A

Infrastructure for Dynamic Adaptation of Data-Intensive Parallel C++ Applications on Non-Uniform Memory Access Systems

REPHRASE PEOPLE, University of RePhrase

CCS Concepts: •**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Additional Key Words and Phrases: Wireless sensor networks, media access control, multi-channel, radio interference, time synchronization

ACM Reference Format:
Gang Zhou, Yafeng Wu, Ting Yan, Tian He, Chengdu Huang, John A. Stankovic, and Tarek F. Abdelzaher, 2010. A multifrequency MAC specially designed for wireless sensor network applications. *ACM Trans. Architec. Code Optim.* V, N, Article A (January YYYY), 3 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Parallel patterns [**?**] provide parameterised implementations of common types of parallel operations and represent a very useful abstraction in bridging the gap between the high-level programming models in which programmers ideally want to code and the complexity of modern multi-core/many-core hardware. However, in order to keep the high-level of abstraction, many low-level implementation details related to thread and data management (such as the placement of threads and data, including possible replication) have to remain implicit to the particular pattern implementation. Furthermore, in some situations, the correct decisions about some of the issues (e.g. how to distribute the data and whether or not to replicate the data) can be made only at runtime as they depend e.g. on the input data of the application or the overall load of the system. To achieve the best parallel performance of an application on a modern parallel system, careful *static* and *dynamic* tuning of pattern realisation, guided by the dynamic information about the application execution and the available hardware, may be be required. State-of-the-art pattern libraries, such as Intel TBB [**?**], OpenMP [**?**] and FastFlow [**?**] allow dynamic tuning of some of the coarse-grained parameters of the patterns, e.g. the number of threads used in parallel sections or the type of scheduling used (static vs. dynamic). However, many of the important features of a parallel application, such as the placement of tasks and data, are either left for the operating system to deal with, or (in the case of the newest OpenMP standard) programmer is offered explicit primitives to deal statically with them.

This work is supported by the National Science Foundation, under grant CNS-0435060, grant CCR-0325197 and grant EN-CS-0329609.
Author's addresses: G. Zhou, Computer Science Department, College of William and Mary; Y. Wu and J. A. Stankovic, Computer Science Department, University of Virginia; T. Yan, Eaton Innovation Center; T. He, Computer Science Department, University of Minnesota; C. Huang, Google; T. F. Abdelzaher, (Current address) NASA Ames Research Center, Moffett Field, California 94035.

DOI: 0000001.0000001

In this paper, we present an infrastructure for static and dynamic tuning of patterned parallel applications. On the static side, we present novel mechanisms for *mapping* parallel threads and data to the underlying hardware resources (computation cores and memory banks), including possible data replication to avoid problems such as congestion of memory bus and false sharing. On the dynamic side, we present machine-learning based for scheduling/remapping application threads to different cores in the

and dynamic adaptation of patterned applications. We combine the techniques of *static mapping*, *dynamic scheduling*, *dynamic recompilation* and *performance monitoring* to provide a run-time adaptivity infrastructure that dynamically changes the application in order to tune it to the changing application behaviour and hardware setting. *Static mapping* is used for initial configuration of the applications' parallel components, in terms of the number of OS threads for each component, pinning threads/data to particular cores/memory regions and possible replication of the data. The pinning decision made by the static mapping can be altered by *dynamic scheduling* to adapt the thread and data placement to the changes in the system load. During the application execution, *dynamic recompilation*, which is used to replace on-the-fly some functions in the application with their alternative versions, further specialises the application and adapts it to the particular behaviour it is exhibiting. The whole process of dynamic adaptation via dynamic recompilation and scheduling is driven by the *Performance monitoring* mechanism which collects and analyses the information about the application and system behaviour and triggers the remapping/recompilation decisions. In this way, we are able to adapt features like structure of parallel patterns, inlining of expressions into functions, switching between parallel and sequential versions of functions, scheduling and data replication.

The specific research contributions of the paper are

— Contribution 1
— Contribution 2

2. BACKGROUND

3. REPHRASE DYNAMIC ADAPTIVITY INFRASTRUCTURE

Figure 1 shows the overview of the RePhrase dynamic adaptivity infrastructure. At the beginning, we are given a patterned application, with possibly some of the extra-functional pattern parameters (such as the number of workers in farm/map patterns and chunking parameters) still missing. It is the job of *static mapper* to "complete" the application by deriving (near-)optimal initial values for the missing parameters, as well as an initial mapping of threads and data to the underlying processor/processor cores and memory nodes. In addition, static mapper chooses the alternative implementations of the same pattern (e.g. the ones that provide data replication) that might be used by the dynamic recompilation mechanism. The initial application is, therefore, transformed into the complete application, together with selected alternative implementations of (some of) the patterns from the application. After that, *dynamic compiler* compiles the application into a binary form, and generates *intermediate representations* of the selected alternative versions of the patterns. The application is then executed on a hardware. During the execution, *performance monitor* collects various information about the application behaviour, as well as the information about the state of the hadrware and computing load, and feeds this information to the *adaptation manager*. Adaptation manager then analyses this information and re-invokes the dynamic compiler (that can compile intermediate representations of alternative versions of patterns into binary form and plug them into the application binary code instead of their default versions) or dynamic scheduler, that remaps threads and data to processor cores and memory nodes.

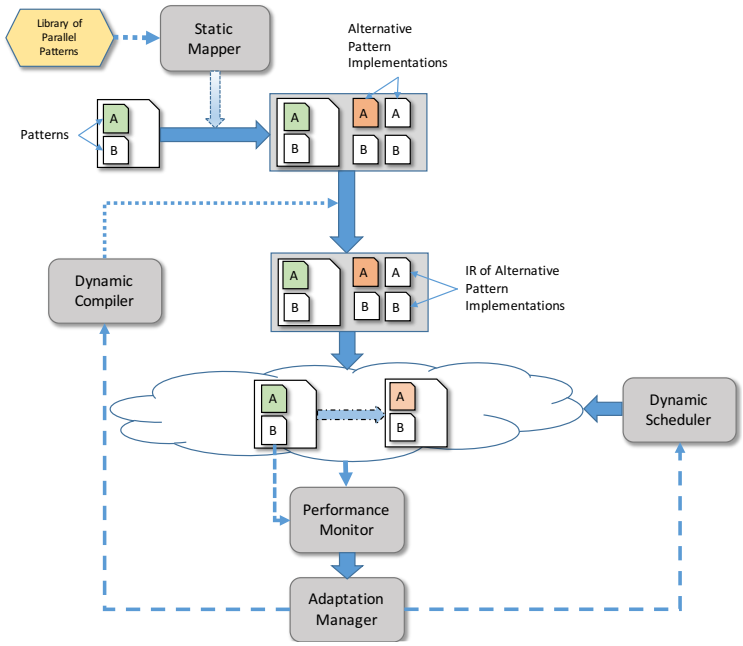

Fig. 1. RePhrase Dynamic Adaptivity Infrastructure

3.1. Static Mapper
3.2. Dynamic Scheduler
3.3. Dynamic Compiler
3.4. Performance Monitor and Adaptation Manager
4. EVALUATION
4.1. Benchmarks
4.2. Use Cases
5. RELATED WORK
6. CONCLUSIONS AND FUTURE WORK
ACKNOWLEDGMENTS

The authors would like to thank someone for something.

Received February 2007; revised March 2009; accepted June 2009

Online Appendix to: Infrastructure for Dynamic Adaptation of Data-Intensive Parallel C++ Applications on Non-Uniform Memory Access Systems

REPHRASE PEOPLE, University of RePhrase

DOI: 0000001.0000001