Adaptive structured parallelism for computational grids

Citation for published version:
González-Vélez, H & Cole, M 2007, 'Adaptive structured parallelism for computational grids'. in PPOPP. ACM Press, pp. 140-141., 10.1145/1229428.1229456

Digital Object Identifier (DOI):
10.1145/1229428.1229456

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Author final version (often known as postprint)

Published In:
PPOPP

Adaptive Structured Parallelism for Computational Grids

Horacio González–Vélez Murray Cole

School of Informatics, University of Edinburgh, Edinburgh EH9 3JZ, UK

h.gv@ed.ac.uk mic@inf.ed.ac.uk

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent Programming—Parallel programming; D.3.3 [*Programming Languages*]: Language Constructs and Features—Concurrent programming structures

General Terms Languages, Algorithms, Performance

Keywords Structured Parallelism, Algorithmic Skeletons, Parallel Language Constructs, Patterns, Computational Grids

Introduction

Algorithmic skeletons [3] abstract commonly-used patterns of parallel computation, communication, and interaction. They provide top-down design composition and control inheritance throughout the whole structure. Parallel programs are expressed by interweaving parameterised skeletons analogously to the way sequential structured programs are constructed [4, 8].

This design paradigm, known as structured parallelism, provides a high-level parallel programming method which allows the abstract description of programs and fosters portability. That is to say, structured parallelism requires the description of the algorithm rather than its implementation, providing a clear and consistent meaning across platforms while their associated structure depends on the particular implementation. By decoupling the structure from the meaning of a parallel program, it benefits entirely from any performance improvements in the systems infrastructure.

Computational grids [5] have long posed a challenge to known distributed systems programming techniques as a result of inherent heterogeneity and dynamism. Over the last decade, their study has constituted an evolving field in computer science, and the associated programming frameworks have incorporated assorted paradigms such as program composition, derivation, construction, and transformation [1, 2].

Algorithmic skeletons possess a crucial property which favours performance optimisation: their structured and predictable meaning for a given program. Nevertheless, little research has been conducted on improving performance by actively using this information from a systems infrastructure perspective.

By identifying the intrinsic properties of an algorithmic skeleton, which capture its essence and distinguish it from the rest, the *GRASP* methodology enables its instrumentation and indeed its adaptivity. Current skeletal libraries have not used these properties to predict the execution of a skeletal program. We argue that by identifying and instrumenting these properties, a structured parallelism program will be able to adapt to the dynamic grid conditions over time by steering its execution.

PPoPP'07 March 14–17, 2007, San Jose, California, USA.
ACM 978-1-59593-602-8/07/0003.

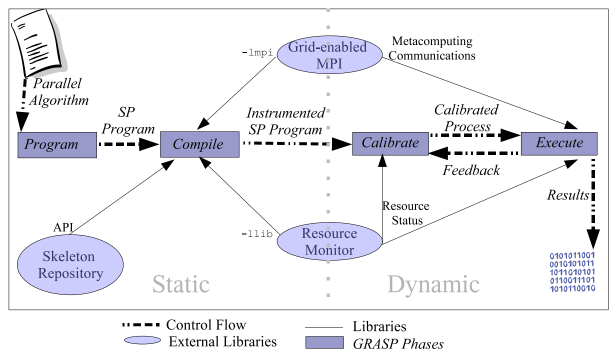

Figure 1. The four-phase *GRASP* methodology

Hence, the main objective of this work is to address the open question:

> How much can the structural forecasting information of structured parallelism help to improve the performance of parallel applications executing in a non-dedicated distributed heterogenous system (computational grid)?

The key challenges in improving such performance include the correct selection of resources (processors, links) from amongst those available, the correct adjustment of algorithmic parameters (for example, blocking of communications, granularity) and most importantly, the ability to adapt all of these factors dynamically in the light of evolving external pressure on the chosen resources.

The main difference to other performance approaches is that *GRASP* intends to be oriented toward structured parallelism, adaptable by construct, and focused on empirical, system infrastructure methodologies.

GRASP currently comprises two algorithmic skeletons, task farm [6] and pipeline [7], programmed as APIs in ANSI C.

GRASP Methodology

GRASP is a generic methodology to incorporate structural information into a parallel program at compile time that helps it to adapt at execution time. It instruments the program with a series of pragmatic rules embedded in the algorithmic skeletons, which depend on particular performance thresholds based on the nature of the skeleton, the computation/communication ratio of the program, and the availability of grid resources. As illustrated in Fig. 1, *GRASP* comprises four phases: programming, compilation, calibration, and execution.

Programming is a design phase in which the application programmer selects a suitable skeleton in order to parallelise an algorithm and interacts with *GRASP* through standard application programming interfaces. Since structured parallelism provides a high-level approach to programming, the programmer only requires to additionally supply the initialisation and termination calls for the parallel environment.

The criterion to choose a skeleton depends entirely on the nature of the parallel algorithm in hand. The programmer identifies the most suitable pattern to address the computational and communication requirements of the algorithm.

Next, the programmer needs to parameterise the API calls to *GRASP*. This parametrisation is crucial to stamp the algorithmic skeleton with correct meaning for the given problem instance. Then, the structured parallelism program is compiled and linked with the *GRASP* code, the parallel environment, and, if any, the resource monitoring library.

This parallel environment handles the underlying metacomputer/computational grid, including the node initialisation, grid resource co-allocation, inter-domain scheduling, and other infrastructure matters.

Both stages are static since they do not require any online interaction or feedback from the underlying platform.

The calibration is an autonomic stage, which executes a sample of the data on every allocated node, extrapolating the node performance in order to select the fittest nodes for the given computation under the current resource conditions. That is to say, the selection of the fittest nodes depends entirely on the resource usage of the platform at the start of execution.

Nodes are ranked by extrapolating their performance based on the execution times only (the faster a node the fitter it is), or on statistical functions, such as univariate and multivariate linear regression involving execution time, processor load, and bandwidth utilisation. This ranking involves the actual execution of the given set of functions on the complete processor pool.

```
Data: F: Set of Functions;
      P: Number of nodes;
Result: Chosen: Table of fittest processing elements;

Execute F over P nodes concurrently;
Set t ← execution times(F);
if root node then
    Collect t from P nodes into T;
    if Statistical Calibration then
        Collect processor and bandwidth values;
        Adjust T statistically;
    end
    Rank P by extrapolating performance based on T;
    Select Chosen from P;
    Send Chosen
else
    Send time from this node to root node;
    Receive Chosen;
end
```

Algorithm 1: Calibration Algorithm

The calibration procedure, as shown in Algorithm 1, accounts for the initial task-to-node allocation based on the intrinsic properties of the algorithmic skeleton. It provides the initial conditions for execution of the parallel program. It is relevant to mention that the processing performed during the calibration contributes to the overall job.

Once the fittest nodes are allocated at calibration, the execution phase monitors periodically the grid conditions and adapts the workload, i.e., if it encounters a performance bottleneck it is addressed according to the inherent properties of the algorithmic skeleton as illustrated by Algorithm 2.

```
Data: F: Set of Functions;
      Chosen: Table of fittest nodes;
      Z:Performance threshold
Result: A: Adaptive Process;

while ¬ Recalibration do
    Execute F over Chosen nodes concurrently;
    Set t ← execution times(F);
    if monitor node then
        Collect t from Chosen nodes into T;
        if min T > Z then
            Set Recalibration ← true;
        end
    else
        Send time from this node to monitor node;
    end
end
```

Algorithm 2: Execution Algorithm

By using the performance threshold while recording the execution times of the given functions, the skeleton adapts to the infrastructure by allowing performance variations up to the threshold. Once the threshold is reached, the skeleton takes action, e.g., feeding back to the calibration phase and/or modifying the task scheduling according to the inherent properties of the skeleton in hand.

Note that both stages are dynamically determined since their behaviour varies according to the overall workload and the resource conditions.

References

[1] BAL, H., CASANOVA, H., DONGARRA, J., AND MATSUOKA, S. Application-level tools. In *The Grid 2: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds., second ed. Morgan Kaufmann, San Francisco, 2003, ch. 24, pp. 463–489.

[2] BERMAN, F., CHIEN, A., COOPER, K., DONGARRA, J., FOSTER, I., GANNON, D., JOHNSSON, L., KENNEDY, K., KESSELMAN, C., MELLOR-CRUMME, J., REED, D., TORCZON, L., AND WOLSKI, R. The GrADS project: Software support for high-level grid application development. *Int. J. High Perform. Comput. Appl. 15*, 4 (2001), 327–344.

[3] COLE, M. *Algorithmic Skeletons: Structured Management of Parallel Computation*. Research Monographs in Parallel and Distributed Computing. MIT Press, Cambridge, 1989.

[4] DARLINGTON, J., KE GUO, Y., TO, H. W., AND YANG, J. Parallel skeletons for structured composition. In *PPoPP '95* (Santa Barbara, USA, 1995), ACM Press, pp. 19–28.

[5] FOSTER, I., KESSELMAN, C., AND TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl. 15*, 3 (2001), 200–222.

[6] GONZÁLEZ-VÉLEZ, H. Self-adaptive skeletal task farm for computational grids. *Parallel Comput. 32*, 7–8 (2006), 479–490.

[7] GONZÁLEZ-VÉLEZ, H., AND COLE, M. Towards fully adaptive pipeline parallelism for heterogeneous distributed environments. In *ISPA 2006* (Sorrento, Italy, Dec. 2006), no. 4330 in Lect. Notes Comput. Sc., Springer-Verlag, pp. 916–926.

[8] PELAGATTI, S. *Structured Development of Parallel Programs*. Taylor & Francis, London, 1997.