The handle http://hdl.handle.net/1887/18622 holds various files of this Leiden University dissertation.

Author: Vu, Van Thieu
Title: Opportunities for performance optimization of applications through code generation
Issue Date: 2012-03-22
Chapter 1

Introduction

The accuracy of weather forecast has a great impact to many aspects of social and economic life, such as manufacturing, traffic, agriculture, forestry, and other activities. Research and development of numerical weather forecast systems started in the beginning of the 20th century. Today, numerical weather forecast is still an important research direction in computational science.

Numerical weather prediction is a computationally intensive problem. Typically, a forecast of the next day's weather requires about a quadrillion ($10^{15}$) arithmetic operations. For manipulating the immense datasets and performing the huge number of calculations, numerical weather forecast is usually performed on powerful supercomputers. However, even with the increasing power of supercomputers, weather forecast is only of value to about ten days ahead, and sometimes with a poor quality. The reason is that atmospheric circulation processes are inherently unstable phenomena, and the governing equations of these processes are nonlinear. Another reason that leads to inaccurate weather forecasting is the limited resolution of numerical forecast models in general. Grid points lie tens of kilometers apart, too coarse for modeling local meteorological effects. The quality of weather forecast can be improved by refining the resolution. However, this results in more calculations which require more computing power.

Coding for computationally intensive problems, such as the weather forecast model, usually faces a number of issues. The main task is to correctly translate the problem from the underlying mathematical theory into an efficient algorithm written in some programming language and to maintain the program when the problem specification is modified. Furthermore, making these programs efficient on different types of computer architectures is a difficult task, because each computer system requires different techniques to achieve the optimal performance.

More specifically, parallel computers can significantly speed up weather forecasting. Basically, the forecast is computed by each processor in small patches
Chapter 1. Introduction

of the atmosphere in parallel. Because the calculations on a certain grid point require information from neighboring grid points, data communication between processors is required. As a consequence, the run time of a parallel program involves communication time, which is a threat to efficiency. The negative impact of the communication time can be minimized by overlapping communication with computation.

Although the basic principles of parallel computing are clear, the parallel implementation of weather forecast models is still a complicated task. Moreover, the appearance of several different types of parallel computer architectures has made application development considerably more difficult, because a single, general paradigm for parallel programming does not exist as of yet.

Weather forecast prediction has achieved great progress from the use of large-scale parallel computing. However, a recent study shows that simply increasing the large-scale parallelism will result in poor performance for many scenarios where strong scaling is required [46]. Another option to parallel processing is to use graphic processing units (GPU). Today, GPUs are being used for many high performance computing applications as low-cost, low-power, and very high performance alternative to regular CPUs.

1.1 The HIRLAM weather forecast model

HIRLAM [29] stands for HIgh Resolution Limited Area Model and is a state-of-the-art numerical weather analysis, forecasting and postprocessing system. It is a cooperative project of Denmark, Estonia, Finland, Iceland, Ireland, the Netherlands, Norway, Spain, and Sweden. The aim of this project is to develop and maintain a numerical short-range weather forecast system for operational use by the participating institutes. The HIRLAM forecast system is now used in routine weather forecasting by the national weather centers of Denmark, Finland, Ireland, The Netherlands, Norway, Spain, and Sweden.

The HIRLAM weather forecast model is one of the three components in the HIRLAM system. It consists of two main modules, the dynamics and the physics. The dynamics solves the so-called primitive equations, describing conservation of horizontal momentum, energy and mass (air and water), and the ideal gas law per grid point. The physics parameterizes the effects of sub-gridscale processes like turbulence, and of the non-adiabatic processes like radiation, phase transitions and exchange with the earth surface. Contrary to the physics scheme, the dynamics equations contain horizontal gradients. Computationally, this implies that the physics does not require horizontal communications, but the dynamics does.

The HIRLAM system solves the dynamics equations by discretization, in time to time steps and in space to grid points. Typically, operational domain sizes are $1000 \times 1000$ grid points in the two horizontal dimensions, and $60$ grid
points in the vertical dimension. The number of computations and communications is proportional to the number of time steps. Hence, the time step should be taken as large as possible. In a straightforward explicit Eulerian discretization scheme, the time step is limited by numerical stability, not by considerations of accuracy. To remove the limitations on the time step posed by numerical stability, the HIRLAM system uses a semi-implicit semi-Lagrangian scheme. Accuracy and numerical stability in the physics allow the time step to be approximately ten times the time step allowable in an explicit Eulerian scheme. This reduces the number of floating point operations, and hence pure computation time, by a factor of 10. On the other hand, it introduces the need for long-range communications: the semi-implicit scheme needs global communications, and the semi-Lagrangian requires long-range communications, whereas communications for an explicit Eulerian scheme do not extend more than 2 grid points away. This range of influence is usually called the halo zone of the computational domain.

This thesis does not address the problems of long-range communications. Some of those problems have been subject of earlier studies with the CTADEL system [45]. Furthermore, the additional costs of long-range communications have the tendency to outgrow the savings by the semi-implicit semi-Lagrangian schemes. In this thesis this effect is also observed, when it is noted that communication time on a GPU-based system may easily exceed computation time.

1.2 The code generation tool CTADEL

The code generation tool CTADEL was developed at Leiden University by Robert van Engelen [18]. The design objective of CTADEL is the so-called “machine-independent programming-in-the-large environment”\(^1\), which must be able to generate efficient execution codes for different computer architectures typically from architecture-independent problem specifications. The problem, specified in a high-level language, is transformed into an Abstract Syntax Tree (AST) using a transformation-database. On the AST global optimizations, common subexpression elimination and architecture dependent optimizations are performed before the resulting code is generated. These transformations ensure the generation of correct and efficient codes, which can compete with hand optimized codes.

The overview of the CTADEL system is shown in Figure 1.1. For more detail, the reader is referred to [18]. CTADEL generates output code from a model specification through the following steps:

\(^1\)Programming-in-the-large environment is concerned with the overall architecture of software systems. It deals with the composition of large systems out of modules, the interfaces between the modules and their specification, and the evolution of the resulting architecture over time [77].
Chapter 1. Introduction

Figure 1.1: The overview of the CTADEL system

- Dimensional analysis of the model;
- Discretization of the continuous equation;
- Common-subexpression elimination and optimization of the intermediate code for a target computer architecture;
- Generate output code.

The kernel components of CTADEL are:

Scripts A collection of scripts is provided containing libraries of PDE-based (Partial Differential Equation) operators, skeletons of computer codes, and pre-defined procedures for symbolic manipulation. The specification of the model is also specified as a script. Loading and compiling of scripts is via a terminal-based command interface.

Rulebases A collection of rulebases containing various transformation rules and strategies for applying transformations. The rulebase transformation strategy can be iteratively applied on expressions.

Parser The parser scans the input, analyses the syntax and parses scripts and user commands.

Symbolic evaluator Expressions are symbolically evaluated which results in expansion of symbolic functions and procedures and the evaluation of symbolic expressions.

Inference engine A model specification may contain typing information to ensure the correctness of the specification. Besides the primary types such as
1.3 Finite elements method

Until now, all codes which have been generated by CTADEL use finite difference discretization techniques. With a finite difference method, the derivatives in a differential equation are approximated by linear combinations of function values at the grid points. Finite difference methods are simple to implement, but these methods are restricted to problems defined on regular geometries, such as an interval in a one-dimensional space, a rectangular domain in a two-dimensional space, and a cubic in a three-dimensional space. With complex geometries and irregular physical structures, finite element methods are better suited. In general, finite element methods are used to study many problems, such as structural and solid mechanics, fluid flow, heat transfer, electricity and magnetism, and various coupled problems. In order to generate code for problems defined on complex domains, in this project we extend CTADEL to finite element methods.
1.4 GPU computing

NVIDIA Graphics Processing Units (GPUs) [58] have seen a rapid development in performance in recent years. As a result, there is growing interest in using GPUs as computational resources for many high performance computing problems. A GPU comprises a set of so-called multiprocessors (MP). Each MP contains a number of scalar processor cores (SP). A GPU has a multi-level memory hierarchy including registers, local, shared, global, constant, and texture memory. Each SP has access to registers and a local memory. A shared memory is accessible by an MP. All SPs of a GPU can access global and constant memory. Registers and shared memory are on-chip. Access to these memories takes a few cycles. Other memories of a GPU are off-chip which have an access time of hundreds of cycles. The CPU and the GPU have separate memories, which is not accessible by each other. Therefore, the input data is transferred to the GPU memory before it is processed and the output result is transferred back to the CPU memory after processing. These transfers are performed on the PCI bus. Due to the limited bandwidth of the PCI bus, transferring input/output data may have an important impact on the performance of an application on the GPU.

Programming GPUs for general-purpose application is enabled by the Compute Unified Device Architecture (CUDA) programming model [57]. CUDA is an extension of C that allows developers to use C as a high-level programming language. The CUDA programming model provides an abstraction of the GPU parallel architecture using a minimal set of programming constructs such as hierarchy of threads, hierarchy of memories, and synchronization primitives.

1.5 Research goals and thesis outline

The goals of this research is to find opportunities to increase the performance of the HIRLAM weather forecast model. In order to accomplish this task, the parallel implementation of the dynamics routine of the HIRLAM weather forecast model is improved by overlapping communications with calculations. Secondly, this dynamics routine is ported to run on GPUs. These modifications require a lot of time to redesign the program. To reduce this coding time, the approach of automatic code generation is investigated. This code generation is used to extend the CTADEL system to incorporate finite element methods. Also, the usefulness of code generation is evaluated for parallelizing the HIRLAM weather forecast model and for porting the whole HIRLAM model to run on GPUs platforms. The outline of the thesis is as follows:

Chapter 2 Previous studies have shown that CTADEL is able to generate efficient code for finite difference methods. We start this thesis by demonstrating that this can also be achieved for finite element methods. This chapter describes
how to extend CTADEL to generate code for the Galerkin finite element schemes. Next, we show how to apply this technique to generate code for the Shallow-Water equations. Parts of this chapter have been published in [81, 82].

Chapter 3 The HIRLAM weather forecast model code has been designed to run on a massively parallel machine. Parallelization is achieved by domain decomposition. The communication routines in the parallel program are from MPICH [48] and OpenMP [60]. A time step consists of a communication step with neighboring processors followed by a calculation step over the domain assigned to a processor. In this chapter we report on experiments to start the calculations while the communications are still in progress. The aim is to reduce the overall execution time by hiding the communications behind the calculations. The obvious advantage of overlapping communications with calculations has a counterweight in increased calculation time. The balance between gains and losses may depend on the computer hardware. Hence, we introduce a range of strategies for overlapping, ranging from one that may be the fastest on vector type machines, to one that aims at maximum overlap, regardless of the additional computational costs of overlapping. The experiments with these overlapping strategies are performed on the DAS-3 system [67]. Parts of this chapter have been published in [79, 80].

Chapter 4 In this chapter we show how to extend CTADEL so that it can generate SPMD parallel programs. We apply this technique to generate parallel code for the Shallow-Water equations and perform experiments on DAS-3. The experimental results show that the parallel code generated by CTADEL has a good scalability and speedup. Parts of this chapter have been published in [79, 80].

Chapter 5 This chapter presents the CUDA implementation of the dynamics of the HIRLAM weather forecast model on GPUs. We convert the original Fortran code to C and CUDA by hand. In the dynamics the calculations on a certain grid point require information from horizontally neighboring grid points. In CUDA threads are organized into blocks. As a consequence synchronization between blocks is needed. However, since the blocks are required to execute independently in an arbitrary and undefined order, CUDA does not support synchronization between blocks itself. To solve this problem, we propose two methods to create independent blocks and assess the performance of them. In a CUDA program, coping data over the PCI bus between CPU and GPU is expensive. We minimize the influence of the copy time by overlapping data transfers with calculations using the stream context of CUDA. The memory of a single GPU limits the computational domain of the dynamics to be 4 times smaller than the operational domain. We enlarge the domain size of the dynamics by using multiple GPUs. Parts of this chapter have been published in [83, 84].
CUDA programming model is complicated. In this chapter we show how to extend CTADEL so that it can automatically generate CUDA programs. We define templates to generate code for parts of a CUDA program. To overlap calculations with data transfers, we implement a technique to generate the optimal CUDA stream program. We apply this technique to the dynamics of the HIRLAM weather forecast model. The experimental results show that the CUDA generated code is more efficient than the handwritten code. Parts of this chapter have been published in [83, 84].

Chapter 7 This chapter presents the conclusions of this thesis.