Profile Driven Application Parallelization

Imran Ashraf, Nader Khammassi, Koen Bertels
Computer Engineering Lab, TU Delft, The Netherlands
Email: {I.Ashraf, N.Khammassi, K.L.M.Bertels}@TUDelft.nl

***Abstract*—The growing demand of processing power is being satisfied mainly by various forms of parallelism in the computing system. Efficient application parallelization to make use of the available parallelism in the architecture, is still an open issue. In this paper, we present a parallelizing framework based mainly on two complementary tools, MCROF and XPU to provide an alternative development path to parallelise applications. This framework addresses the challenges of identifying potential parallelism and exploiting it in a different way. The MCROF tool provides a detailed application profile helping in extracting parallelism in a sequential application and the XPU programming paradigm provides an intuitive and simple interface to express parallelism as well as the necessary runtime support. The framework takes a sequential application, generates the required static and dynamic information and utilizes this infomration to automatically generate the parallel representation of the application. We demonstrate through a use case that our framework is able to extract various forms of fine and coarse grained parallelism.**

***Keywords*—Memory profiling, dependence analysis, program parallelization.**

I. Introduction

The number of transistors per chip is growing due to technology scaling and increasing the clock rate of processors is becoming technologically less viable [1]. The current trend is therefore to integrate a growing number of processing cores on chip, forcing parallelizing compilers to mature rapidly and to provide efficient code for the multi-core processors.

Despite decades of research and development of parallelizing compilers, automatic parallelization of sequential code using a compiler is standing as the holy grail of parallel computing and has had a limited success. Utilization of tool-chains which assist the programmer by full or partial automation of various parallelization phases presents a better alternative which can offer better performance-productivity trade off.

In this work, we present a framework based on MCROF and XPU tools to help programmers extract and express parallelization. The MCROF tool provides a detailed profile of the data flowing inside an application and the XPU programming paradigm provides an intuitive and simple interface to express parallelism as well as the necessary runtime support. The framework automates the process of extracting various forms of parallelism in sequential applications. The framework takes a sequential application, generates the required static and dynamic information and utilizes this information to automatically generate the parallel representation of the application. We demonstrate the working of framework by a case study to highlight the information generated by various tools involved in the developed framework. We also discuss that our framework is able to extract various forms of fine and coarse grained parallelism.

Rest of the paper is organized as follows. Section II discusses some of the available parallelizing compiler frameworks. Section III introduces the parallelizing framework followed by Section IV describing the use of the proposed framework. Experimental results are provided in Section V. Finally Section VI concludes this work.

II. Related Work

Though static-analysis tools [2] can also track data-communication, a large number of tools [3], [4] utilize dynamic-analysis to collect producer-consumer relationship at runtime. These tools have high run-time overhead, which limits their use for realistic workloads affecting the quality of the generated information. Furthermore, the provided information lacks necessary dynamic details and is not linked to the source code, making it hard to utilize this information. A number of automatic parallelization compilers exist such as Par4All [5], Cetus [6], Parallware [7], Polaris [8] or PolyCC/PLUTO [9] which are source to source compilers that can produce parallel code after analyzing the sequential code using different parallelization techniques. The Intel ICC [10] is a popular compiler which provides an automatic parallelization feature which allows both instruction-level and thread-level parallelization of sequential regions of the input code.

Automatic parallelization of sequential code has had limited success [11]. Great advances have been made in automatic parallelism extraction at the instruction level, however, in order to exploit efficiently modern multicore platforms, compilers need to capture parallelism also at thread-level which is a challenging task. Pareon by Vector Fabrics [12] is an example of a tool, which assists the programmer and guides the parallelization process instead of performing automatic code parallelization.

III. MCPROF-XPU Parallelizing Framework

This section briefly presents the MCROF-XPU parallelizing framework. Figure 1 presents an overview of the framework.

A. *ROSE Compiler*

ROSE [13] is an open-source, object-oriented compiler infrastructure facilitating ease of building tools for analysis and transformation of programs in various languages including C/C++. We have used ROSE for the following tasks.

1) Inserting markers in the source-code to mark loop boundries.

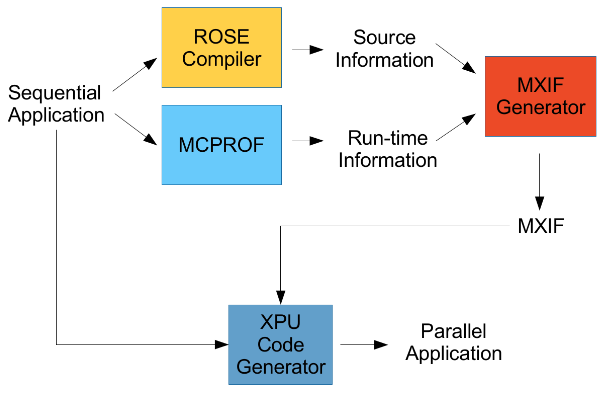

Fig. 1. MCROF-XPU Parallelizing Framework.

2) Outlining loops to consider them as separate functions.
3) Generating source location information like starting and ending position of loops, variable names etc.

B. MCPROF

MCROF [14] is a runtime memory and communication profiler which generates detailed application profile in terms of memory access based on Intel Pin Dynamic Binary Instrumentation (DBI) framework [15]. MCROF performs instruction-level instrumentation to track memory reads and writes by each instruction. Furthermore, routine-level instrumentation is utilized to keep track of the currently executing function. These are tracked by maintaining a call-stack of the functions executing in the application. To track the dynamic allocations in the application, image-level instrumentation is utilized to selectively instrument library images for memory (re)allocation/free routines.

The tracked memory reads/writes are associated with parts of the source code depending upon the selected granularity i.e. functions, loops or other marked regions in the source code. In this way, a producer-consumer relationship is established between functions/loop/objects in the source code and reported in the form of a communication graph. In this work we extended MCROF to generate the following information:

- Runtime callgraph to show the loop nests as well in the callgraph. This is important for function splitting at the coarser granularity of loop nests.
- Coarse-grained dependence information, for instance, dependence between functions and loop nests.
- Fine-grained dependence information, for instance, dependence among loop iterations.

C. MXIF Generator

This block takes static and dynamic information generated by various tools to generate parallel application. In order to help programmers easily apply the parallelization manually, or help a tool automatically generate the parallel application, we generate a representation of the parallel application in MCROF-XPU Interchage Format (MXIF).

D. XPU Code Generator

This block takes sequential application and the representation of parallel application in MXIF to generate parallel application with XPU code. The generated parallel code can then be compiled with ordinary compiler, for instance *g++* to generate the executable.

XPU [16] is a structured parallel programming framework which aims to easily express and exploit parallelism. XPU allows the expression of different types of parallelism at different levels of granularity. Supported parallelism types includes data parallelism [17], task parallelism [18] and pipeline parallelism [19]. These different parallelism types can be composed hierarchically in the same application [20].

XPU utilizes C++ meta-programming techniques [21][22] exploiting the potential of the standard C++ language and thus does not require any particular tool except standard C++ compiler. XPU provides a friendly and light weight programming interface which enables the programmer to design parallel applications or parallelize sequential applications with minimal code changes without any significant alteration.

IV. Case-study

```
 1  void addsub(float* sum, float* diff, float* in1, float* in2
      , int N)
 2  {
 3      for (int i=0; i<N; i++)
 4          sum[i] = in1[i] + in2[i];
 5
 6      for (i=0; i<N; i++)
 7          diff[i] = in1[i] - in2[i];
 8  }
 9  void muldiv(float* prod, float* qout, float* in1, float*
      in2, int N)
10  {
11      for (int i=0; i<N; i++)
12          prod[i] = in1[i] * in2[i];
13
14      for (int i=0; i<N; i++)
15          qout[i] = in1[i] / in2[i];
16  }
17  int main(int argc, char **argv)
18  {
19      int i, N = 5000;
20      /* allocation of a,b,c,d,e and f */
21
22      for (i=0; i<N; i++){
23          e[i] = i+1.7;
24          f[i] = i+1.7;
25      }
26      addsub(a,b,e,f,N);
27      muldiv(c,d,e,f,N);
28
29      /* de-allocation of a,b,c,d,e and f */
30  }
```

Listing 1. Example Sequential Application.

The focus of this case-study is to show various intermediate steps taken by the parallelizing framework to parallelize a sequential application presented in Listing (1) and show the information generated by various tools in the framework. Though this is a simple application, it contains various forms of parallelisms:

- **Coarse-grained:** Function calls at Lines 26-27 can be executed in parallel.
- **Coarse-grained:** Both loops in function *addsub* can be executed in parallel to each other.

```
sequential main_0_1
    parallel_for main_15_2_pf
        task main_15_2
    parallel dummy_par_3
        parallel addsub_23_3
            parallel_for addsub_7_4_pf
                task addsub_7_4
            parallel_for addsub_9_5_pf
                task addsub_9_5
        parallel muldiv_24_6
            parallel_for muldiv_11_7_pf
                task muldiv_11_7
        parallel_for muldiv_13_8_pf
            task muldiv_13_8
```

Fig. 2. Simplified representation of parallel application.

- **Coarse-grained:** Both loops in function *muldiv* can be executed in parallel to each other.
- **Fine-grained:** Iterations of all the *for* loops can be executed in parallel.

```
( task_graph dummy_par_3
   ( parallel
      ( task_graph addsub_23_3)
      ( task_graph muldiv_24_6)
   )
)
( task_graph addsub_23_3
   ( parallel
      ( task_graph addsub_7_4_pf)
      ( task_graph addsub_9_5_pf)
   )
)
( task_graph addsub_7_4_pf
   ( parallel_for
      ( task addsub_7_4)
   )
)
( task addsub_7_4
   ( mapping_of
      ( chunk_of addsub_7
         ( function addsub
            ( file "test.c" )
            ( lines 5 22 )
         )
         ( lines 9 13 )
         ( input sum " float * " )
         ( input in1 " float * " )
         ( input in2 " float * " )
         ( input N " int " )
         ( input i " int " )
         ( output NULL )
      )
   )
   ( call_file "test.c" )
   ( call_line 9 )
)
```

Listing 2. Part of the Generated MXIF.

Figure 2 shows the result of parallelization in simplified form. It can be seen from this figure that the framework is able to extract the available parallelism in the application. Listing (2) shows the generated MXIF.

V. Experimental Results

In order to check the scalability of the parallel application, we have executed it on two machines. Machine 1 is has 40 core (20 core 2 way hyper threaded) on Intel(R) Xeon(R) CPU E5-2670 v2 running at 2.50 GHz having 96 GB of main memory. Machine 2 has 64 core (16 core 4 way hyper threaded) IBM Power 7 v2.3 running at 3.3 GHz containing 164 GB main memory. Figure 3 and Figure 4 depict the speedup results for both the machines showing the scalability upto 64 and 128 threads respectively.

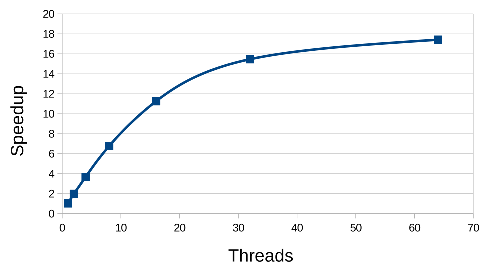

Fig. 3. Speedup Results on Machine 1.

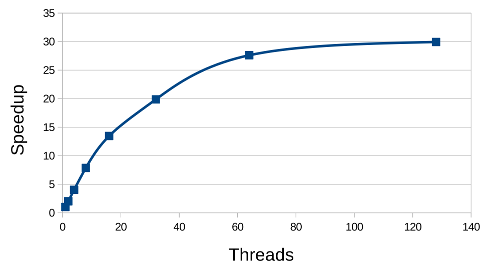

Fig. 4. Speedup Results on Machine 2.

VI. Conclusions

With the emergence of multicore processor architectures, we can no longer avoid parallelizing applications but parallelizing compilers have not been very successful in this regard. In this paper, we have presented the integrated use of two tools which are very complementary in their functionality. MCROF provides a detailed profile which is combined with the source level information to automatically generate parallel representation of the application. XPU is then used for parallelism expression as it provides minimally invasive code changes to express the available parallelism in an application. Not only does the combined approach extracts the available parallelism, it also reduces substantially the overall time needed to parallelize sequential applications. Future work includes the completion of code-generation part and detailed testing of the framework with a wide variety of applications.

Acknowledgment

This research is supported by Artemis EMC2 Project (grant 621429), Artemis Almarvi Project (grant 621439) and Artemis CRAFTERS Project (grant 295371).

References

[1] M. Horowitz and W. Dally, “How Scaling Will change Processor Architecture,” in *ISSCC*, vol. 1, 2004, pp. 132–133.

[2] M. D. Ernst, “Static and Dynamic Analysis: Synergy and Duality,” in *WODA 2003: Workshop on Dynamic Analysis*, Portland, Oregon, May 9, 2003, pp. 24–27.

[3] W. Heirman, D. Stroobandt, N. R. Miniskar, and R. Wuyts, “A communication profiler to optimize embedded resource usage,” *Annual Workshop on Circuits, Systems and Signal Processing, 20th, Proceedings*, pp. HASH(0x57aafb8)–HASH(0x57aafb8), 2009. [Online]. Available: https://biblio.ugent.be/publication/820031

[4] S. Ostadzadeh, “Quantitative Application Data Flow Characterization for Heterogeneous Multicore Architectures,” Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, December 2012.

[5] “Par4All: An Automatic Parallelizing and Optimizing Compiler by HPC Project,” http://www.par4all.org.

[6] C. Dave, H. Bae, S. Min, S. Lee, R. Eigenmann, and S. Midkiff, “Cetus: A Source-to-Source Compiler Infrastructure for Multicores,” *Computer*, vol. 42, no. 12, pp. 36–42, Dec. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.385

[7] Appentra, “Parallware,” http://www.appentra.com/products/parallware.

[8] W. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. A. Padua, P. Petersen, W. M. Pottenger, L. Rauchwerger, P. Tu, and S. Weatherford, “Polaris: Improving the Effectiveness of Parallelizing Compilers,” in *Proceedings of the 7th International Workshop on Languages and Compilers for Parallel Computing*, ser. LCPC ’94. London, UK, UK: Springer-Verlag, 1995, pp. 141–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=645672.665547

[9] U. Bondhugula, J. Ramanujam, and P. Sadayappan, “PLuTo: A Practical and Fully Automatic Polyhedral Parallelizer and Locality Optimizer,” The Ohio State University, Tech. Rep., Oct 2007.

[10] Intel, “Automatic Parallelization with Intel Compilers,” https://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers.

[11] J. Shen and M. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill Higher Education, 2002. [Online]. Available: http://books.google.fr/books?id=VIWLAAAACAAJ

[12] “Pareon by Vector Fabrics B.V,” http://www.vectorfabrics.com/products.

[13] “ROSE Compiler Infrastructure,” http://www.rosecompiler.org.

[14] I. Ashraf, V. Sima, and K. Bertels, “Intra-Application Data-Communication Characterization,” in *Proc. 1st International Workshop on Communication Architectures at Extreme Scale*, Frankfurt, Germany, July 2015.

[15] C. Luk and et al., “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation,” in *PLDI ’05*. New York, NY, USA: ACM, 2005, pp. 190–200.

[16] N. Khammassi, “High-level Structured Programming Models for Explicit and Automatic Parallelization on Multicore Architectures,” Theses, Université de Bretagne Sud, Dec. 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01207434

[17] N. Khammassi et al., “Design and Implementation of a Cache Hierarchy-aware Task Scheduling for Parallel Loops on Multicore Architectures,” in *PDCTA*, Sydney, Australia, 2014.

[18] N. Khammassi, J. Le Lann, J. Diguet, and A. Skrzyniarz, “MHPM: Multi-Scale Hybrid Programming Model: A Flexible Parallelization Methodology,” in *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication*, ser. HPCC ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 71–80. [Online]. Available: http://dx.doi.org/10.1109/HPCC.2012.20

[19] N. Khammassi and J.-C. Le Lann, “A High-level Programming Model to Ease Pipeline Parallelism Expression on Shared Memory Multicore Architectures,” in *Proceedings of the High Performance Computing Symposium*, ser. HPC ’14. San Diego, CA, USA: Society for Computer Simulation International, 2014, pp. 9:1–9:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2663510.2663519

[20] N. Khammassi and J. Le Lann, “Tackling Real-Time Signal Processing Applications on Shared Memory Multicore Architectures Using XPU,” Feb 2014, conference proceeding. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00958087

[21] J. Koskinen, “Metaprogramming in C++.” [Online]. Available: www.cs.tut.fi/~kk/webstuff/MetaprogrammingCpp.pdf

[22] H. Singh, “Introspective C++,” Ph.D. dissertation, Virginia Polytechnic Institute, Virginia, VA, USA, 2004.