Author Retrospective for PYRROS: Static Task Scheduling and Code Generation for Message Passing Multiprocessors

Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
tyang@cs.ucsb.edu

Apostolos Gerasoulis
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903, USA
gerasoul@cs.rutgers.edu

ABSTRACT

Given a program with annotated task parallelism represented as a directed acyclic graph (DAG), the PYRROS project was focused on fast DAG scheduling, code generation and runtime execution on distributed memory architectures. PYRROS scheduling goes through several processing stages including clustering of tasks, cluster mapping, and task execution ordering. Since the publication of the PYRROS project, there have been new advancements in the area of DAG scheduling algorithms, the use of DAG scheduling for irregular and large-scale computation, and software system development with annotated task parallelism on modern parallel and cloud architectures. This retrospective describes our experience from this project and the follow-up work, and reviews representative papers related to DAG scheduling published in the last decade.

Original paper: http://dx.doi.org/10.1145/143369.143446

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); D.1.3 [**Programming Techniques**]: Parallel Programming; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

Keywords

DAG, parallel processing, scheduling, task graphs

1. INTRODUCTION

Directed-acyclic task graphs (DAGs) have been widely used for parallelism representation, performance modeling, code generation, and execution optimization of parallel programs. Earlier work in parallelizing compilers and runtime systems [21, 28, 6] adopted DAG-based task parallelism. The contribution of the PYRROS project was to develop fast scheduling algorithms for parallel code generation and resource management in dealing with a large task graph and taking communication cost into consideration. Experience from PYRROS demonstrates that scheduling and code generation systems can be built with low complexity to simplify parallel programming of complex parallel architectures, especially for coarse grain computation with many task components or irregular patterns.

During the last decade, the development in high performance computing infrastructures continues to present a formidable challenge for parallel programming. It is difficult for application users to fully harness the power of multi-core clusters or hybrid platforms with GPU accelerators or other co-processors. DAG-based codes have been suggested as a suitable candidate for exascale applications [1, 14] to exploit modern parallel and distributed architectures. New algorithms and techniques were developed for DAG-based computation execution on emerging architectures. We will give a short survey of those techniques as a retrospective on DAG scheduling in high performance computing.

1.1 The PYRROS system and follow-up work

PYRROS contains a task graph language with an interface to C and Fortran, allowing users to define partitioned programs and data; a scheduling system; a graphic displayer for visualizing task graphs and scheduling results; a code generator that inserts synchronization primitives and performs code optimization for distributed memory machines. PYRROS uses the following multistep approach to fast scheduling: 1) Perform clustering using the Dominant Sequence Algorithm (DSC) [33]. 2) Merge and map the clusters to available physical machines. 3) Order the execution of tasks in each processor using the RCP algorithm [32]. The overall algorithm complexity is $O(e + vlogv)$ for a DAG with v nodes and e edges, and the near-linear complexity allows the system to handle a large task graph efficiently.

The experience learned from the PYRROS system is that an automatic system for scheduling and code generation is useful in many ways. If the scheduling is determined at compile time then the complex parallel architecture can be utilized better. A programmer does not have to get involved in low level programming and synchronization, and the parallel code generated or managed by such a system can provide better portability. The scheduling system can also facilitate performance prediction and comparison of different optimization strategies without actual code execution.

As a follow-up work to leverage DAG-scheduling for unstructured computation [10], we studied the use of PYRROS

ICS 25th Anniversary Volume. 2014
ACM 978-1-4503-2840-1/14/06.
http://dx.doi.org/10.1145/2591635.2591647.

scheduling for irregular code [20] and developed a software system called RAPID based on an inspector/executor approach for runtime parallelization [18, 17, 34]. RAPID provides library functions for specifying irregular data objects and tasks that access these objects, extracts a task dependence graph from data access patterns, and executes tasks efficiently on distributed memory machines. Additional research issues addressed were space/time-efficient DAG scheduling optimization for better data locality and load balancing, the use of advanced hardware features for low-overhead asynchronous communication, efficient task-level synchronization protocols for maintaining data consistency, and runtime memory management for maximizing space reuse. The effectiveness of RAPID was demonstrated in parallelizing irregular problems with static or slowly changing dynamic computation patterns such as sparse matrix solving and the adaptive fast multipole method for n-body simulation. Using RAPID, we developed fast DAG-based parallel code for sparse LU factorization with partial pivoting and set up a new performance record [16, 15].

2. PROGRESS IN DAG-BASED PARALLELIZATION AND SCHEDULING

There has been a large body of active research on DAG-based parallelization and computation scheduling. We summarize a number of representative papers published during the last decade in the following three areas.

Algorithms and compilation support for DAGs. An earlier survey paper on DAG scheduling was by Kwok and Ahmad [24]. Cosnard et al. [11, 12] studied the compact representation and symbolic scheduling of DAGs to avoid the unrolling of dependence structure. Sinnen et al. [29] presented scheduling with task duplication under communication contention. Bozdağ et al. [8] optimized the number of processors required in duplication-based scheduling. Vydyanathan et al. [31] presented DAG scheduling with task duplication in processing a stream of input data. Daoud and Kharma [13] considered the architecture heterogeneity in two-phase task scheduling.

Adve and Sakellariou [2] described parallelizing compiler techniques to automate the task graph constructions for the MPI code generated for a High-Performance-Fortran program. Baskaran et al. [4] discussed compile-time parallelization techniques in the Pluto system to enable dynamic extraction of inter-tile dependences at run-time, and dynamic scheduling of the parallel tiles on multi-core architectures. Bellens et al. [5] presented a programming model called CellSs with annotated functional parallelism. A source-to-source compiler generates the necessary code that builds a runtime task dependency graph and supports a locality-aware task scheduling.

Parallel programming and application experience with DAGs. The DAGuE system developed by Bosilca et al. [7, 14] targets at exascale parallel programming with annotated task parallelism. It uses a concise representation of the DAG to avoid task graph unrolling and reduce the memory usage. The algebraic representation of task dependencies extracts more potential parallelism, while still enabling out-of-order task execution.

Motivated by the need of exascale applications, Meng et al. [25] investigated the portability and scalability of DAG-based computation with the Uintah software on large computing infrastructures. Through their experimental studies, the authors indicate that the adaptive DAG-based approach provides a powerful abstraction for solving challenging multiscale multi-physics engineering problems on the three of the fastest computers appeared in the top 500 list of November 2012. Bueno et al. [9] developed a task graph based programming system called OmpSs that lets a user annotate a sequential application and provides semi-automatic parallelization to simplify programming on GPU clusters.

DAG parallelism on clouds and emerging architectures. Henzinger et. al [22, 23] discussed the DAG scheduling and resource management challenges, and developed a cloud programming and pricing model which considers the tradeoff of execution speed and price on a per-job basis. Tang et al. [30] considered handling of hardware and software failures in task graph scheduling for a grid computing system in which distributed scientific and engineering applications often require multi-institutional collaboration, large-scale resource sharing, and wide-area communication.

Hardware specialization is an important architecture direction to improve microprocessor performance and transistor energy efficiency. One key specialization technique is to map large regions of computation to the hardware with a spatial architecture paradigm. Mercaldi et al. [26] developed a practical DAG scheduling algorithm that generates efficient code schedules for tiled architectures. The scheduler decides where and when an instruction will execute. For example, placing dependent instructions on the same or adjacent tiles reduces producer-to-consumer operand latency. Nowatzkiy et al. [27] studied a scheduling framework usable for various spatial architectures.

One key challenge in high performance quantum computing research is to scale experimental quantum computers from a handful of quantum bits to a large number of quantum bits. Balensiefer et al. [3] presented a set of infrastructural tools that enable the quantitative evaluation of architectures for quantum computers with thousands of quantum bits and billions of time steps. One of its tools is a device scheduler that maps an assembly source into a set of device specific primitive operations for controlling a quantum micro-architecture. The ability to process billions of operations was critical in designing the scheduler. The paper considers a trade-off of optimality for speed and employs a variant of list scheduling [32].

3. CONCLUDING REMARKS

DAG scheduling is an important technique for software tools, compilers, and runtime systems in optimizing the use of parallel computing resource. Our earlier theoretical study on task granularity [19] shows that scheduling needs to take communication overhead into account especially for distributed memory architectures. Fast scheduling heuristic algorithms can attain good performance in solving the resource optimization problem. Those scheduling techniques can be practical with an integration of a software system that aids task graph description or derivation and manages code generation and runtime support for executing DAG computation. Such systems are critical for supporting large-scale scientific and data-intensive applications in the current and future high performance computing platforms.

Acknowledgments. This work was supported in part by NSF IIS-1118106. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

4. REFERENCES

[1] D. Brown and P. Messina (Chairs). *Scientific Grand Challenges: Crosscutting Technologies for Computing at the Exascale*. Report from the 2010 Workshop. U.S. Department of Energy, 2010.

[2] Vikram S. Adve and Rizos Sakellariou. Compiler synthesis of task graphs for parallel program performance prediction. In *Proc. of 13th Inter. Workshop on Languages and Compilers for Parallel Computing-Revised Papers*, pages 208–226, 2001.

[3] Steven Balensiefer, Lucas Kregor-Stickles, and Mark Oskin. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. In *Proc. of 32nd Annual Inter. Symposium on Computer Architecture*, pages 186–196, 2005.

[4] Muthu Manikandan Baskaran, Nagavijayalakshmi Vydyanathan, Uday Kumar Reddy Bondhugula, J. Ramanujam, Atanas Rountev, and P. Sadayappan. Compiler-assisted dynamic scheduling for effective parallelization of loop nests on multicore processors. In *Proc. of 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 219–228, 2009.

[5] Pieter Bellens, Josep M. Perez, Rosa M. Badia, and Jesus Labarta. Cellss: A programming model for the cell be architecture. In *Proc. of ACM/IEEE Supercomputing'06*.

[6] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, and Y. Zhou. Cilk: An Efficient Multithreaded Runtime System. In *Proc. of 5th ACM Symposium on Principles and Practice of Parallel Programming*, pages 207–216, 1995.

[7] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Herault, Pierre Lemarinier, and Jack Dongarra. Dague: A generic distributed dag engine for high performance computing. *Parallel Comput.*, 38(1-2):37–51, January 2012.

[8] Doruk Bozdağ, Füsun Özgüner, and Umit V. Catalyurek. Compaction of schedules and a two-stage approach for duplication-based dag scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 20(6):857–871, June 2009.

[9] Javier Bueno, Judit Planas, Alejandro Duran, Rosa M. Badia, Xavier Martorell, Eduard Ayguade, and Jesus Labarta. Productive programming of gpu clusters with ompss. In *Proc. of 2012 IEEE 26th Inter. Parallel and Distributed Processing Symposium*, pages 557–568, 2012.

[10] F. T. Chong, S. D. Sharma, E. A. Brewer, and J. Saltz. Multiprocessor Runtime Support for Fine-Grained Irregular DAGs. In Rajiv K. Kalia and Priya Vashishta, editors, *Toward Teraflop Computing and New Grand Challenge Applications.*, New York, 1995. Nova Science Publishers.

[11] Michel Cosnard and Emmanuel Jeannot. Compact dag representation and its dynamic scheduling. *J. Parallel Distrib. Comput.*, 58(3):487–514, September 1999.

[12] Michel Cosnard, Emmanuel Jeannot, and Tao Yang. Compact dag representation and its symbolic scheduling. *J. Parallel Distrib. Comput.*, 64(8):921–935, August 2004.

[13] Mohammad I. Daoud and Nawwaf Kharma. A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks. *J. Parallel Distrib. Comput.*, 71(11):1518–1531, November 2011.

[14] Jack Dongarra. *Achitecture Aware Algorithms and Software for Peta and Exascale*. Presentation at Ken Kennedy Institute of Information Technology, Feb 2014.

[15] C. Fu, X. Jiao, and T. Yang. Efficient Sparse LU Factorization with Partial Pivoting on Distributed Memory Architectures. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):109–125, 1998.

[16] C. Fu and T. Yang. Sparse LU Factorization with Partial Pivoting on Distributed Memory Machines. In *Proc. of ACM/IEEE SuperComputing'96*.

[17] C. Fu and T. Yang. Run-time Compilation for Parallel Sparse Matrix Computations. In *Proc. of ACM Inter. Conf. on Supercomputing*, pages 237–244, 1996.

[18] C. Fu and T. Yang. Space and Time Efficient Execution of Parallel Irregular Computations. In *Proc. of ACM Symposium on Principles & Practice of Parallel Programming*, pages 57–68, 1997.

[19] A. Gerasoulis and T. Yang. On the Granularity and Clustering of Directed Acyclic Task Graphs . *IEEE Trans. on Parallel and Distributed Syst.*, 4(6):686–701, June 1993.

[20] Apostolos Gerasoulis and Jia Jiao. Rescheduling support for mapping dynamic scientific computation onto distributed memory multiprocessors. In *Proc. of Third Inter. Euro-Par Conference on Parallel Processing*, pages 905–912, 1997.

[21] M. Girkar and C. Polychronopoulos. Automatic Extraction of Functinal Parallelism from Ordinary Programs. *IEEE Trans. on Parallel and Distributed Syst.*, 3(2):166–178, 1992.

[22] Thomas A. Henzinger, Anmol V. Singh, Vasu Singh, Thomas Wies, and Damien Zufferey. A marketplace for cloud resources. In *Proc. of Tenth ACM Inter. Conf. on Embedded Software*, pages 1–8, 2010.

[23] Thomas A. Henzinger, Anmol V. Singh, Vasu Singh, Thomas Wies, and Damien Zufferey. Static scheduling in clouds. In *Proc. of 3rd USENIX HotCloud*, 2011.

[24] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[25] Qingyu Meng, Alan Humphrey, John Schmidt, and Martin Berzins. Investigating applications portability with the uintah dag-based runtime system on petascale supercomputers. In *Proc. of SC13: Inter. Conf. for High Performance Computing, Networking, Storage and Analysis*, pages 96:1–96:12, 2013.

[26] Martha Mercaldi, Steven Swanson, Andrew Petersen, Andrew Putnam, Andrew Schwerin, Mark Oskin, and Susan J. Eggers. Instruction scheduling for a tiled dataflow architecture. In *Proc. of 12th Inter. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 141–150, 2006.

[27] Tony Nowatzki, Michael Sartin-Tarm, Lorenzo De Carli, Karthikeyan Sankaralingam, Cristian Estan, and Behnam Robatmili. A general constraint-centric scheduling framework for spatial architectures. In *Proc. of 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 495–506, 2013.

[28] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors.* MIT Press, 1989.

[29] Oliver Sinnen, Andrea To, and Manpreet Kaur. Contention-aware scheduling with task duplication. *J. Parallel Distrib. Comput.*, 71(1):77–86, January 2011.

[30] Xiaoyong Tang, Kenli Li, Meikang Qiu, and Edwin H. M. Sha. A hierarchical reliability-driven scheduling algorithm in grid systems. *J. Parallel Distrib. Comput.*, 72(4):525–535, April 2012.

[31] Naga Vydyanathan, Umit Catalyurek, Tahsin Kurc, Ponnuswamy Sadayappan, and Joel Saltz. Optimizing latency and throughput of application workflows on clusters. *Parallel Comput.*, 37(10-11):694–712, 2011.

[32] T. Yang and A. Gerasoulis. List Scheduling With and Without Communication . *Parallel Computing*, 19:1321–1344, 1993.

[33] T. Yang and A. Gerasoulis. DSC: Scheduling Parallel Tasks on An Unbounded Number of Processors. *IEEE Trans. on Parallel and Distributed Syst.*, 5(9):951–967, 1994.

[34] Tao Yang and Cong Fu. Space/time-efficient scheduling and execution of parallel irregular computations. *ACM Trans. Program. Lang. Syst.*, 20(6):1195–1222, November 1998.