Tomorrow's Weather Forecast: Automatic Code Generation for Atmospheric Modeling

ROBERT VAN ENGELEN AND LEX WOLTERS
Leiden University
GERARD CATS
Royal Netherlands Meteorological Institute

The Ctadel programming environment transforms high-level PDE problem specifications into efficient codes for serial, vector, and parallel computer architectures using computing-cost heuristics and architecture-specific symbolic transformations. Ctadel-generated codes for the Hirlam weather forecast system perform comparably with handwritten codes.

Many decisions in today's society are made based on short- and long-term weather forecasts. Because of the major economic impact of such decisions, research and development of numerical weather forecast systems has been ongoing since the beginning of this century. Today numerical weather forecasting, as part of climate modeling, is classified as one of the Grand Challenges in computational science.

Simulating atmospheric processes is computationally intensive: a typical forecast of the next day's weather requires about a trillion (10^{12}) arithmetic operations. Even given the immense processing power of today's supercomputers, a 24-hour weather forecast can still take an hour to complete. Generally, no more time than this can safely be allotted, in order to meet the time constraints imposed by the practice of updating forecasts every 6 hours.

Despite the allocation of significant computing power, weather forecasts can sometimes be disappointingly poor. One of the reasons is that atmospheric circulation processes comprise inherently unstable phenomena, and the mathematical equations governing these processes are nonlinear: a small disturbance of the atmosphere at one part of the globe may have an disproportionately large effect on atmospheric motion somewhere else. In fact, E.N. Lorenz's 1963 discovery of the mathematical implications of this phenomenon[1] gave an important impetus to the current formulation of chaos theory. Because of his work, we now know that the sensitivity of weather forecast models to the exact starting conditions limits the accuracy of forecasts. It also limits the validity of long-term predictions to about two weeks ahead, at the most.

Another reason for inaccurate weather forecasts is the limited resolution of forecast models in general. Grid points lie tens of kilometers apart, too coarse for modeling local meteorological effects. Though the problem of sensitivity cannot be overcome, we can still improve short-term forecasts by improving resolution. However, since this increases the total number of operations to be performed within the time span allocated, computing power must increase accordingly.

Parallel computers can significantly speed up weather forecasting. Basically, the forecast is computed for small patches of the atmosphere in parallel. Because the atmosphere can be tiled into smaller patches, the computing time for a forecast is inversely proportional to the

1070-9924/97/$10.00 © 1997 IEEE

number of patches, assuming that each patch can be handled by one processor of the parallel computer. L.F. Richardson had already envisaged this principle in 1922, before the first electronic computer was built![2]

Although the basic principles of parallel computation are clear, the parallel implementation of weather forecasts is still a major research topic. One reason is that the appearance of several different types of parallel architectures has made application development considerably more difficult, because a single, general paradigm for parallel programming does not yet exist. Large-scale serial versions of scientific applications can require extensive recoding to transform them into scalable parallel codes. Luckily, in a field such as weather forecasting, in which the model has an inherently large component of parallelism, this effort can be well worth it. Faced with a parallel computer architecture that has specific exploitable hardware characteristics, a developer can employ various software tools to meet the demand for a new, efficient version of code for that machine. These tools include

- restructuring and parallelizing compilers,
- portable computer languages,
- numerical software libraries, and
- problem-solving environments.

These software solutions are complementary; they do not necessarily rule each other out, and a combination is likely to be the most effective. Experience has shown that automatically parallelized codes (created with the first of the tools above) are not always efficient, since advanced restructuring compilers cannot effectively restructure low-level source codes that lack high-level information. Furthermore, restructuring compilers need relatively "clean" code without many of the programming tricks that human programmers often use. This may lead to adopting another solution: (re)write codes in a (higher-level) portable computer language. Portability and efficiency, however, are conflicting goals. Portable codes do not optimally exploit the architecture of a target computer. A similar observation holds for portable numerical libraries. These observations more or less summarize the current software crisis in scientific computing, which is becoming more apparent as more types of parallel computers emerge.

Problem-solving environments,[3] or PSEs, can provide a programming methodology for application development by means of *software synthesis*. By software synthesis we mean the automatic translation of a problem—defined at a high level of abstraction—into executable code, by stepwise refinement. Using software synthesis, we can react much more quickly to exploit improvements in computer architectures, mathematical and numerical models, solution methods and algorithms, and visualization tools. PSEs that use software synthesis can produce new codes that are updated and fine-tuned to the current state of the art in an application area. This has a major impact on the software engineering methodology and therefore economical consequences for the development and maintenance of large-scale scientific applications, one reason for the growing importance of problem-solving environments.

The Ctadel system

A software development environment that integrates software solutions for a specific application is called an *application driver*. For this kind of integrated system in a production or performance-critical environment, it is important that the generated codes be at least comparable in efficiency to handwritten codes.

We made a first attempt to implement this idea of an integrated environment for the Hirlam[4] weather forecast system. This has resulted in a prototype system called *Ctadel* (*c*ode-generation *t*ool for *a*pplications based on *d*ifferential *e*quations using high-level *l*anguage specifications).[5] Part of the system is based on practical experience in manually parallelizing the Hirlam routines for atmospheric dynamics.[6]

Coping with different architectures

Different computer architectures have different hardware characteristics that can be exploited for efficient computation. On the highest level, different discretization methods and solution algorithms for PDE problems yield varying performance results depending on the target computer architecture. For weather forecast models, however, the choice of solution algorithms is limited to explicit and semi-implicit finite-difference methods and so-called spectral methods.

Another architecture-specific issue that is dealt with at the highest level is the adoption of a communication library for generating codes for distributed parallel computer systems. Ctadel uses a domain-splitting method that guarantees minimal interprocessor communication overhead by distributing all data prior to the computations and allowing communication and computation

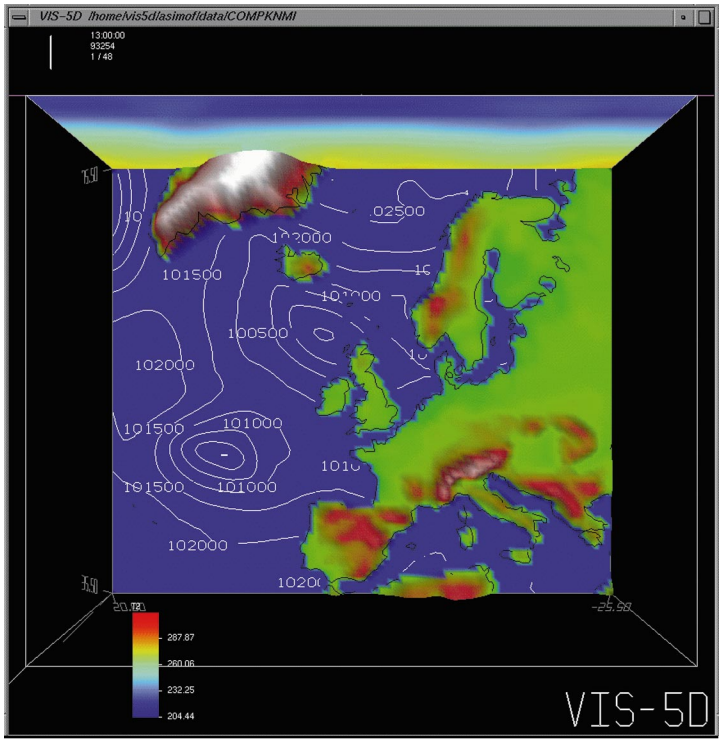

Figure 1. A Hirlam weather forecast as visualized by Vis5D.

to be interleaved. The data volume communicated is minimal because Ctadel automatically determines which data is shared by processors in so-called "halos," which are overlapping ring-shaped regions resulting from distributing the global domain over the processor grid.

On the intermediate level, Ctadel uses different strategies for applying the rewrite rules when optimizing the code for different computer architectures. For example, the conditional expressions used for specifying boundary conditions follow an architecture-specific transformation when being optimized for a SIMD machine.

On the lowest hardware level, Ctadel takes into account that each memory and arithmetic operation executed incurs a specific cost in processing time. Especially when common subexpressions are found that can be replaced with a temporary variable in the form of an array variable, the additional cost of loads and stores should be carefully balanced against the total number of arithmetic operations saved. Since it is far too expensive to profile all possible code configurations with all combinations of common subexpressions during runtime, we use heuristic measures that estimate the computing cost/gain of these subexpressions based on the relative costs of memory and arithmetic operations of the target computer. Ctadel uses a code refinement algorithm that makes a trade-off between introducing a temporary variable for a common subexpression (which requires additional storage) and the expected computing time saved.

These three levels of architecture-specific optimizations enable us to generate efficient codes from a high-level PDE specification of the Hirlam weather forecast model.

The Hirlam system

The Hirlam system,[7] a cooperative project of Denmark, Finland, Iceland, Ireland, the Netherlands, Norway, Spain, and Sweden, is a "limited-area model" encompassing a local, and often logically rectangular, part of the globe (see Figure 1). In the figure the Hirlam output is visualized using Vis5D.[8]

Both limited-area and global-area forecast models have two main computational components:

- the *dynamics*, primarily concerned with the fluid dynamics of the atmosphere, and
- the *physics*, concerned with the computation of physical parameterizations of subgrid processes.

The dynamics routines solve the "primitive equations" that describe the behavior of what are called the "prognostic variables": wind, temperature, and humidity. The physics routines compute the aggregate effect of subgrid processes and processes that are not described by the primitive equations. Examples are condensation, solar radiation, and vertical diffusion in the atmosphere. Both the dynamics and physics operate on three-dimensional grids.

Hirlam dynamics

The Hirlam dynamics describe the dynamical tendencies of the prognostic variables (see the sidebar on Governing Equations), the horizontal diffusion, and boundary relaxation. Because the equations are nonlinear and computing time is limited, the numerical schemes are based on explicit second-order finite-difference methods in space and time with a semi-implicit correction in time resulting in solving a set of Helmholtz equations.[4]

In our research efforts, we first focused on generating codes for the dynamics of the Hirlam forecast model, because the dynamics are well documented and the mathematical equations governing the atmosphere provide a clear formal specification of the problem. This contrasts

with the Hirlam physics, of which only parts are documented. In principle, however, the necessary semantic information for the physics can also be incorporated in Ctadel.[9]

The Ctadel specification language

Today a large collection of problem-solving environments and software tools exist for PDE-based problems.[10] Well-known systems are Ellpack, Deqsol, SciNapse, and in the field of parallel computing Parallel Ellpack. However, none of the existing systems provides a problem specification language that can specify a full weather forecast model in sufficient detail. The large PDE problems describing, for example, atmospheric or ocean circulation models require the computation of aggregate operations on the PDE's right-hand sides such as quadratures, Fourier transforms, and differentiation using different types of finite-difference stencils.

Like most problem specification languages of existing PSEs, the Ctadel language is declarative. The advantage of declarative and functional programming languages is the complete absence of side effects, which gives the underlying transformation system a maximum of freedom in algebraically transforming the PDEs. In contrast, imperative programming languages allow possible side effects to occur, which imposes a severe impediment for the optimization of codes.

For the Ctadel language constructs, there are four problem abstraction levels:

- PDE problem in vector notation
- scalar PDE problem
- numerical solution schemes
- source code

In this taxonomy, the highest level of abstraction is the PDE problem in vector notation and the lowest level is the application's source code. In between, each intermediate form represents a problem from a higher level of abstraction in a new lower form by adopting specific application knowledge and implementation details. Note that the presented language levels reflect a "separation of concerns" for the implementation of a PDE model.

The rationale for a PDE specification language with mixed low- to high-level representations is to allow the user to have full control over the starting point of the automatic translation process. Although Ctadel contains many details on how to transform a PDE problem in

Governing Equations of the Hirlam Weather Forecast Model

The model is defined on a (x, y, η) spherical coordinate system, with a hybrid vertical coordinate η that follows the terrain contours on the ground level and layers of constant atmospheric pressure on higher levels.

The model contains five essential equations. The *momentum equation* describes the motion of the atmosphere:

$$\frac{\partial \mathbf{V}}{\partial t} = (f+\xi)\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\mathbf{V} - \dot{\eta}\frac{\partial \mathbf{V}}{\partial \eta} - \frac{1}{\mathbf{h}}\left(R_d T_v \nabla \ln p - \nabla\left(\Phi + \frac{1}{2}(u^2+v^2)\right)\right). \quad \text{(A)}$$

The *thermodynamic equation* describes the tendency of the temperature T using the principle of conservation of energy:

$$\frac{\partial T}{\partial t} = -\frac{u}{r h_x}\frac{\partial T}{\partial x} - \frac{v}{r h_y}\frac{\partial T}{\partial y} - \dot{\eta}\frac{\partial T}{\partial \eta} + \frac{\kappa T_v\left(\int_\eta^1 \nabla\cdot\left(\mathbf{V}\frac{\partial p}{\partial \eta}\right)d\eta + \mathbf{V}\cdot\nabla p - \int_0^1 \nabla\cdot\left(\mathbf{V}\frac{\partial p}{\partial \eta}\right)d\eta\right)}{(1+(\delta-1)q)p}. \quad \text{(B)}$$

The *moisture equation* describes conservation of atmospheric humidity q:

$$\frac{\partial q}{\partial t} = -\frac{u}{r h_x}\frac{\partial q}{\partial x} - \frac{v}{r h_y}\frac{\partial q}{\partial y} - \dot{\eta}\frac{\partial q}{\partial \eta}. \quad \text{(C)}$$

The *hydrostatic equation* defines the geopotential Φ:

$$\frac{\partial \Phi}{\partial \eta} = -\frac{R_d T_v}{p}\frac{\partial p}{\partial \eta}. \quad \text{(D)}$$

Finally, the *continuity equation* expresses the conservation of mass:

$$\frac{\partial}{\partial \eta}\frac{\partial p}{\partial t} + \nabla\cdot\left(\mathbf{V}\frac{\partial p}{\partial \eta}\right) + \frac{\partial}{\partial \eta}\left(\dot{\eta}\frac{\partial p}{\partial \eta}\right) = 0. \quad \text{(E)}$$

In the equations above, $\mathbf{V} = (u, v)^T$ is the horizontal wind velocity vector, p is the pressure, f is the coriolis force, the relative vorticity ξ is defined as

$$\xi = \frac{1}{r h_x h_y}\left(\frac{\partial (h_y v)}{\partial x} - \frac{\partial (h_x u)}{\partial y}\right),$$

the virtual temperature T_v is defined by $T_v = (1 + (R_v/R_d - 1)q)T$, the vector $\mathbf{h} = r(h_x, h_y)^T$ is a vector of metric coefficients that define spherical coordinates, and ∇ is the spherical horizontal gradient operator.

vector notation into application code, it is still necessary on occasion to give a discrete operator to bypass the automatic discretization. Ctadel provides feedback in the form of LaTex typeset output so the user can verify the results of the translation.

Mathematical vector notation

By using several notational conventions, an expert PDE model developer can exploit the expressive power of mathematical vector notation to formulate a PDE problem in a mathematically concise way. Ctadel implements a translation mechanism with inherent vector and matrix semantics to transform vector equations into scalar equations. The translation mechanism follows standard notational conventions for PDE operators and adopts a Matlab-like programming style for symbolic matrix and vector operations.

As an illustrative example we will consider a subexpression of the thermodynamic equation in the Hirlam model:

$$\int_0^1 \nabla \cdot \left(\mathbf{V} \frac{\partial p}{\partial \eta} \right) d\eta . \tag{1}$$

We can specify Equation 1 as

```
V := [u, v].
int(nabla .* (V * diff(p, eta)),
    eta = eta(0) .. eta(nlev)).
```

where `nabla` denotes ∇ and ".*" denotes the inner-product operator. Note that the integration domain includes grid information for η. Except for some trivial cases, a continuous domain cannot be automatically mapped on a discrete domain because of the difference in handling boundary conditions.

Vector notation is useful only when an appropriate coordinate system can be given. In Ctadel different coordinate systems can be used by setting two special vectors to the coordinates and metric coefficients that define the desired coordinate system. For Hirlam they are

```
coordinates := [x, y].
coefficients := r * [h_x, h_y].
```

where `r` is the earth's radius and `h_x` and `h_y` are metric coefficient arrays used in the Hirlam system. This results in a scalar form that is output in LaTex typesetting as

$$\int_{\eta(0)}^{\eta(nlev)} \frac{1}{h_x h_y} r^{(-2)} \left(\frac{\partial \left(h_y r u \frac{\partial p}{\partial \eta} \right)}{\partial x} + \frac{\partial \left(h_x r v \frac{\partial p}{\partial \eta} \right)}{\partial y} \right) d\eta \tag{2}$$

Discretization

The numerical schemes for the Hirlam forecast model are based on finite-difference methods both in space and time. In combination with finite-difference methods, the use of staggered grids[11] is most common for discretizing atmospheric models (see the sidebar on Finite-Difference Methods).

In Ctadel we view the higher-level continuous operations in a physical model as abstractions for discrete operators used in the numerical schemes for the model. The bridge between a model with continuous derivatives and integrals and the numerical schemes with stencil operations and quadratures is laid by employing so-called operator overloading techniques, one of the foundations of type inference in compilers for languages with polymorphic functions. The derivative operator, for example, is overloaded with several alternative finite-difference stencils of equal approximation order, for instance,

$$\frac{\partial u}{\partial x} \approx \delta_x^+ u = \frac{u_{i+1} - u_i}{\Delta x}$$

or

$$\frac{\partial u}{\partial x} \approx \delta_x^- u = \frac{u_i - u_{i-1}}{\Delta x} .$$

The stencil selected depends on the location of variables on whole or half grid points.

Ctadel uses an iterative-deepening search algorithm for searching among possible discrete configurations of an expression for the "cheapest" one, which is the discrete form that requires none or at least a minimum total number of grid-staggering conversion operations. In this way, Equation 2 is translated into the discrete form

$$\frac{1}{r\, hxt_{i,j}\, hyt_{i,j}} \left(\delta_x^- \left(hyu_{i,j} \sum_{k=0}^{nlev-1} \delta_\eta^+ \overline{p_{i,j,k}}^{\,x} u_{i,j,k} \right) + \delta_y^- \left(hxv_{i,j} \sum_{k=0}^{nlev-1} \delta_\eta^+ \overline{p_{i,j,k}}^{\,y} v_{i,j,k} \right) \right) \tag{3}$$

Finite-Difference Methods and Staggered Grids

Centered finite differences are desirable to use in atmospheric models because they provide a second-order approximation resulting in numerically stable schemes with a minimum of arithmetic. So-called "staggered grids" make it possible to use centered differences while guaranteeing a coherent solution by avoiding the problem of numerical decoupling present with central differences. The most commonly used staggered grids are the Arakawa A-, B-, C-, and E-grids shown in Figure A.

The grids locate variables u, v, and T differently on whole and on half grid points. For example, the advection of temperature T in a compressible fluid flow with velocity (u, v) requires a C-grid:

$$\frac{\partial T}{\partial t} = T\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)$$
$$\approx T(x_i, y_j)\left(\frac{u(x_{i+1/2}, y_j) - u(x_{i-1/2}, y_j)}{\Delta x} + \frac{v(x_i, y_{j+1/2}) - v(x_i, y_{j-1/2})}{\Delta y}\right),$$

with Δx and Δy the x and y grid-point distances and Δt the time step. Note that while field u is located on a half point in the x-direction, the derivative $\partial u/\partial x$ is located on a whole point.

Atmospheric models consist mainly of these types of PDEs, and C-grids are most commonly used. Other models such as ocean circulation models comprising incompressible fluid flows require other types of grids, such as B-grids.

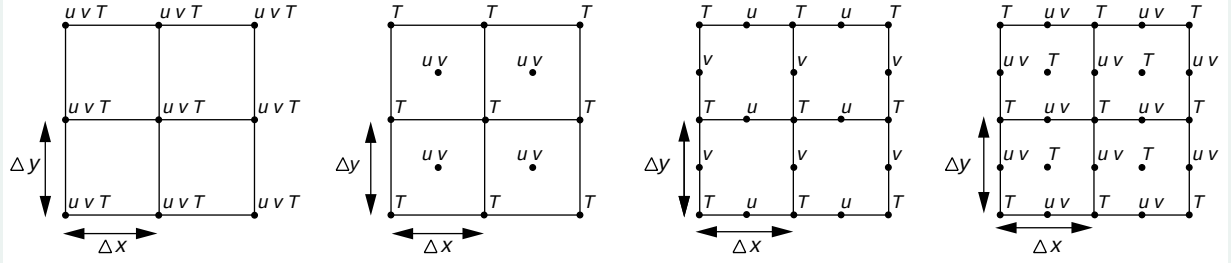

Figure A. Left to right: *A-grid, B-grid, C-grid, and E-grid.*

where $\delta_x^-(u_i) = (u_i - u_{i-1})/\Delta x$,
$\delta_\eta^+(p_k) = p_{k+1} - p_k$, and
$\overline{p_i}^x = (p_i + p_{i+1})/2$.

This last operation performs a point to half-point conversion by linear averaging. The *hxt* and *hxv* fields denote differently staggered versions of the h_x field. Equation 3 was further algebraically optimized using the linearity of finite differences and sums.

The transformation system

The Ctadel system incorporates a symbolic and algebraic simplifier to transform problem specifications into intermediate representations and for applying simplification and optimization on the intermediate problem representations and codes. In this section we briefly discuss some details that a Ctadel user need not fully understand, but they present important ideas on how the transformations take place in the Ctadel system.

For an application developer the algebraic properties of operators used in a model are one of the basic and powerful tools that can be exploited to implement the model efficiently. By rearranging the application order and by combining operators he can reduce the computational complexity and therefore reduce the expected runtime of the application.

For associating algebraic properties to operators we have employed an object-oriented framework, which has the advantage that an operator may have properties inherited from multiple operator classes. An excerpt of the operator class hierarchy is shown in Figure 2.

Elemental operators like the primitive arithmetic functions operate element-wise (grid-point wise) on some domain. A (self-) commuting operator commutes with other operators in this class in the first argument under a "domain orthogonality" constraint. Examples of these are derivatives, finite-difference stencils, quadratures, summations, and (inverse) Fourier transforms applied in different domain directions. We have depicted some examples of object-oriented "messages" for

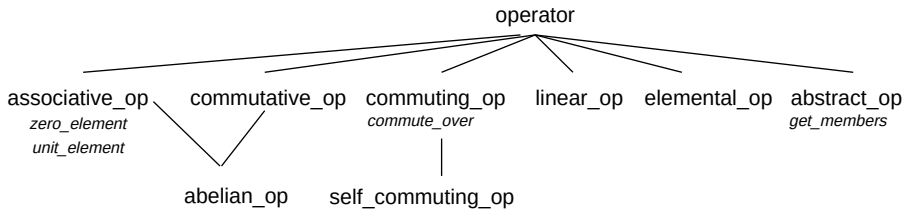

Figure 2. Operator hierarchy.

invoking methods on operator instances: methods for retrieving the zero and unit elements of an associative operator, and a method for retrieving the concrete members of an abstract operator, such as $\delta_x^- \in get\_members(\partial/\partial x)$ that was selected for Equation 3).

The operator classes associative_op, commutative_op, and commuting_op have a special purpose in the transformation system. An algebraic expression is first rendered invariant of associativity, commutativity, and commuting of operators before matching the expression with a left-hand side of a rewrite rule. For example, using the linearity of derivatives and the fact that the derivative operator is part of the self_commuting_op class (that is, it commutes with itself), the transformation system rewrites

$$\frac{\partial^2 u}{\partial x\,\partial y} + a + \frac{\partial v}{\partial x} \quad \text{into} \quad \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y} + v\right) + a.$$

The rewriting is performed by applying just one rewrite rule of the form

```
df(X1, C) + df(X2, C) => df(X1 + X2, C)
```

where `df` is the derivative operator, `X1` and `X2` are templates for expressions, and `C` is a template for a coordinate. Hence, for matching the left-hand side the associativity and commutativity of the "`+`" operator and self-commuting of the `df` operator are *implicitly* applied. This method of implicit rewriting before matching has proved very useful for the development of transformations. The mechanism alleviates any persistent phase-ordering problems inherent in transformation systems that are based on term rewriting.

Eliminating common subexpressions

Finding and eliminating common subexpressions is a key factor in obtaining efficiency. Typically, an application developer tries to optimize the numerical schemes of a model by grouping similar operations together and by introducing temporary variables and tables.

Ctadel applies common subexpression elimination on the level of the numerical schemes within the presented framework of algebraic properties of operators. Operators are commuted and the associative and commutative laws are applied upon matching common subexpressions. Furthermore, common subexpressions in the numerical schemes may have different index offsets, constant indices, and linear index transformations. As an example, assume that the following summations are part of a numerical scheme, with the i, j, and k indices having the ranges shown in the bracketed intervals:

$$f_{i,j,k} = \sum_{j=j+1}^{m} u_{i,j} + u_{i-1,j} + a_{j,k}$$
$$(i = [1,n],\ j = [1,m],\ k = [1,\ell\,]);$$
$$g = \sum_{j=1}^{m}\sum_{i=1}^{n} u_{i,j} + u_{i+1,j} + a_{j,i}\,.$$

We assign a special meaning to the sum $\Sigma_{j=j+1}^{m}$ in Ctadel: it denotes a so-called "prefix sum" operation that sums the values of a variable using a *local* index j, which starts at the value of the *global* index $j + 1$ and goes up to m. Prefix sums are introduced to implement (for example) partial integrations. With this syntactic convention for sums, Ctadel is able to find that the value of array variable f can be used in the computation of variable g, and a new set of assignments is produced:

$$f_{i,j,k} = \sum_{j=j+1}^{m} u_{i,j} + u_{i-1,j} + a_{j,k}$$
$$(i = [1,\ n+1],\ j = [0,m],\ k = [1,\max(n,\ell)]);$$
$$g = \sum_{i=1}^{n} f_{i+1,0,i}\,.$$

Only by close inspection does this transformation reveal its full potential to a user unaware of

its secrets. Note that variable f is now reindexed in the k-index range from 1 to max(n, ℓ). This, however, may be undesirable when the n and ℓ bounds significantly deviate, a fact only the user of the system knows. Ctadel lets the user specify the strength of the common subexpression elimination by setting optional switches that enable or disable the finding of common subexpressions under certain types of index transformations. In this way, the user can easily experiment with different forms of codes derived with different techniques of common-subexpression elimination.

Results

We generated several different architecture-dependent codes for the Hirlam dynamics and ran the codes on an HP 9000/720 system, SGI Indy and Indigo systems, a Convex C4 system (one CPU), a Cray C98 system (with one CPU active), a MasPar MP-1 system (1,024 PEs in a square mesh, SIMD architecture), and a Cray T3D (128 PEs, MIMD architecture with a logically shared but physically distributed memory interconnected in a 3D torus). For all codes, compiler switches for maximum optimization were used. The performance results have been reported elsewhere.[5]

For measuring the relative speedups of the generated codes with respect to handwritten codes, we used three handwritten codes that can be considered as the "best by human." These three versions are the original production Fortran 77 code optimized for vector architectures, a data-parallel Fortran 90 version derived from the original code, and a message-passing parallel version.

Code quality

By comparing the generated Hirlam dynamics codes with the original handwritten production code, we discovered two errors in the original handwritten code. The first error was a programming mistake related to boundaries of array variables; the second was in the numerical scheme of the Hirlam dynamics. The forecast model deals with spherical grids, whereas the model was originally intended to handle more general curvilinear grids. These mistakes did not affect the quality of the forecast but confirmed our point that formal methods, and in particular code generation from a formal specification, have advantages for correctness.

The numerical outputs of the generated codes and production code agreed by at least six decimal digits using 64-bit IEEE floating-point number format. Numerical deviations result from the algebraic transformation of the forecast model by application of the distributive law for the factorization. The deviations are not considered significant for weather predictions because they are within the limits of accuracy.

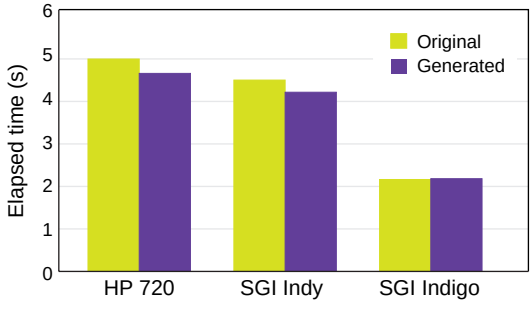

Figure 3. Serial codes performance.

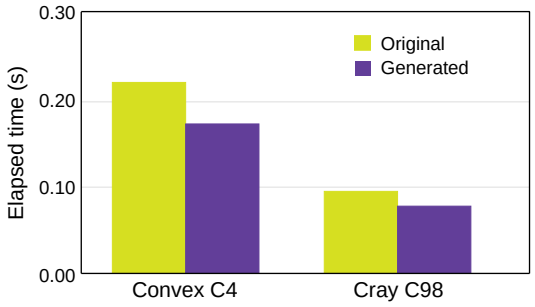

Figure 4. Vector codes performance.

Code performance

We will discuss the performance of serial, vector, SIMD, and MIMD codes separately.

Serial codes. Figure 3 shows the performance of the original handwritten reference and Ctadel-generated Fortran 77 codes for serial architectures with a $110 \times 100 \times 16$ grid, an often-used grid size for the weather predictions of the operational Hirlam system over Europe.

The average speedup of the generated codes for serial computers was 7.3 percent, but at the cost of an increase in memory requirements of about 8 percent depending on the machine and compiler used, even when we manually optimized the generated codes for memory usage.

Vector codes. For vector architectures we generated Fortran 77 code with loops structured in such a way (such as loop distribution) that vector operations can be optimized by the back-end compiler. Figure 4 shows the performance of the original handwritten and Ctadel-generated codes for vector architectures with a $110 \times 100 \times 16$ grid.

On average, we obtained speedups of 13 per-

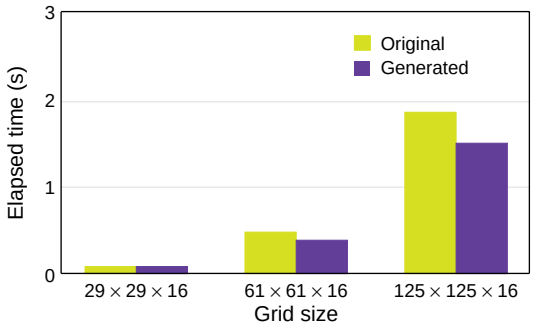

Figure 5. Data-parallel codes performance on a MasPar MP-1.

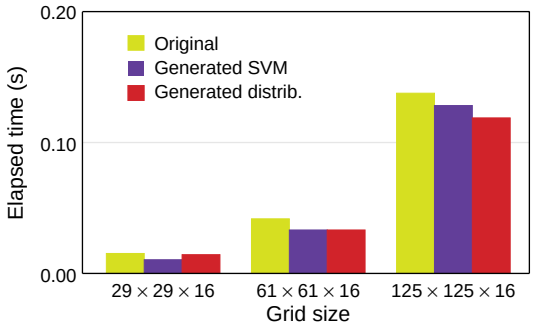

Figure 6. Cray T3D codes performance.

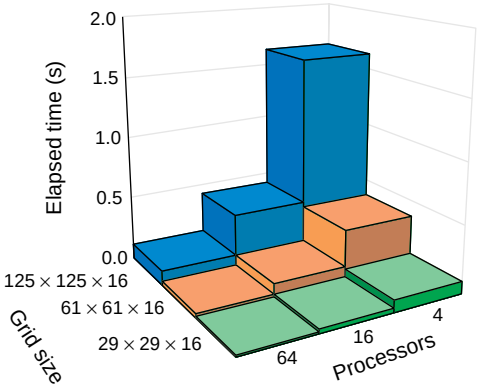

Figure 7. Scalability of the generated SVM codes on a Cray T3D.

cent and 23 percent with the generated codes for a Convex C4 and Cray C98, respectively. In this case some of the generated codes required up to 40 percent more memory due to back-end compiler optimizations that require additional temporary storage.

SIMD codes. For SIMD or data-parallel systems, we generated Fortran 90 code including HPF `FORALL` statements. The `FORALL` statements turned out to be much more efficient on the MasPar MP-1 than Fortran 90 array syntax.

We added directives in both the original handwritten and the generated data-parallel Fortran 90 codes to distribute the two-dimensional horizontal domain using cyclic array distributions. Furthermore, all arrays were aligned in memory by setting the lower bounds of the array declarations equal. Figure 5 shows the performance of the handwritten reference and generated codes on a MasPar MP-1 with 1,024 processors. On the MasPar we measured speedups ranging from 1 percent for small grids ($29 \times 29 \times 16$) up to 24 percent for large grids ($125 \times 125 \times 16$).

MIMD codes. For shared virtual-memory parallel (SVM) computers where each processor is equipped with local memory like the Cray T3D, we generated SPMD Fortran 77 code using a domain-splitting method to split the global domain into subdomains. The input for each code for a subdomain is copied from shared to local memory. By allowing overlapping regions, each subdivided problem can be solved independently using local memory only, thereby avoiding data traffic between shared (nonlocal) and local memory.

We also generated SPMD Fortran 77 code using a communication library based on PVM Version 3, implementing subdomain-splitting for distributed memory parallel computers. Similar to the previous type of code, the subdomains have mutually overlapping regions containing duplicated data from logically neighboring processors.

The performance results with 64 PEs on a Cray T3D are shown in Figure 6 where the original code is a SVM type of code. Figure 7 shows the scalability of the generated SVM codes.

On the Cray T3D we measured a speedup of up to 44 percent. However, the speedup for different configurations of grid sizes and number of processors varied. Furthermore, the time to copy shared arrays to local arrays turned out to be very costly: 13 percent up to 25 percent of the total elapsed time.

The performance results we have reported here show that the generation of efficient codes is feasible within Ctadel's symbolic framework. These symbolic techniques are also applicable to PDE-based application areas other than weather prediction, and we are currently investigating code generation for advanced numerical high-resolution methods in advection–diffusion problems. ♦

Acknowledgment

Support was provided by the Foundation for Computer Science (SION) of the Netherlands Organization for Scientific Research (NWO) under project no. 612-17-120.

References

1. E.N. Lorenz, "Deterministic Nonperiodic Flow," *J. Atmospheric Sciences*, Vol. 20, 1963, pp. 130–141.
2. L.F. Richardson, "Weather Prediction by Numerical Process," Cambridge Univ. Press, London, 1922; reprinted by Dover, 1965.
3. E. Gallopoulos, E. Houstis, and J.R. Rice, "Computer as Thinker/Doer: Problem-Solving Environments for Computational Science," *IEEE Computational Science & Eng.,* Vol. 1, No. 2, Summer 1994, pp. 11–23.
4. G. Cats and L. Wolters, "The Hirlam Project," *IEEE Computational Science & Eng.,* Vol. 3, No. 4, Winter 1996, pp. 4–7.
5. R. van Engelen, L. Wolters, and G. Cats, "Ctadel: A Generator of Multi-Platform High Performance Codes for PDE-based Scientific Applications," *Proc. 10th ACM Int'l Conf. Supercomputing*, ACM Press, New York, 1996, pp. 86–93.
6. R. van Engelen and L. Wolters, "A Comparison of Parallel Programming Paradigms and Data Distributions for a Limited Area Numerical Weather Forecast Routine," *Proc. Ninth ACM Int'l Conf. Supercomputing*, ACM Press, New York, 1995, pp. 357–364.
7. E. Kållén, ed., "HIRLAM Documentation Manual System 2.5," June 1996. Available from SMHI, S–60179 Norrköping, Sweden.
8. W. Hibbard and D. Santek, "The VIS-5D System for Easy Interactive Visualization," *Proc. IEEE Visualization '90*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1990, pp. 129–134.
9. R. van Engelen et al., "Incorporating Application Dependent Information in an Automatic Code Generating Environment," *Proc. 11th ACM Int'l Conf. Supercomputing*, ACM Press, New York, 1997, pp. 180–187.
10. E. Gallopoulos, E. Houstis, and J.R. Rice, *Future Research Directions in Problem Solving Environments for Computational Science: Report of a Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science,* Tech. Report 1259, Center for Supercomputing Research and Development, Univ. of Illinois at Urbana-Champaign, 1992.
11. A. Arakawa and V.R. Lamb, "A Potential Enstrophy and Energy Conserving Scheme for the Shallow Water Equations," *Monthly Weather Rev.,* Vol. 109, 1981, pp. 18–36.

Robert van Engelen is a doctoral candidate in computer science at Leiden University, The Netherlands. His current research interests are high-performance computing, software synthesis, and probabilistic and causal networks. He received an MSc in computer science from Utrecht University. E-mail, robert@cs.leidenuniv.nl; WWW, http://www.wi.leidenuniv.nl/~robert.

Lex Wolters is assistant professor of computer science at Leiden University, The Netherlands. His research interests are large-scale applications in relation to high-performance computing, in particular automatic code generation. He received his PhD in physics from Utrecht University. E-mail, llexx@cs.leidenuniv.nl; WWW, http://www.wi.leidenuniv.nl/~llexx.

Gerard Cats is system manager of the Hirlam project and senior researcher at the Royal Netherlands Meteorological Institute in De Bilt, The Netherlands. His interests are the development and implementation of numerical weather prediction models and all related aspects. He received a MSc in physics from Utrecht University. E-mail, cats@knmi.nl; WWW, http://www.knmi.nl/~cats.

Corresponding author: Robert van Engelen, Leiden University, Department of Computer Science, PO Box 9512, 2300 RA Leiden, The Netherlands.