Applying Logistic Regression Model on HPX Parallel Loops

Zahra Khatami
Lukas Troska
Hartmut Kaiser
J. Ramanujam

Louisiana State University
The STE||AR Group, http://stellar-group.org

15th Charm++ Workshop
Outline

Motivation

HPX

HPX Current Challenges

Proposed Methods

Experimental Results

Conclusion
Motivation

- Loop-level parallelism.
  1. Some of the loops cannot scale desirably to a large number of threads.
  2. Overheads of manually tuning loop parameters.
- Considering both dynamic runtime and static compile time information to achieve maximal parallel performance.
✓ Parallel C++ runtime system.
✓ Enabling fine-grained task parallelism: Resulting in a better load balancing.
✓ Providing efficient scalable parallelism.
✓ Reducing SLOW factors:
  1. Starvation,
  2. Latencies,
  3. Overhead,
  4. Waiting.

---

Motivation

HPX

HPX Current Challenges

Proposed Methods

Experimental Results

Conclusion

Louisiana State University

**HPX**

(a)

(b)

Time

Barrier

Active

Waiting

Locality 1

future.get()

suspend thread 1

execute thread 2

reactivate thread 1

Locality 2

Zahra Khatami

15th Charm++ Workshop

Logistic Regression Model on HPX Loops
HPX Current Challenges

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
<th>Implemented by</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>sequential execution</td>
<td>Parallelism TS, HPX</td>
</tr>
<tr>
<td>par</td>
<td>parallel execution</td>
<td>Parallelism TS, HPX</td>
</tr>
<tr>
<td>par_vec</td>
<td>parallel and vectorized execution</td>
<td>Parallelism TS</td>
</tr>
<tr>
<td>seq(task)</td>
<td>sequential and asynchronous execution</td>
<td>HPX</td>
</tr>
<tr>
<td>par(task)</td>
<td>parallel and asynchronous execution</td>
<td>HPX</td>
</tr>
</tbody>
</table>

execution_policy: specifying execution restrictions of the work items:

- sequential execution policy: run sequentially.
- parallel execution policy: run in parallel.

Problem: Manually selecting execution policies for executing HPX parallel algorithms\(^1\).

• chunk_sizes: Overheads of determining chunk size\textsuperscript{1}:
  1 \texttt{auto\_partitioner}: exposed by the HPX algorithms.
  2 \texttt{static/dynamic chunk}: execution policy’s parameter.

✓ Automating parameters selections by considering loops characteristics implemented in a learning model.
✓ Combining machine learning technique, compiler and runtime methods for utilizing maximum resource availability.
Proposed Method

1. Designing Learning Model
2. Special Execution Policy
3. Feature Extraction: Collecting static and dynamic features
4. Learning Model Implementation

---

Designing Learning Model

✓ Logistic regression models

- execution_policy: Binary logistic regression model.
- chunk_sizes: Multinomial logistic regression model.

1https://github.com/STEllARGROUP/hpxML/LearningAlgorithm
Binary Logistic Regression Model

- Output = Sequential or parallel

Updating weights: \( W^T = [\omega_0, \omega_1, \omega_2, \ldots] \)

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

Experiments: \( X(i) = [1, x_1(i), x_2(i), \ldots]^T \)

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

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

Decision rule: \( y(x) = 1 \iff p(y = 1|x) > 0.5 \)
Multinomial Logistic Regression Model

- Output = Efficient chunk size → 0.001, 0.01, 0.1, and 0.5 of the loop’s iteration.

Updating weights: \( \omega_{\text{new}} = \omega_{\text{old}} - H^{-1} \nabla E(\omega) \)

Cross entropy error function:
\[
E(\omega_1, \omega_2, ..., \omega_C) = - \sum_{n=1}^{N} \sum_{c=1}^{C} t_{nc} \ln y_{nc}
\]

\[
y_{nc} = y_c(X_n) = \frac{\exp(W_c^T X_n)}{\sum_{i=1}^{C} \exp(W_i^T X_n)}
\]

Hessian matrix:
\[
\nabla_{\omega_i} \nabla_{\omega_j} E(\omega_1, \omega_2, ..., \omega_C) = \sum_{n=1}^{N} y_{ni}(l_{ij} - y_{nj})X_nX_n^T
\]
✓ Machine Learning

Machine Learning

W_BLR
W_MLR
✓ Applying it on a loop makes implementing learning model on that loop.

- execution_policy $\rightarrow$ par_if (execution policy).
- chunk_sizes $\rightarrow$ adaptive_chunk_size() (execution policy’s parameter).

```cpp
for_each(par_if, range.begin(), range.end(), lambda);

for_each(policy.with(adaptive_chunk_size), range.begin(), range.end(), lambda);
```
✓ Introducing new ClangTool named `ForEachCallHandler`.

```cpp
virtual void run(const MatchFinder::MatchResult &Result) {
    ...
    if (policy_string.find("par-if") != string::npos ||
        policy_string.find("adaptive_chunk_size") != string::npos) {
        extract_features(lambda_body);
        ...
    }
}
```
# Feature Extraction

<table>
<thead>
<tr>
<th>Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic</td>
<td>number of threads</td>
</tr>
<tr>
<td>dynamic</td>
<td>number of iterations</td>
</tr>
<tr>
<td>static</td>
<td>number of total operations</td>
</tr>
<tr>
<td>static</td>
<td>number of float operations</td>
</tr>
<tr>
<td>static</td>
<td>number of comparison operations</td>
</tr>
<tr>
<td>static</td>
<td>deepest loop level</td>
</tr>
<tr>
<td>static</td>
<td>number of integer variables</td>
</tr>
<tr>
<td>static</td>
<td>number of float variables</td>
</tr>
<tr>
<td>static</td>
<td>number of if statements</td>
</tr>
<tr>
<td>static</td>
<td>number of if statements within inner loops</td>
</tr>
<tr>
<td>static</td>
<td>number of function calls</td>
</tr>
<tr>
<td>static</td>
<td>number of function calls within inner loops</td>
</tr>
</tbody>
</table>

### Feature Selection

<table>
<thead>
<tr>
<th>Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic</td>
<td>number of threads*</td>
</tr>
<tr>
<td>dynamic</td>
<td>number of iterations*</td>
</tr>
<tr>
<td>static</td>
<td>number of total operations*</td>
</tr>
<tr>
<td>static</td>
<td>number of float operations*</td>
</tr>
<tr>
<td>static</td>
<td>number of comparison operations*</td>
</tr>
<tr>
<td>static</td>
<td>deepest loop level*</td>
</tr>
<tr>
<td>static</td>
<td>number of integer variables</td>
</tr>
<tr>
<td>static</td>
<td>number of float variables</td>
</tr>
<tr>
<td>static</td>
<td>number of if statements</td>
</tr>
<tr>
<td>static</td>
<td>number of if statements within inner loops</td>
</tr>
<tr>
<td>static</td>
<td>number of function calls</td>
</tr>
<tr>
<td>static</td>
<td>number of function calls within inner loops</td>
</tr>
</tbody>
</table>

* Features selected with implementing decision tree classification technique\(^1\).

Learning Model Implementation

✓ `seq_par` & `chunk_size_determination`: making runtime choosing loop’s parameters by considering static and dynamic features in costs_fnc cost function.

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

dynamic_chunk_size chunk_size_determination(F &&features)
{
    return costs_fnc(features, retrieving_MLR_weights());
}
```
✓ Machine Learning & Compiler
Learning Model Implementation

Before compilation:

```cpp
for_each (par_if, range.begin(), range.end(), lambda);
for_each (policy.with(adaptive_chunk_size), range.begin(), range.end(), lambda);
```

After compilation:

```cpp
if (seq_par(EXTRACTED_STATIC_DYNAMIC_FEATURES))
    for_each (seq, range.begin(), range.end(), lambda);
else
    for_each (par, range.begin(), range.end(), lambda);

for_each (policy.with(chunk_size_determination(EXTRACTED_STATIC_DYNAMIC_FEATURES)), range.begin(), range.end(), lambda);
```
✓ Machine Learning & Compiler & Runtime
## Experimental Results

<table>
<thead>
<tr>
<th>Item</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon E5-2630</td>
</tr>
<tr>
<td>Compiler</td>
<td>Clang 4.0.0</td>
</tr>
<tr>
<td>Cores</td>
<td>8</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.4GHZ</td>
</tr>
<tr>
<td>OS</td>
<td>32 bit Linux Mint 17.2</td>
</tr>
<tr>
<td>HPX</td>
<td>0.9.99</td>
</tr>
<tr>
<td>Main Memory</td>
<td>65GB</td>
</tr>
</tbody>
</table>
## Experimental Results

<table>
<thead>
<tr>
<th>Test</th>
<th>Loop</th>
<th>ltr.</th>
<th>Total opr.</th>
<th>Float opr.</th>
<th>Cmpr. opr.</th>
<th>level</th>
<th>Policy</th>
<th>% Chunk size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$l_1$</td>
<td>10000</td>
<td>400100</td>
<td>200000</td>
<td>101010</td>
<td>2</td>
<td>par (8)</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>20000</td>
<td>450026</td>
<td>250000</td>
<td>150503</td>
<td>2</td>
<td>par (8)</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>$l_3$</td>
<td>20000</td>
<td>502040</td>
<td>250000</td>
<td>103051</td>
<td>2</td>
<td>par (8)</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>$l_4$</td>
<td>500</td>
<td>550402</td>
<td>200000</td>
<td>150102</td>
<td>1</td>
<td>par (8)</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>$l_1$</td>
<td>150000</td>
<td>350106</td>
<td>101010</td>
<td>500</td>
<td>2</td>
<td>par (8)</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>100</td>
<td>10050016</td>
<td>5000000</td>
<td>2505013</td>
<td>3</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>$l_3$</td>
<td>100</td>
<td>2500000</td>
<td>3010204</td>
<td>1500204</td>
<td>3</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>$l_4$</td>
<td>50000</td>
<td>4000450</td>
<td>200000</td>
<td>100150</td>
<td>1</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>$l_1$</td>
<td>500</td>
<td>4504030</td>
<td>250000</td>
<td>150300</td>
<td>2</td>
<td>par (8)</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>400</td>
<td>3502020</td>
<td>200000</td>
<td>100405</td>
<td>1</td>
<td>par (8)</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>$l_3$</td>
<td>2000</td>
<td>250033</td>
<td>150000</td>
<td>103040</td>
<td>3</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>$l_4$</td>
<td>2500</td>
<td>350400</td>
<td>150000</td>
<td>100600</td>
<td>3</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>$l_1$</td>
<td>20000</td>
<td>204002</td>
<td>100000</td>
<td>10320</td>
<td>2</td>
<td>par (8)</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>30000</td>
<td>400000</td>
<td>150102</td>
<td>100000</td>
<td>2</td>
<td>par (8)</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>$l_3$</td>
<td>300</td>
<td>550000</td>
<td>44000</td>
<td>20030</td>
<td>3</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>$l_4$</td>
<td>400</td>
<td>450000</td>
<td>50400</td>
<td>10602</td>
<td>3</td>
<td>seq</td>
<td>0.1</td>
</tr>
<tr>
<td>5</td>
<td>$l_1$</td>
<td>200</td>
<td>4502001</td>
<td>150000</td>
<td>101004</td>
<td>3</td>
<td>par (8)</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>700</td>
<td>400020</td>
<td>300000</td>
<td>150006</td>
<td>3</td>
<td>par (8)</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>$l_3$</td>
<td>300</td>
<td>302020</td>
<td>20000</td>
<td>14005</td>
<td>2</td>
<td>par (8)</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>$l_4$</td>
<td>100</td>
<td>50400</td>
<td>20000</td>
<td>10110</td>
<td>2</td>
<td>seq</td>
<td>0.1</td>
</tr>
</tbody>
</table>
As all 4 execution policy determined for the first test is par, the overhead of the costs_fnc resulted in degrading performance.

✓ 15% – 20% improvement.
45%, 32%, 37% and 58% improvement over setting chunks to be 0.001, 0.01, 0.1, or 0.5 iterations.
Conclusion

- https://github.com/STEllAR-GROUP/hpxML
- Join our IRC channel #ste|ar if you need any help 😊.
Thanks for your attention!
Questions?