Applying Machine Learning Techniques on HPX Parallel Algorithms

STE||AR GROUP @ Louisiana State University

LSU

Zahra Khatami, Lukas Troska, Hartmut Kaiser and J. Ramanujam
Center for Computation and Technology, Louisiana State University
The STE||AR Group, http://stellar-group.org

Objectives

Manually parallelizing all loops within an application may result in degrading performance, as some of the loops cannot scale desirably on a lager number of threads. We illustrate how machine learning techniques can be applied to address this challenge. Our main goal here is to:

✓ Automatically determining execution policy by considering static compiler information and dynamic runtime characteristics implemented within a learning model.

Proposed Method

Our technique for determining the execution path (sequential or parallel) had 4 stages as shown below:

1. Design of Learning Model
2. Special Execution Policy
3. Features Extraction
4. Implement Learning Model

I. Design of Learning Model

- Model: Binary Logistic Regression Model
- Output: Sequential or parallel

Updating weights: $W^T = [\omega_0, \omega_1, \omega_2, ...]$

$\omega_{k+1} = (X^T S_k X)^{-1} X^T (S_k X \omega_k + y - \mu_k)$

Experiments: $X(i) = [1, x_1(i), x_2(i), ...]^T$

$S(i,i) = \mu(i)(1 - \mu(i))$

Bernoulli distribution value:
$\mu(i) = 1/(1 + e^{-W^T x(i)})$

Decision rule:
$y(x) = 1 \longleftrightarrow p(y = 1|x) > 0.5$

II. Special Execution Policy

- **NEW** execution_policy $\rightarrow$ *par_if*.

✓ Passing *par_if* as an execution policy triggers the compiler to insert a hook for the learning model.

Example:

$for\_each(par\_if,$ //$ExecutionPolicy$
$range.begin(), range.end(),$ //$Range$
$lambda);$ //$LambdaFunction$

III. Feature Extraction

✓ Introducing new ClangTool named *ForEachCallHandler* to extract features from the loop.

```
virtual void
run(const MatchFinder :: Result& result)
{
 ...
 if (policy.find("par_if")! = string :: npos)
      extract_features(lambda_body);
 ...
}
```

✓ Features extracted:

1. Number of threads (dynamic)
2. Number of iterations (dynamic)
3. Number of total operations (static)
4. Number of float operations (static)
5. Number of comparison operations (static)
6. Deepest loop level (static)

Contact Information

Web https://github.com/STEllAR-GROUP/hpxML
Email z.khatami88@gmail.com, hkaiser@cct.lsu.edu
Supported by NSF award 1339782.

IV. Learning Model Implementation

```
bool seq_par(F &&features) {
  return costs_fnc(features,
      retrieving_leraning_weights()); }
```

✓ **NEW** function: *seq_par* $\rightarrow$ making runtime choosing loop's parameters by considering static and dynamic features in costs_fnc cost function.

- New function seq_par passes the extracted features to the runtime.
- Clang adds extra lines with seq_par function within a user's code automatically.
- seq_par makes runtime to decide whether execute a loop sequentially or parallel.
- The attached parameters and executors can be reattached to the final determined execution policy.

Before compilation:

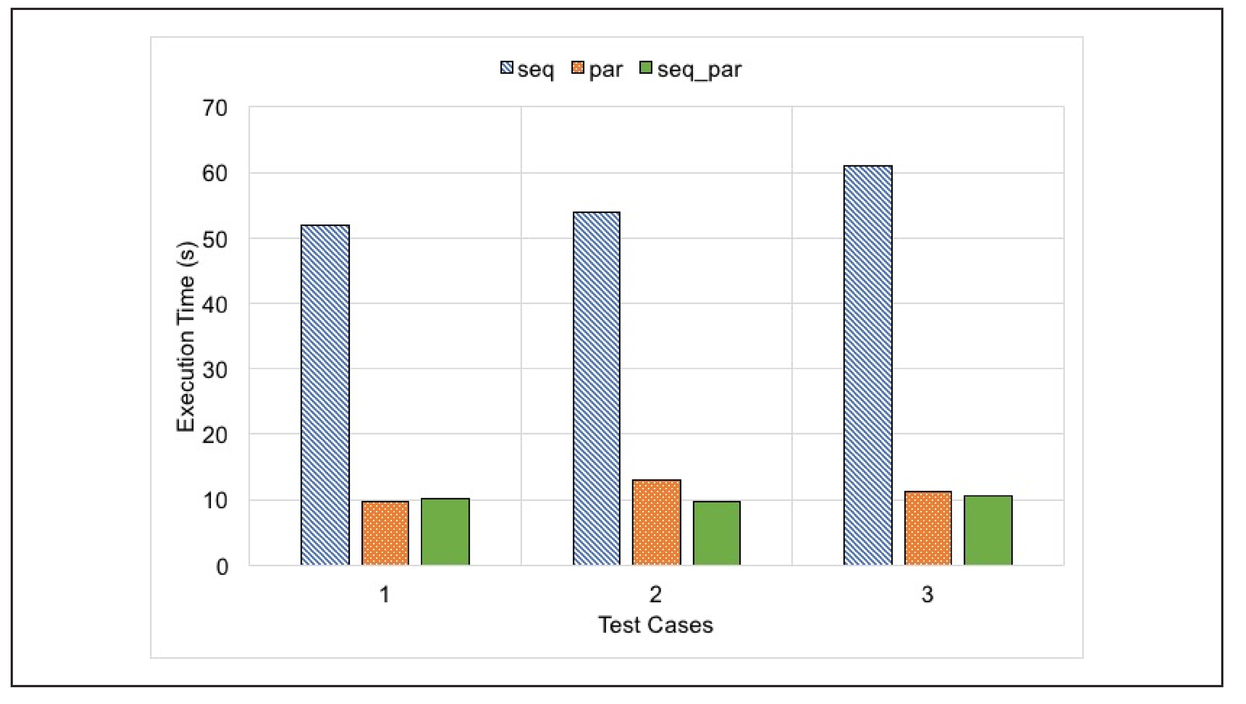
```
for_each(par_if,
        range.begin(), range.end(),
        lambda_fnc);
```

After compilation:

```
if (seq_par({f_0, ...f_n})) //Extracted Features
  for_each(seq,
        range.begin(), range.end(),
        lambda_fnc);
else
  for_each(par,
        range.begin(), range.end(),
        lambda_fnc);
```

Experimental Results

Test	Loop	Itr.	Total opr.	Float opr.	Cmpr. opr.	level	Policy
1	l_1	10000	400100	200000	101010	2	par
	l_2	20000	450026	250000	150503	2	par
	l_3	20000	502040	250000	103051	2	par
	l_4	500	550402	200000	150102	1	par
2	l_1	150000	350106	101010	500	2	par
	l_2	100	10050016	5000000	2505013	3	seq
	l_3	100	25000000	3010204	1500204	3	seq
	l_4	50000	4000450	200000	100150	1	par
3	l_1	500	4504030	250000	150300	2	par
	l_2	400	3502020	200000	100405	1	par
	l_3	2000	250033	150000	103040	3	seq
	l_4	2500	350400	150000	100600	3	seq



Execution time comparisons.

- Clang 4.0.0, HPX 0.9.99, Intel Xeon E5-2630, 8 cores, $2.4GHZ$.

1. Optimizes HPX application performance by predicting optimum execution policy for each loop.
2. Both dynamic and static information are considered to provide sufficient optimizations.

✓ $15\% - 20\%$ improvement.