Processing of Multimedia Data using the P2G Framework

Paul B. Beskow, Håkon K. Stensland, Håvard Espeland, Espen A. Kristiansen,
Preben N. Olsen, Ståle Kristoffersen, Carsten Griwodz, Pål Halvorsen
Simula Research Laboratory, Norway
Department of Informatics, University of Oslo, Norway
{paulbb, haakonks, haavares, prebenno, espeak, staalebk, griff, paalh}@ifi.uio.no

ABSTRACT

In this demo, we present the *P2G* framework designed for processing distributed real-time multimedia data. P2G supports arbitrarily complex dependency graphs with cycles, branches and deadlines. P2G is implemented to scale transparently with available resources, i.e., a concept familiar from the cloud computing paradigm. Additionally, P2G supports heterogeneous computing resources, such as x86 and GPU processing cores. We have implemented an interchangeable P2G *kernel language* which is meant to expose fundamental concepts of the P2G programming model and ease the application development. Here, we demonstrate the P2G execution node using a MJPEG encoder as an example workload when dynamically adding and removing processing cores.

Categories and Subject Descriptors

D.1.3 [**PROGRAMMING TECHNIQUE**]: Concurrent Programming—*Parallel programming*

General Terms

Languages, Performance

1. INTRODUCTION

As the number of multimedia services grows, so do the computational demand of multimedia data processing. New multi-core hardware architectures provide the required resources, but parallel, distributed applications are much harder to write than sequential programs. Large processing frameworks like Google's MapReduce [1] and Microsoft's Dryad [3] are steps in the right direction, but they are targeted towards batch processing. As such, we present *P2G* [2], a framework designed to integrate concepts from modern batch processing frameworks into the world of real-time multimedia processing. We seek to scale transparently with the available resources (following the cloud computing paradigm) and to support heterogeneous computing resources, such as GPU processing cores. The idea is to encourage the application developer to express the application as fine granular as possible along two axes, i.e., both data and functional parallelism. In this respect, many of the existing systems sacrifice flexibility in one axis to accommodate for the other, e.g., MapReduce has no flexibility in the functional domain,

MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.
ACM 978-1-4503-0616-4/11/11.

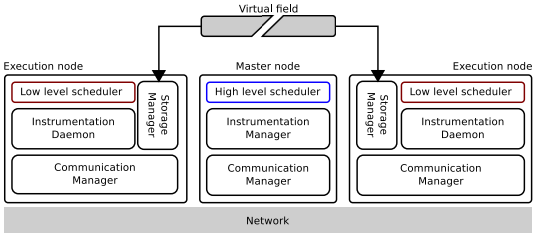


Figure 1: Overview of nodes in the P2G system.

but allows for fine-grained parallelism in the data domain. In P2G, functional blocks called kernels operate on slices of multi-dimensional fields. We use fields to store multimedia data and implicitly express data decomposition. A write-once semantics of the fields provides the needed boundaries and barriers for functional decomposition to exist in our run-time and ensures deterministic output. P2G has intrinsic support for deadlines. Moreover, the compiler and run-time analyze dependencies dynamically and then merge or split kernels based on resource availability and performance monitoring.

We have implemented a prototype of a P2G execution node. Here, we demonstrate an operating execution node of P2G using MJPEG as an example. We are able to show that P2G scales dynamically with the number of available x86 processing cores.

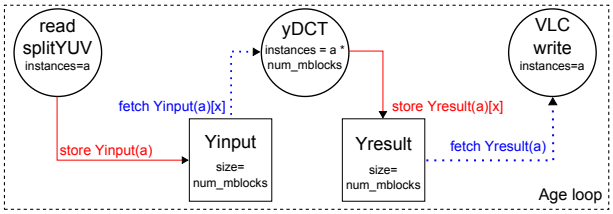
2. THE P2G ARCHITECTURE

As shown in figure 1, P2G consists of a *master node* and an arbitrary number of *execution nodes*. Each execution node reports its local topology (i.e., multi-core, GPU, etc) to the master node, which combines this information to form a global topology of available resources. As such, the global topology can change during run-time as execution nodes can be dynamically added and removed to accommodate for changes in the global load.

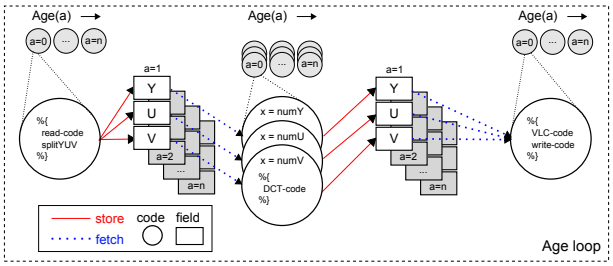
As the master node receives workloads, it uses its high-level scheduler to determine which execution nodes to delegate partial or complete parts of the workload to. This process can be achieved in a number of ways. However, as a workload in P2G forms an implicit dependency graph based on its *store* and *fetch* operations, the high-level scheduler can utilize graph partitioning algorithms, or similar, to map such an implicit dependency graph to the global topology. The utilization of available resources can thus be maximized.

P2G uses a low-level scheduler at each execution node to maximize the local scheduling decisions, i.e., the low-level scheduler can decide to combine functional and data decomposition to minimize overhead. During run-time, the

master node will collect statistics on resource usage from all execution nodes, which all run an instrumentation daemon to acquire this information. The master node can then combine this run-time instrumentation data with the implicit dependency graph derived from the source code and the global topology to make continuous refinements to the high-level scheduling decisions. As such, P2G relies on its combination of a high-level scheduler, low-level schedulers, instrumentation data and the global topology to make best use of the performance of several (possibly heterogeneous) cores in a distributed system.



(a) Implicit dependency graph for the Y-component



(b) Kernel and field instances

Figure 2: Programming model (MJPEG example)

P2G provides a kernel language for the programmer to write their application in, which they do by writing isolated, sequential pieces of code (kernels). Kernels operate on slices of *fields* through *fetch* and *store* operations and have native code embedded within them. In this model, we encourage the programmer to specify the inherent parallelism in their application in as fine a granularity as possible in the domains of both functional and data decomposition.

The multi-dimensional fields offer a natural way to express multimedia data, and provide a direct way for kernels to *fetch* fine granular data slices. A write-once semantics of the fields provides deterministic output, though not necessarily deterministic execution of individual kernels. Given write-once semantics, iteration is supported in P2G by introducing the concept of aging, as seen in figure 2(a), where storing and fetching to the same field position, at different ages, makes it possible to form loops. The write-once semantics also provides natural boundaries and barriers for functional decomposition, as the low-level scheduler can analyze the dependencies of a kernel instance to determine if it is ready for execution. Furthermore, the compiler and the run-time can analyze dependencies dynamically and merge or split kernels based on resource availability and performance monitoring.

Given a workload specified using the P2G kernel language, P2G is designed to compile the source code for a number of heterogeneous architectures. At the time of writing, P2G consists of what we call an execution node, which is capable of executing workloads on a single x86 multi-core node. As such, the implementation of a high-level scheduler, support for heterogeneous architectures and distribution mechanisms are ongoing work.

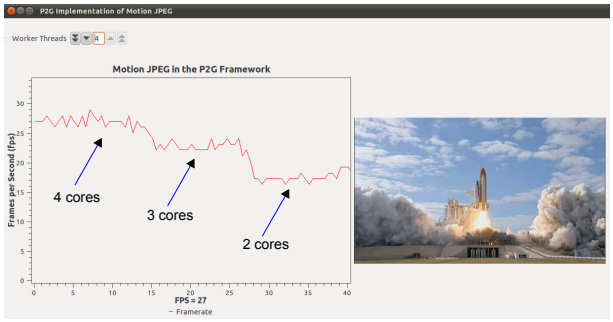


Figure 3: Screenshot of the MJPEG demo.

3. WORKLOAD

We have implemented several simple workloads used in multimedia processing, such as matrix multiplication, k-means clustering to test the prototype implementation of P2G. In this demo, we will focus on our implementation of the MJPEG encoder.

Figure 2(b) shows how the MJPEG encoding process is split into kernels and their running instances. YUV-input video are read into 8x8 macro-blocks and stored in three global fields, by the *read + splitYUV* kernel. For example, given the CIF resolution of 352x288 pixels per frame used in our tests, this generates 1584 macro-blocks of Y (luminance) data, each with 64 pixel values. This makes it possible to create 1584 instances per age of the *DCT* kernel transforming the luminance component of the picture. Each of these kernel instances stores the DCT'ed macro-block into global result fields. Finally, the *VLC + write* kernel does variable length coding and stores the MJPEG bit-stream to disk.

4. DEMONSTRATION

In this demo, we will explain and discuss the P2G ideas for multimedia processing. A screenshot of the MJPEG encoder running in the P2G framework is shown in figure 3. We are able to dynamically add and remove processing cores available to the framework. To show the performance, we live plot the achieved frame-rate from the encoder (left) and show a preview of the encoded video stream (right) as seen in the screenshot.

Acknowledgements

This work has been performed in the context of the *iAD* centre for Research-based Innovation (project number 174867) funded by the Norwegian Research Council.

5. REFERENCES

[1] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. In *Proc. of USENIX OSDI* (2004), pp. 10–10.

[2] Espeland, H., Beskow, P. B., Stensland, H. K., Olsen, P. N., Kristoffersen, S., Griwodz, C., and Halvorsen, P. P2G: A framework for distributed real-time processing of multimedia data. In *Proc. of SRMPDS 2011* (2011), IEEE.

[3] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. In *Proc. of ACM EuroSys* (2007), ACM, pp. 59–72.