Tool-Supported Mini-App Extraction to Facilitate Program Analysis and Parallelization

International Conference on Parallel Processing (ICPP ‘21)
Aug. 10th 2021, Virtual Conference

Jan-Patrick Lehr  
@jplehr
jan-patrick.lehr@tu-darmstadt.de

Christian Bischof, Florian Dewald, Heiko Mantel, Mohammad Norouzi, Felix Wolf

github.com/tudasc
github.com/discopop-project
Motivation

- The vastness of today’s software challenges manual and tool-supported analysis, for example, for correctness or parallelization.

- Mini-apps are an intriguing vehicle to reduce complexity while maintaining key properties of the original application.

- Creation of mini-apps challenging and time-consuming
  - Requires expertise in domain and computer science.
Approach Overview

Tool-support
1) PIRA [1] for automatic kernel detection
2) Novel Clang-based source-to-source translator
3) Type-safe checkpoint restart using TyCart [2] / TypeART [3]
4) Hierarchical clustering of execution metrics (Caliper [4] / PAPI [5], PIN [6])
Approach Overview

Tool-support
1) PIRA [1] for automatic kernel detection
2) Novel Clang-based source-to-source translator
3) Type-safe checkpoint restart using TyCart [2] / TypeART [3]
4) Hierarchical clustering of execution metrics (Caliper [4] / PAPI [5], PIN [6])
## Approach Overview

**Tool-support**

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])
Approach Overview

**Tool-support**

1) **PIRA** [1] for automatic kernel detection
2) Novel **Clang**-based source-to-source translator
3) Type-safe checkpoint restart using **TyCart** [2] / **TypeART** [3]
4) Hierarchical clustering of execution metrics (**Caliper** [4] / **PAPI** [5], **PIN** [6])
Approach Overview

Tool-support
1) PIRA [1] for automatic kernel detection
2) Novel Clang-based source-to-source translator
3) Type-safe checkpoint restart using TyCart [2] / TypeART [3]
4) Hierarchical clustering of execution metrics (Caliper [4] / PAPI [5], PIN [6])
I. Kernel Identification

- PIRA automatically filters short-running functions from measurement
- PIRA uses Extra-P [7] to construct empirical performance models

- Performance models are used to extrapolate function runtimes for larger inputs
  - Determines points in the program that limit scaling
  - Shows parts that benefit from (additional) parallelism, due to increased workload
II. Source Extraction

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion

- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

```c
double some_const; // required global variable
float compute_X(); // non-system function
void kernel(int i, double *pd, double *r) {
    // performs heavy computation with pd
    r = some_const * compute_X() * ...;
}

double* compute(int arg_i, double* arg_d) {
    double *res = malloc(sizeof(double));
    kernel(arg_i, arg_d, res); // kernel call-site
    return res;
}
```
II. Source Extraction

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion

- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

```c
1   double some_const; // required global variable
2   float compute_X(); // non-system function
3   void kernel(int i, double *pd, double *r) {
4       // performs heavy computation with pd
5       r = some_const * compute_X() * ...;
6   }
7   double* compute(int arg_i, double* arg_d) {
8       double *res = malloc(sizeof(double));
9       kernel(arg_i, arg_d, res); // kernel call-site
10      return res;
11 }
```
II. Source Extraction

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion

- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

```c
1 double some_const; // required global variable
2 float compute_X(); // non-system function
3 void kernel(int i, double *pd, double *r) {
   // performs heavy computation with pd
   r = some_const * compute_X() * ...;
4 }
5 double* compute(int arg_i, double* arg_d) {
6   double *res = malloc(sizeof(double));
7   kernel(arg_i, arg_d, res); // kernel call-site
8   return res;
9 }
10
11
```
II. Source Extraction

- **Clang**-based source extractor
  - Works in a mark-and-sweep fashion

- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

```c
1  double some_const; // required global variable
2  float compute_X(); // non-system function
3  void kernel(int i, double *pd, double *r) {
4      // performs heavy computation with pd
5      r = some_const * compute_X() * ...;
6  }
7  double* compute(int arg_i, double* arg_d) {
8      double *res = malloc(sizeof(double));
9      kernel(arg_i, arg_d, res); // kernel call-site
10     return res;
11 }
```
II. Source Extraction

- **Clang-based source extractor**
  - Works in a mark-and-sweep fashion

- From call site identify and mark: functions, variables and types by analyzing the **Clang** abstract syntax tree (AST)

- All marked entities are extracted into a new source file
  - Includes are maintained or hints are displayed to the user which files are required

```c
1 double some_const;  // required global variable
2 float compute_X();  // non-system function
3 void kernel(int i, double *pd, double *r) {
4     // performs heavy computation with pd
5     r = some_const * compute_X() * ...;
6 }
7 double* compute(int arg_i, double* arg_d) {
8     double *res = malloc(sizeof(double));
9     kernel(arg_i, arg_d, res);  // kernel call-site
10    return res;
11 }
```
III. Data Capturing

- Use type-safe checkpoint-restart (CPR) interface provided by **TyCart**
  - Built on top of **TypeART** with one sequential as well as **VeloC** [8] and **FTI** [9] backends

- Calls to CPR inserted into *kernel wrapper* to capture (original app) or restore (mini-app) application-level data
  - **TyCart** allows to query for type and length of an allocation
IV. Validation

- Representativeness validation by performing hierarchical clustering of execution metrics
- Hardware performance counters for functions and kernel region
- Dynamic instruction mix for full application execution

![Diagram](https://example.com/diagram.png)
Evaluation

Apply to astrophysics simulation of ~8.5m lines of code

- PIRA performs four iterations and identifies three kernels.
- Flat profile shows clearly the dominating functions.
- Performance models are – as expected – in the number of Monte-Carlo points evaluated.

<table>
<thead>
<tr>
<th>Function</th>
<th>Runtime (s)</th>
<th>% Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inclusive</td>
<td>Exclusive</td>
</tr>
<tr>
<td>main</td>
<td>490.0</td>
<td>1.2</td>
</tr>
<tr>
<td>Int_NN_HF</td>
<td>488.9</td>
<td>1.2</td>
</tr>
<tr>
<td>f_NN_HF</td>
<td>488.1</td>
<td>359.9</td>
</tr>
<tr>
<td>get_qNN</td>
<td>128.2</td>
<td>26.3</td>
</tr>
<tr>
<td>SphericalYN</td>
<td>101.9</td>
<td>101.9</td>
</tr>
</tbody>
</table>

performance model

\[-C_1 + 4.2^{-5} \times N\]
\[-C_2 + 2.3^{-5} \times N\]
\[-C_3 + 8.5^{-5} \times (N^{\frac{3}{4}}) \times \log_2(N)\]
Evaluation

Apply to astrophysics simulation of ~8.5m lines of code

- **PIRA** performs four iterations and identifies three kernels.
- Flat profile shows clearly the dominating functions.
- Performance models are – as expected – in the number of Monte-Carlo points evaluated.

<table>
<thead>
<tr>
<th>Function</th>
<th>Runtime (s)</th>
<th>% Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inclusive</td>
<td>Exclusive</td>
</tr>
<tr>
<td>main</td>
<td>490.0</td>
<td>1.2</td>
</tr>
<tr>
<td>Int_NN_HF</td>
<td>488.9</td>
<td>1.2</td>
</tr>
<tr>
<td>f_NN_HF</td>
<td>488.1</td>
<td>359.9</td>
</tr>
<tr>
<td>get_qNN</td>
<td>128.2</td>
<td>26.3</td>
</tr>
<tr>
<td>SphericalY</td>
<td>101.9</td>
<td>101.9</td>
</tr>
</tbody>
</table>

Performance model:
- \(-C_1 + 4.2^{-5} \times N\)
- \(-C_2 + 2.3^{-5} \times N\)
- \(-C_3 + 8.5^{-5} \times (N^{\frac{3}{4}}) \times \log_2(N)\)
Evaluation

Apply to astrophysics simulation of ~8.5m lines of code

- **PIRA** performs four iterations and identifies three kernels.
- Flat profile shows clearly the dominating functions.
- Performance models are – as expected – in the number of Monte-Carlo points evaluated.

<table>
<thead>
<tr>
<th>Function</th>
<th>Inclusive (s)</th>
<th>Exclusive (s)</th>
<th>% Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>490.0</td>
<td>1.2</td>
<td>0.3 %</td>
</tr>
<tr>
<td>Int_NN_HF</td>
<td>488.9</td>
<td>1.2</td>
<td>0.3 %</td>
</tr>
<tr>
<td>f_NN_HF</td>
<td>488.1</td>
<td>359.9</td>
<td>73.5 %</td>
</tr>
<tr>
<td>get_qNN</td>
<td>128.2</td>
<td>26.3</td>
<td>5.4 %</td>
</tr>
<tr>
<td>SphericalY</td>
<td>101.9</td>
<td>101.9</td>
<td>20.8 %</td>
</tr>
</tbody>
</table>

Function | performance model |
---------|-------------------|
\( f_{\text{NN\_HF}} \) | \(-C_1 + 4.2^{-5} \times N\) |
\( \text{get\_qNN} \) | \(-C_2 + 2.3^{-5} \times N\) |
\( \text{SphericalY} \) | \(-C_3 + 8.5^{-5} \times (N^{3/4}) \times \log_2(N)\) |
Evaluation

- **Clang**-based source-to-source translator created Mini-App
  - Manual addition of two includes (using the translator hints)

- Manual addition of Monte-Carlo library code results in MCS Mini-App
  - To expose more potential parallelism to **DiscoPoP**

Reduction achieved (MCS Mini-App vs Original):
- Lines of Code: **7674 x**
- Functions: **380 x**
- Variables: **3.4 x**
- Types: **4.9 x**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Original</th>
<th>Mini-App</th>
<th>MCS Mini-App</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td>8,571,815</td>
<td>422</td>
<td>1,117</td>
</tr>
<tr>
<td>Functions</td>
<td>31,196</td>
<td>46</td>
<td>82</td>
</tr>
<tr>
<td>- Reachable</td>
<td>445</td>
<td>33</td>
<td>52</td>
</tr>
<tr>
<td>Variables</td>
<td>1,206</td>
<td>183</td>
<td>356</td>
</tr>
<tr>
<td>Types</td>
<td>462</td>
<td>18</td>
<td>94</td>
</tr>
</tbody>
</table>
Evaluation

- **Clang**-based source-to-source translator created Mini-App
  - Manual addition of two includes (using the translator hints)

- Manual addition of Monte-Carlo library code results in MCS Mini-App
  - To expose more potential parallelism to **DiscoPoP**

Reduction achieved (MCS Mini-App vs Original):
- Lines of Code: **7674 x**
- Functions: **380 x**
- Variables: **3.4 x**
- Types: **4.9 x**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Original</th>
<th>Mini-App</th>
<th>MCS Mini-App</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td>8,571,815</td>
<td>422</td>
<td>1,117</td>
</tr>
<tr>
<td>Functions</td>
<td>31,196</td>
<td>46</td>
<td>82</td>
</tr>
<tr>
<td>– Reachable</td>
<td>445</td>
<td>33</td>
<td>52</td>
</tr>
<tr>
<td>Variables</td>
<td>1,206</td>
<td>183</td>
<td>356</td>
</tr>
<tr>
<td>Types</td>
<td>462</td>
<td>18</td>
<td>94</td>
</tr>
</tbody>
</table>
**Evaluation**

- Using **Caliper** to store **PAPI** in JSON format
- Kernel regions are similar in dendrogram
- Dynamic instruction mix is almost identical
  - Some differences in, e.g., system calls, due to more file I/O in original application

```
Smaller Ward distance is better for regions with the same name
```

```
Optimal: light and gray bar are equal in height.
```
Parallelization

- Used the MCS Mini-App for tool-supported parallelization with DiscoPoP [10]
  - Identified 42 unique parallelization opportunities in the mini-app
  - Focus on “DoAll” opportunities

- Speedup achieved: 35% w/ 4 threads
- Correctness manually verified in Isabelle/HOL

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Containing Function</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>get_NN_Pot</td>
<td>N</td>
</tr>
<tr>
<td>442</td>
<td>get_qNN</td>
<td>Y</td>
</tr>
<tr>
<td>474</td>
<td>f_NN_HF</td>
<td>Y</td>
</tr>
<tr>
<td>482</td>
<td>f_NN_HF</td>
<td>Y</td>
</tr>
<tr>
<td>490</td>
<td>f_NN_HF</td>
<td>Y</td>
</tr>
</tbody>
</table>
Summary / Future Work

- Tool-supported approach for mini-app extraction from large-scale applications
  - **PIRA**: automatic kernel identification
  - Source-to-source translator for code extraction
  - **TyCart**: type-safe checkpoint-restart
  - Hierarchical cluster representative-analysis

- Application to astrophysics simulation and subsequent tool-supported parallelization achieved speed-up of 35%

Future: Improve source-to-source translator for C++ w/ multi translation-unit support

Software Available

- [github.com/tudasc](https://github.com/tudasc)
- [github.com/discopop-project](https://github.com/discopop-project)
References

- [8] Nicolae, B. et al.: VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale, 2019. DOI: 10.1109/IPDPS.2019
- [10] Li, Z et al.: Unveiling Parallelization Opportunities in Sequential Programs, 2016. DOI: 10.1016/j.jss.2016.03.045