3.193	**0.968**
consumer_tiff2bw	1.180	1.002	4.502	0.943	2.471	**0.987**
consumer_tiff2rgba	1.180	1.250	4.681	1.036	2.657	**1.012**
consumer_tiffdither	1.180	1.046	1.213	0.967	2.655	**1.002**
consumer_tiffmedian	1.180	1.017	1.000	0.970	3.021	**0.943**
network_dijkstra	1.000	0.973	1.000	0.990	1	**1.037**
network_patricia	1.000	0.970	1.000	0.974	1	**1.004**
office_ghostscript	1.275	1.000	1.000	1.000	1.12	**1.001**
office_ispell	0.996	1.000	1.000	1.000	1	**1**
office_rsynth	1.000	1.000	2.024	1.010	2.024	**1.011**
office_stringsearch1	1.000	1.006	1.000	1.000	1	**1**
security_blowfish_d	1.000	1.002	1.000	0.978	1	**0.965**
security_blowfish_e	1.000	1.003	1.000	1.015	1	**0.981**
security_pgp_d	1.861	0.993	1.000	0.991	13.144	**1**
security_pgp_e	1.861	1.017	1.000	1.006	13.144	**1.006**
security_rijndael_d	1.000	0.990	3.823	0.966	1	**0.966**
security_rijndael_e	1.000	1.001	3.823	0.974	1	**0.995**
security_sha	0.999	1.270	1.000	1.174	0.999	**1.501**
telecom_adpcm_c	1.000	1.000	1.000	1.008	1	**1.038**
telecom_adpcm_d	1.000	0.997	1.000	1.132	1	**1.138**
telecom_CRC32	1.000	1.094	1.000	1.042	1	**1.112**
telecom_gsm	1.810	1.076	1.815	1.15	1.81	**1.117**
Geomean	1.152	**1.021**	1.818	**1.01**	2.442	**1.024**

Summary

- Scalability of ACPO
  - Bringing ML into compilers for compiler Engineers
  - Provides the ability to reproduce models on different architecture
  - Methodology lays out the steps to regenerate data, retrain a new model, and deploy with LLVM
  - Leveraging AOT infrastructure in LLVM (C++ & Python support)
- Multi-objective optimization
  - Trade-off between performance, code size, and power consumption
- Open Source
  - https://github.com/Huawei-CPLLab/ACPO
    - Models and scripts
    - Python abstraction layer
      - Tensorflow
      - We are working on adding PyTorch support
  - https://gitee.com/openeuler
    - Compiler side infrastructure, made agnostic of underlying ML framework
    - Feature collectors for our models, easily extendible
      - 80+ features we used for effective performance-oriented model design

Thank you

www.huawei.com

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.