Performance and Usability Evaluation of a Pattern-Oriented Parallel Programming Interface for Multi-Core Architectures

Authors: Dalvan Griebler, Daniel Adornes, Luiz Gustavo Fernandes

E-mail: dalvan.griebler@acad.pucrs.br

Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Programa de Pós-Graduação em Ciência da Computação - PPGCC
Grupo de Modelagem de Aplicações Paralelas - GMAP

Software Engineering and Knowledge Engineering - SEKE 2014

July 2014
Outline

1. Introduction
2. Related Work
3. DSL-POPP
4. Usability Evaluation
5. Performance Experiments
6. Conclusions
7. References
Introduction

Motivation and Challenge
- Computer architectures are parallel
- Programming is not easy and requires some effort
- High performance and scalability in parallel programs

Goal
- A pattern-oriented interface to reduce programming effort and achieve good performance

Contributions
- A new master/slave programming interface for DSL-POPP
- Usability experiment (DSL-POPP vs Pthreads)
- Performance experiment (4 applications)
Related Work

Domain-Specific Languages
- Delite [1]

Pattern/Skeleton-Based Programming
- Introduced By Murray Cole [2]
- $CO_2P_3S$ [3]
- TBB (Thread Building Blocks) [4]
- FastFlow [5]

Other Programming Interfaces
- OpenMP [6]
- Charm++ [7]
- Cilk++ [8]
Domain-Specific Language for Pattern-Oriented Parallel Programming (DSL-POPP)

**POPP model [9]**

- **It is**: a standard way to implement parallel patterns in routines that uses code blocks to design parallel applications.

- **The goal**: is that developers explicitly implement parallel programs oriented by a set of patterns pre-implemented in a high-level programming interface.

- **Features**: define how to implement a pattern, combine, nesting pattern and achieve levels of parallelism.
Domain-Specific Language for Pattern-Oriented Parallel Programming (DSL-POPP)

Master/Slave pattern represented in the POPP model

(a) Nesting

(b) Levels of parallelism
Domain-Specific Language for Pattern-Oriented Parallel Programming (DSL-POPP)

**DSL-POPP: Master/Slave Programming Interface**

- **buffer**: channel to communicate master and slave
- **size**: the size of the buffer
- **num_threads**: total number of slave threads
- **policy**: load balancing policy

```c
$MasterSlavePattern int func_name(){
    @Master(void *buffer, const int size){
        // full C code
    }
    @Slave(const int num_threads, void *buffer, const int size, const policy){
        // full C code
    }
    // full C code

```
**Domain-Specific Language for Pattern-Oriented Parallel Programming (DSL-POPP)**

**DSL-POPP System**

- DSL-POPP library (*poppLinux.h*)
- POPP model and C code is automatically done by the pre-compiler
- The pre-compiler generates the C parallel code using the Pthreads library
- The GNU C compiler generates binary code for the target platform

**Figure:** Compiler overview.
Matrix Multiplication (MM) with POSIX Pthreads

1#include <stdio.h>
2#include <pthread.h>
3#define MX 1000 // matrix size
4long int num_threads=2;
5long int matrix1[MX][MX], matrix2[MX][MX], matrix[MX][MX];
6double timer() {/* return the time in seconds */}
7void attr_val(long int **matrix, long int **matrix1, long int **matrix2) {/* values attribution */}
8void printMatrix(long int **matrix){/* prints a matrix */}
9void *Thread(void *th_id){
10long int id= (long int*) th_id;
11long int i, j, k, end;
12end=(id*(MX/num_threads))+(MX/num_threads);
13if(id==num_threads-1)
14end=MX;
15for(i=id*(MX/num_threads); i<end; i++)
16for(j=0; j<MX; j++)
17for(k=0; k<MX; k++)
18matrix[i][j] += (matrix1[i][k] * matrix2[k][j]);
19pthread_exit(NULL);
20}
21int main(){
22double t_start, t_end;
23pthread_t th[num_threads];
24void *status;
25t_start = timer();
26attr_val(matrix,matrix1,matrix2);
27int i;
28for(i=0; i<num_threads; i++)
29pthread_create(&th[i], NULL, &Thread, (void *) i);
30for(i=0; i<num_threads; i++)
31pthread_join(&th[i], &status);
32t_end = timer();
33printf("EXECUTION TIME: %.2f seconds\n", t_end-t_start);
34printMatrix(matrix);
35}
Matrix Multiplication (MM) with DSL-POPP

```c
#include <stdio.h>
#include "poppLinux.h"
#define MX 1000 // matrix size
long int num_threads=2;
long int matrix1[MX][MX], matrix2[MX][MX], matrix[MX][MX];
double timer(){/* return the time in seconds*/}
void attr_val(long int **matrix, long int **matrix1, long int **matrix2){/* values attribution*/}
void printMatrix(long int **matrix){/* prints a matrix*/}

$MasterSlavePattern int main(){
  @Master(matrix, MX){
    double t_start, t_end;
    t_start = timer();
    attr_val(matrix, matrix1, matrix2);
    @Slave(num_threads, matrix, MX, POPP_STATIC){
      long int i, j, k;
      for(i=0; i<MX; i++)
        for(j=0; j<MX; j++)
          for(k=0; k<MX; k++)
            matrix[i][j] += (matrix1[i][k]*matrix2[k][j]);
    }
    t_end = timer();
    printf("EXECUTION TIME: %lf seconds\n", t_end-t_start);
    printMatrix(matrix);
  }
}  
```

Usability Evaluation

Environment

- **Metric**: time spent to implement in parallel the MM algorithm
- **Samples**: Ph.D. students in computer science
- **Statistic**: 95% of reliability using SPSS tool
- **Scenario**: controlled (a workstation, access to manual, and sequential algorithm)

Hypotheses

- **Null hypothesis** \((H_0)\): Pthreads = DSL-POPP.
- **Alternative hypothesis** \((H_1)\): Pthreads > DSL-POPP.

Experiment Conduction

- Evaluate the previously knowledge to select the students (subjective)
- Split the 20 students in two groups with balanced knowledge
- All student have to finish their experiment achieving at least 90% of speed-up
Usability Evaluation

- Pthreads mean: **51.45** (confidence interval is 42.98 – 59.92 minutes)
- DSL-POPP mean: **20.70** (confidence interval is 17.59 – 23.81 minutes)

**Table:** Effort evaluation results.

<table>
<thead>
<tr>
<th>ID</th>
<th>Experience with Pthreads</th>
<th>DSL-POPP Time (min)</th>
<th>Pthreads Time (min)</th>
<th>ID</th>
<th>Experience with Pthreads</th>
<th>DSL-POPP Time (min)</th>
<th>Pthreads Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Medium</td>
<td>29</td>
<td>18</td>
<td>11</td>
<td>High</td>
<td>15</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
<td>23</td>
<td>32</td>
<td>12</td>
<td>Medium</td>
<td>17</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>Low</td>
<td>13</td>
<td>30</td>
<td>13</td>
<td>Medium</td>
<td>12</td>
<td>65</td>
</tr>
<tr>
<td>4</td>
<td>Medium</td>
<td>25</td>
<td>29</td>
<td>14</td>
<td>Low</td>
<td>15</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>Medium</td>
<td>19</td>
<td>27</td>
<td>15</td>
<td>Low</td>
<td>17</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>Low</td>
<td>33</td>
<td>65</td>
<td>16</td>
<td>Low</td>
<td>15</td>
<td>71</td>
</tr>
<tr>
<td>7</td>
<td>Low</td>
<td>15</td>
<td>39</td>
<td>17</td>
<td>Low</td>
<td>12</td>
<td>61</td>
</tr>
<tr>
<td>8</td>
<td>Low</td>
<td>31</td>
<td>81</td>
<td>18</td>
<td>Zero</td>
<td>21</td>
<td>63</td>
</tr>
<tr>
<td>9</td>
<td>Zero</td>
<td>28</td>
<td>59</td>
<td>19</td>
<td>Zero</td>
<td>24</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>Low</td>
<td>28</td>
<td>45</td>
<td>20</td>
<td>Low</td>
<td>22</td>
<td>66</td>
</tr>
</tbody>
</table>
Usability Evaluation

- Pthreads mean: **51.45** (confidence interval is 42.98 – 59.92 minutes)
- DSL-POPP mean: **20.70** (confidence interval is 17.59 – 23.81 minutes)

**Table:** Statistical hypothesis test.

<table>
<thead>
<tr>
<th>Ranks</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative Ranks</td>
<td>1</td>
<td>4.0</td>
<td>4.0</td>
</tr>
<tr>
<td>Positive Ranks</td>
<td>19</td>
<td>10.8</td>
<td>206.0</td>
</tr>
<tr>
<td>Ties</td>
<td>0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>20</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

- (a) Pthreads < DSL-POPP
- (b) Pthreads > DSL-POPP
- (c) Pthreads = DSL-POPP

**Wilcoxon test**

<table>
<thead>
<tr>
<th>Pthreads vs. DSL-POPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>Asymp. Sig. (2-tailed)</td>
</tr>
</tbody>
</table>

*Based on negative ranks.
## Performance Experiments

### Environment
- **Metric**: execution time to calculate efficiency, speed-up and compare the differences of performance
- **Hardware**: a machine with an Intel Xeon X3470 (2.93GHz), 8GB of main memory and 2TB of disk, running Ubuntu-Linux-12.04-server-64bits.

### Hypotheses
- **Null hypothesis** ($H_0$): Pthreads $=$ DSL-POPP.
- **Alternative hypothesis** ($H_1$): Pthreads $!=$ DSL-POPP.

### Experiment Conduction
- A voluntary implementation of four applications with DSL-POPP and Pthreads
- We certified that the output result was the same of the sequential ones
- 40 random executions were performed for each sample
Performance Experiments

Applications
- Estimates an Integral (EI)
- Molecular Dynamic (MD)
- Matrix Multiplication (MM)
- Primers Number (PN)

Table: Code analysis.

<table>
<thead>
<tr>
<th>App.</th>
<th>Source Lines of Code (SLOC)</th>
<th>Development Effort Estimate (Min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Orion</td>
<td>DSL-POPP</td>
</tr>
<tr>
<td>EI</td>
<td>91</td>
<td>93</td>
</tr>
<tr>
<td>MD</td>
<td>249</td>
<td>259</td>
</tr>
<tr>
<td>MM</td>
<td>99</td>
<td>105</td>
</tr>
<tr>
<td>PN</td>
<td>78</td>
<td>100</td>
</tr>
</tbody>
</table>
Performance Experiments

![Graphs showing performance experiments for EI, MD, MM, and PN with speedup and efficiency variations with the number of threads.](image)
## Performance Experiments

**Table:** SPSS output for the test of significance (Sig.)

<table>
<thead>
<tr>
<th>Threads</th>
<th>EI</th>
<th>MD</th>
<th>MM</th>
<th>PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.455</td>
<td>0.135</td>
<td>0.059</td>
<td>0.281</td>
</tr>
<tr>
<td>4</td>
<td>0.740</td>
<td>0.090</td>
<td>0.174</td>
<td>0.001</td>
</tr>
<tr>
<td>6</td>
<td>0.000</td>
<td>0.000</td>
<td>0.837</td>
<td>0.199</td>
</tr>
<tr>
<td>8</td>
<td>0.000</td>
<td>0.269</td>
<td>0.381</td>
<td>0.011</td>
</tr>
</tbody>
</table>
Conclusions

Overview of Results

- Performance and usability evaluation using software experiments and statistics analysis
- DSL-POPP requires less effort than Pthreads
- The performance is not significantly affected
- The SLOCCount tool also shows that DSL-POPP requires less effort than Pthreads

Future Works

- Evaluate programming effort using nesting pattern feature
- Repeat this experiments using other patterns available in DSL-POPP
- Compare performance and usability with OpenMP and Cilk++
References I


Laxmikant V. Kale and Sanjeev Krishnan.
CHARM++: A Portable Concurrent Object Oriented System Based on C++.

Charles E. Leiserson.
The Cilk++ Concurrency Platform.

Dalvan Griebler and Luiz G. Fernandes.
Towards a Domain-Specific Language for Patterns-Oriented Parallel Programming.
Performance and Usability Evaluation of a Pattern-Oriented Parallel Programming Interface for Multi-Core Architectures

Authors: Dalvan Griebler, Daniel Adornes, Luiz Gustavo Fernandes

E-mail: dalvan.griebler@acad.pucrs.br

Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Programa de Pós-Graduação em Ciência da Computação - PPGCC
Grupo de Modelagem de Aplicações Paralelas - GMAP

Software Engineering and Knowledge Engineering - SEKE 2014

July 2014