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

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]

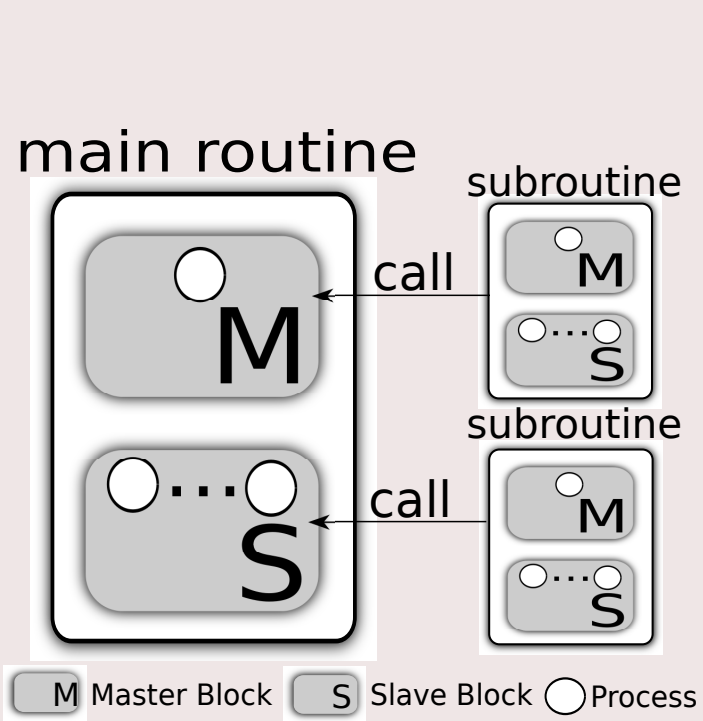
Domain-Specific Language for Pattern-Oriented Parallel Programming (DSL-POPP)

POPP model [9]

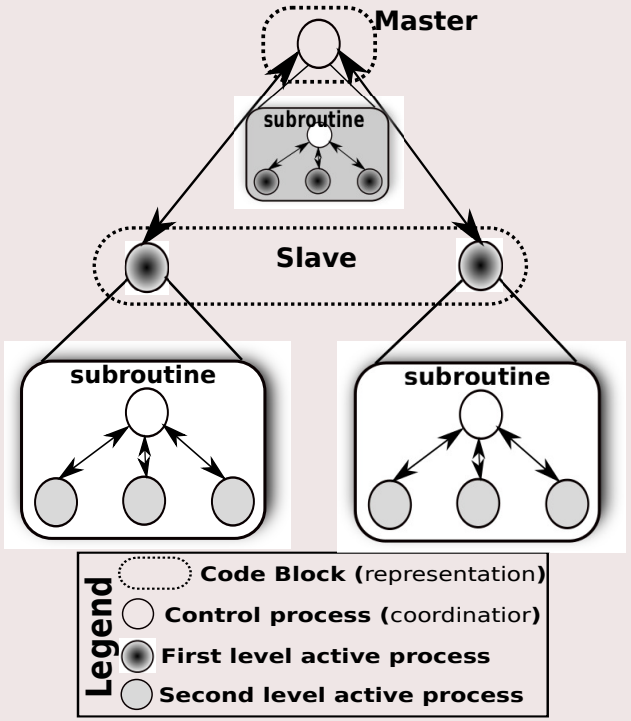
- **It is**: a standard way to implement parallel patterns in routines that uses code blocks to design parallel applications
- **The goal**: is that developers explicitly implement parallel programs oriented by a set of patterns pre-implemented in a high-level programming interface
- **Features**: define 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

```
1 $MasterSlavePattern int func_name(){
2   @Master(void *buffer, const int size){
3     // full C code
4     @Slave(const int num_threads, void *buffer,const int size,
5     const policy){
6       // full C code
7     }
8     // full C code
9   }
10 }
```

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

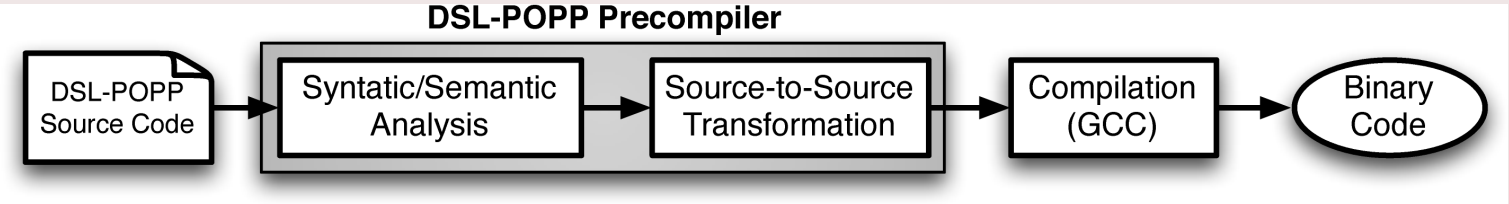


Figure: Compiler overview.

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){
10  long int id= (long int*) th_id;
11  long int i, j, k, end;
12  end=(id*(MX/num_threads))+(MX/num_threads);
13  if(id==num_threads-1)
14      end=MX;
15  for(i=id*(MX/num_threads); i<end; i++)
16    for(j=0; j<MX; j++)
17      for(k=0; k<MX; k++)
18          matrix[i][j] += (matrix1[i][k] * matrix2[k][j]);
19   pthread_exit(NULL);
20}
21int main(){
22  double t_start, t_end;
23  pthread_t th[num_threads];
24  void *status;
25  t_start = timer();
26  attr_val(matrix,matrix1,matrix2);
27  int i;
28  for(i=0;i<num_threads; i++)
29      pthread_create(&th[i], NULL, &Thread, (void *) i);
30  for(i=0; i<num_threads; i++)
31      pthread_join(&th[i], &status);
32  t_end = timer();
33  printf("EXECUTION TIME: %lf seconds\n", t_end-t_start);
34  printMatrix(matrix);
35}
```

Matrix Multiplication (MM) with DSL-POPP

```
 1#include <stdio.h>
 2#include "poppLinux.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 a
 8void printMatrix(long int **matrix){/*prints a matrix*/}
 9$MasterSlavePattern int main(){
10  @Master(matrix, MX){
11    double t_start, t_end;
12    t_start = timer();
13    attr_val(matrix,matrix1,matrix2);
14    @Slave(num_threads, matrix, MX, POPP_STATIC){
15      long int i, j, k;
16      for(i=0; i<MX; i++)
17        for(j=0; j<MX; j++)
18          for(k=0; k<MX; k++)
19            matrix[i][j] += (matrix1[i][k]*matrix2[k][j]);
20    }
21    t_end = timer();
22    printf("EXECUTION TIME: %lf seconds\n",t_end-t_start);
23    printMatrix(matrix);
24  }
25}
```

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.

	Group 1: DSL-POPP → Pthreads				**Group 2: Pthreads → DSL-POPP**		
ID	Experience with Pthreads	DSL-POPP Time (min)	Pthreads Time (min)	ID	Experience with Pthreads	DSL-POPP Time (min)	Pthreads Time (min)
1	**Medium**	**29**	**18**	11	High	15	33
2	Medium	23	32	12	Medium	17	54
3	Low	13	30	13	Medium	12	65
4	Medium	25	29	14	Low	15	70
5	Medium	19	27	15	Low	17	60
6	Low	33	65	16	Low	15	71
7	Low	15	39	17	Low	12	61
8	Low	31	81	18	Zero	21	63
9	Zero	28	59	19	Zero	24	61
10	Low	28	45	20	Low	22	66

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.

Ranks	N	Mean Rank	Sum of Ranks
Negative Ranks	1 *(a)*	4.0	4.0
Positive Ranks	19 *(b)*	10.8	206.0
Ties	0 *(c)*	–	–
Total	**20**		

(a) Pthreads < DSL_POPP
(b) Pthreads > DSL_POPP
(c) Pthreads = DSL_POPP

Wilcoxon test	Pthreads vs. DSL-POPP
Z	-3.771*
Asymp. Sig. (2-tailed)	0.0

**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 a 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

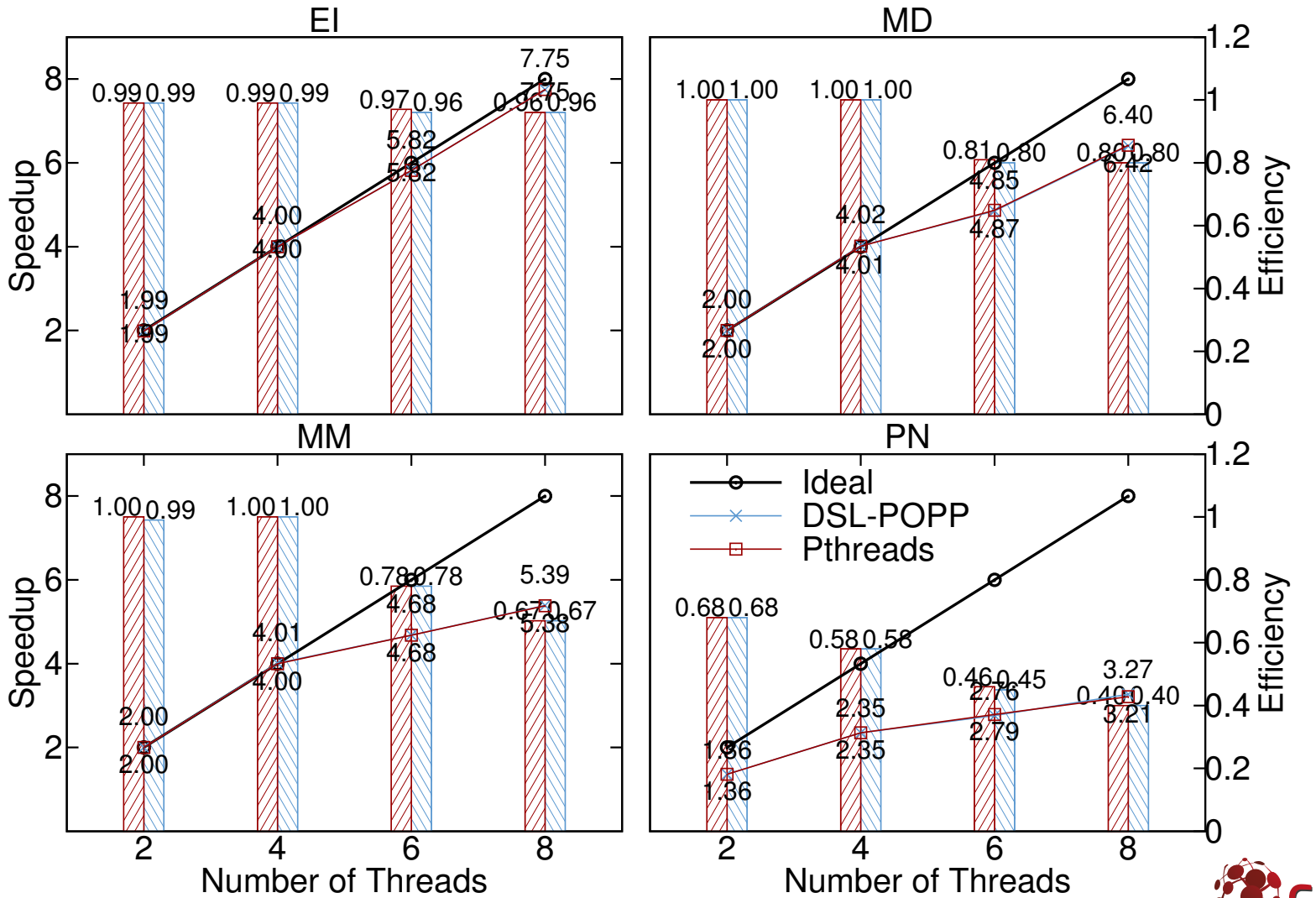
Applications

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

Table: Code analysis.

	Source Lines of Code (SLOC)			*Development Effort Estimate (Min.)*		
App.	**Ori.**	**DSL-POPP**	**Pthreads**	**Ori.**	**DSL-POPP**	**Pthreads**
EI	91	93	135	8322	8760	12702
MD	249	259	352	24528	25404	35040
MM	99	105	209	9198	10074	20148
PN	78	100	142	7008	9198	13578

Performance Experiments



Performance Experiments

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

Threads	EI	MD	MM	PN
2	0.455	0.135	0.059	0.281
4	0.740	0.090	0.174	**0.001**
6	**0.000**	**0.000**	0.837	0.199
8	**0.000**	0.269	0.381	**0.011**

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

Hassan Chafi, Arvind Sujeeth, Kevin Brown, HyoukJoong Lee, Anand Atreya, and Kunle Olukotun.
A Domain-specific Approach to Heterogeneous Parallelism.
In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, volume 1, pages 35–46, New York, NY, USA, February 2011. ACM.

Murray Cole.
Algorithmic Skeletons: Structured Management of Parallel Computation.
MIT Press, Cambridge, USA, 1989.

Steve Bromling, Steve MacDonald, John Anvik, Jonathan Schaeffer, Duane Szafron, and Kai Tan.
Pattern-Based Parallel Programming.
In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 257–265, British Columbia, Canada, 2002. IEEE Computer Society.

Intel.
Thread Building Block (Intel® TBB), Extracted from <https://www.threadingbuildingblocks.org>, 2014.

Marco Aldinucci, Marco Danelutto, Peter Kilpatrick, Massimiliano Meneghin, and Massimo Torquati.
Accelerating Code on Multi-cores with FastFlow.
In *Euro-Par 2011 Parallel Processing*, volume 6853, pages 170–181, Berlin, Heidelberg, August 2011. Springer-Verlag.

Barbara Chapman, Gabriele Jost, and Ruud Pas.
Using OpenMP: Portable Shared Memory Parallel Programming.
Massachusetts Institute of Technology, London, England, 2008.

References II

Laxmikant V. Kale and Sanjeev Krishnan.

CHARM++: A Portable Concurrent Object Oriented System Based on C++.
In *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*, pages 91–108, New York, NY, USA, October 1993. ACM.

Charles E. Leiserson.

The Cilk++ Concurrency Platform.
In *Proceedings of the 46th Annual Design Automation Conference*, volume 1, pages 522–527, New York, NY, USA, July 2009. ACM.

Dalvan Griebler and Luiz G. Fernandes.

Towards a Domain-Specific Language for Patterns-Oriented Parallel Programming.
In *Programming Languages - 17th Brazilian Symposium - SBLP*, volume 8129 of *Lecture Notes in Computer Science*, pages 105–119, Brasilia, Brazil, October 2013. Springer Berlin Heidelberg.

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