Published in proceedings of the HIPS'97 Second International Workshop, April 1–5, 1997, Geneva, Switzerland, IEEE Computer Society Press, pp. 105–109.

PDE-Oriented Language Compilation and Optimization with CTADEL for Parallel Computing

Robert van Engelen* & Lex Wolters
High Performance Computing Division
Dept. of Computer Science, Leiden University
PB. 9512, 2300 RA Leiden, The Netherlands
robert@cs.leidenuniv.nl

Gerard Cats
Royal Netherlands Meteorological Institute
PB. 201, 3730 AE De Bilt, The Netherlands
cats@knmi.nl

Abstract

In this paper we focus on generating efficient parallel codes for solving scientific problems within large-scale performance-critical applications. More specifically, we present techniques for high-level symbolic optimization for automatic generation of efficient codes that numerically solve sets of computationally expensive right-hand sides (RHSs) of systems of partial differential equations (PDEs). These type of PDE problems can be found in application areas like numerical weather prediction, ocean circulation modeling, and climate modeling. To solve the PDEs describing the physical problem, high-performance software environments and computer platforms are required. The CTADEL *application driver is a software environment for generating optimized codes for serial, vector, or parallel architectures from a high-level PDE problem description of a weather forecast model. The advantage of the high-level problem description is that all of the high-level information can be exploited for parallelization, restructuring, and optimization from the highest descriptive level to the low-level generated code.*

1. Introduction

Computationally expensive aggregate operations such as integrations, Fourier transforms, and differentiations are frequently used for solving PDEs in application areas of numerical weather prediction and ocean circulation modeling. The PDEs involved in these application areas describe the coupling of large sets of variables, and the RHSs of the PDEs correspondingly consists of arithmetically complex expressions. For example, in the dynamics of the HIRLAM [5] numerical weather forecast system[1], non-linear PDEs describe the dynamical tendencies of the prognostic variables wind, humidity, temperature, and pressure depending on the current state of the atmosphere. The PDEs take the current values of the variables, their spatial and time derivatives and integrated values on the RHS and the time derivatives on the left-hand side (LHS). The PDEs are solved by computing all RHSs first, then by making an explicit time step for the LHSs followed by a so-called semi-implicit correction resulting in solving a Helmholtz equation. In this scheme, the computation of the RHSs requires a significant amount of the total execution time for a HIRLAM production run. One possible way to obtain better execution times is to write parallel code for the dynamics and to run the code on a parallel platform [2]. However, high performance can only be achieved by exploiting specific hardware characteristics of the target (parallel) computer architecture. Writing the efficient codes by hand requires extensive efforts and is error prone. Furthermore, from a maintenance point of view, proposed improvements in the PDE model require even more programming efforts to implement the improvements in all the hardware-specific versions of code.

The CTADEL Code-generation Tool for Applications based on Differential Equations using high-level Language specifications is an application driver [3] for the HIRLAM system. The tool has been designed to generate optimized code for serial, vector, or parallel architectures. Performance results [3] have demonstrated that the generated codes outperform both the hand-written and optimized production code running on vector architectures and the parallel codes obtained using a parallelizing compiler from the production code [2, 6]. The problem of restructuring low-level code is that high-level information on the model is not available while this information is crucial for effectively re-

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

[1]The HIRLAM system is developed by the HIRLAM-project group, a cooperative project of Denmark, Finland, Iceland, Ireland, The Netherlands, Norway, and Sweden.

E_1	E_2	CSE
$\bigotimes_{i=a}^{b} u_i$	$\bigotimes_{i=i+c}^{b} u_i$	$E_1 = E_{2[i\leftarrow a-c]}$
$\bigotimes_{i=a}^{b} u_i$	$\bigotimes_{i=a}^{i+c} u_i$	$E_1 = E_{2[i\leftarrow b-c]}$
$\bigotimes_{i=i+a}^{b} u_i$	$\bigotimes_{i=i+c}^{b} u_i$	$E_1 = E_{2[i\leftarrow i+a-c]}$
$\bigotimes_{i=a}^{i+b} u_i$	$\bigotimes_{i=a}^{i+c} u_i$	$E_1 = E_{2[i\leftarrow i+b-c]}$
$u_a \otimes v_a$	$u_i \otimes v_i$	$E_1 = E_{2[i\leftarrow a]}$
$u_{i+a} \otimes v_{i+a}$	$u_i \otimes v_i$	$E_1 = E_{2[i\leftarrow i+a]}$
$f(u_a)$	$f(u_i)$	$E_1 = E_{2[i\leftarrow a]}$
$f(u_i)$	$f(u_{i+a})$	$E_1 = E_{2[i\leftarrow i+a]}$

E_1	E_2	CSE
$\bigotimes_{j=c}^{d} \bigotimes_{i=a}^{b} u_{i,j}$	$\bigotimes_{i=a}^{b} u_{i,j}$	$E_1 = \bigotimes_{j=c}^{d} E_2$
$\bigoplus_{j=c}^{d} \bigoplus_{i=a}^{b} u_{i,j}$	$\bigoplus_{j=c}^{d} u_{i,j}$	$E_1 = \bigoplus_{i=a}^{b} E_2$
$v_a \oplus u_a$	$u_i \oplus v_i$	$E_1 = E_{2[i\leftarrow a]}$
$v_{i+a} \oplus u_{i+a}$	$u_i \oplus v_i$	$E_1 = E_{2[i\leftarrow i+a]}$
$u_a \oplus w \oplus v_a$	$u_i \oplus v_i$	$E_1 = w \oplus E_{2[i\leftarrow a]}$
$u_{i+a} \oplus w \oplus v_{i+a}$	$u_i \oplus v_i$	$E_1 = w \oplus E_{2[i\leftarrow i+a]}$
$g(h(u,a),b)$	$h(u,a)$	$E_1 = g(E_2, b)$
$g(h(u,a),b)$	$g(u,b)$	$E_1 = h(E_2, a)$

Table 1. Types of common subexpressions recognized between expressions E_1 and E_2.

structuring and optimizing the code for another machine.

In this paper we will briefly introduce CTADEL and describe the basic techniques that it applies for high-level symbolic optimization of reduction and scan operations for efficient serial, vector, and parallel computing. In the sequel, we will restrict the discussion to methods for generating efficient codes for the RHSs of a system of PDEs. Therefore, it is assumed that the PDEs are specified in the form $LHS = RHS$ where each *LHS* constitutes a field or a vector field only. Sets of general PDEs of the form $\mathcal{L}_i\, u_i = RHS_i$ can be solved by computing $LHS_i = RHS_i$ first and then solving $\mathcal{L}_i\, u_i = LHS_i$ for u_i using explicit and semi-implicit methods with appropriate boundary conditions where u_i is some (new) field and $\mathcal{L}_i$ is a differential operator, e.g. $\mathcal{L}_i = \frac{\partial}{\partial t}$ or $\mathcal{L}_i = \nabla^2$. Using implicit methods, the coefficients of the assumed linear RHS of the equation define a new set of 'simple' equations $c_j = RHS_j$, each for the j^{th} coefficient c_j. These equations are optimized as described and the resulting symbolic expressions are used for the generation of a solver for the resulting set of linear equations. The selection and generation of code for such a solver can be performed using PSEs like (parallel) ELLPACK [4] or ALPAL [1].

2. Compilation and Optimization with CTADEL

The CTADEL system uses symbolic techniques to derive codes that are comparable in efficiency to hand-written and optimized codes. Firstly, the system incorporates a symbolic and algebraic computing system (SAC) for high-level symbolic computing purposes such as to optimize equations by exploiting properties of operators, e.g. the linearity of difference and quadrature operators used in the PDEs. Secondly, a global common-subexpression eliminator is used to reduce the computational complexity on a global scale by taking the hardware characteristics into account, e.g. the relative cost of memory operations versus arithmetic operations. Thirdly, the system uses restructuring compiler techniques on the low-level generated code to improve memory usage and for vectorization and parallelization.

2.1. Algebraic Simplification: GPAS

All high-level to low-level code transformations are performed using the GPAS General-Purpose Algebraic Simplifier by application of rewrite rules. In GPAS, abstract properties of operators are encapsulated in an object-oriented hierarchy defining several operator classes. The rewrite rules are applied by taking the properties of the operators into account. This allows new operators with specific properties to be easily added to the system. For example, new FFT operators can be declared as instances of the linear operator class and self-commuting operator class[2]. Hence, expressions using the new operator can be simplified by exploiting the linear and commutativity properties of the operator. Other (commercial) SACs such as MAPLE, MATHEMATICA, or REDUCE lack such facilities. Furthermore, the simplification of expressions by these SACs do not always result in expressions that are acceptable for generating efficient codes by the code generator. The reason is that the simplification performed by most SACs is not fully aimed at reducing arithmetic complexity and optimizing for parallel and distributed computation.

2.2. Common Subexpression Elimination: DICE

While GPAS optimizes equations on a local scale, the DICE Domain-shift Invariant Common-subexpression Eliminator removes subexpressions on a global scale. That is, sets of equations are optimized by removing redundant computations. The type of common subexpressions (CSEs) recognized by DICE are shown in Table 1. In Table 1, E_1 and E_2 denote two subexpressions, $\otimes$ denotes a dyadic associative operator, $\oplus$ denotes a dyadic associative and commutative operator (e.g. $\oplus = +$ and $\bigoplus = \sum$), f is a elemental function, and g and h denote two explicitly declared

[2]Self-commutativity of a dyadic operator f means that we have the identity $f(f(u,x),y) = f(f(u,y),x)$.

(self-)commuting operators. Note that some of the CSEs require an index substitution, denoted as $[i \leftarrow a]$ where the i-index is to be replaced with expression a. The notation $\bigotimes_{i=i+c}^{b}$ (where e.g. $\bigotimes = \sum$) denotes the use of *two* distinct i-indexes: a *local* i-index for the aggregate operation (e.g. summation) from the current value of the *global* i-index in the outer context of the operation increased by c, ranging up to b. That is, $\bigotimes_{i=i+c}^{b}$ represents an exclusive scan or reversed prefix operation. By introducing this notational convention, reduction and scan operations can be easily distinguished and optimized accordingly.

The presented list of CSEs is not exhaustive. In addition to the CSEs shown in Table 1, other CSEs can be found by using any combinations of the listed 'primitive' CSEs. For example, let $E_1 = \text{fft}(\text{fft}(u_{i,j,k}, i = 1\,..\,n), j = 1\,..\,m)$ and $E_2 = \text{fft}(u_{i,j,k+1}, j = 1\,..\,m)$ be two (sub)expressions. Assume that the fft-operator is declared as an instance of the self-commuting operator class. Then, with DICE we obtain $E_1 = \text{fft}(E_{2[k\leftarrow k-1]}, i = 1\,..\,n)$ thereby saving an FFT operation. Upon removing CSEs, storage is allocated for the CSEs in the form of temporary variables.

3. Reduction and Scan Optimization

The optimization of reduction and scan operations for efficient serial and parallel computing is an example of a combined use of symbolic algebra techniques for communication optimization. The optimization techniques are illustrated by means of a problem example. The examples are simplified problems derived from the documentation of the HIRLAM weather forecast model [5] and are illustrative for the type of problems solved by CTADEL. Consider

$$p = \int_0^1 \int_y^1 \frac{\partial u}{\partial x}\, dy\, dz \qquad \forall (x,y) \in \Omega_{x,y} \tag{1}$$

where $p(x,y)$ and $u(x,y,z)$ are dependent variables, x, y, z are the independent variables on the domain $\Omega = [0,1]^3$. Discretization of Eq. (1) using finite differences yields

$$p_{i,j} = \sum_{k=1}^{\ell} \sum_{j=j+1}^{m} \frac{1}{h}\,(u_{i+1,j,k} - u_{i,j,k}) \qquad \forall (i,j) \in D(p) \tag{2}$$

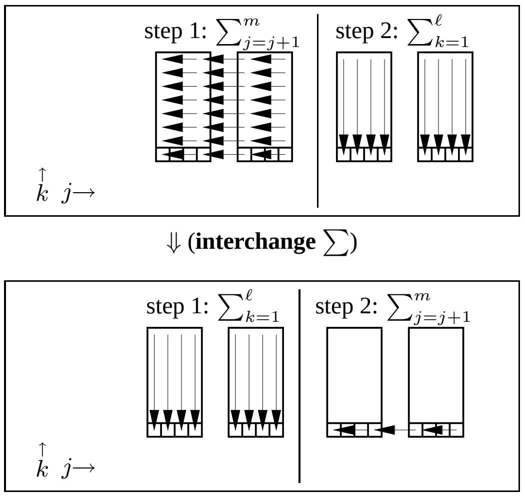
where $D(p)$ is the discretized domain or grid of p and the u and p fields have effectively become functions of the discrete (i,j,k)-grid with domain $[1,n]\times[1,m]\times[1,\ell] \subseteq \mathbb{Z}^3$. Here, the double integration is replaced with a double summation using the midpoint quadrature formula, and the partial derivative is replaced by a difference quotient assuming grid-point distance h in the x-direction.

Eq. (2) can be simplified using an algebraic simplifier. For algebraic simplification in CTADEL, we use the built-in GPAS algebraic simplifier, giving

$$p_{i,j} = \frac{1}{h} \sum_{k=1}^{\ell} \sum_{j=j+1}^{m} (u_{i+1,j,k} - u_{i,j,k}) \tag{3}$$

Note that the result is just slightly different from Eq. (2) while at least ℓ multiplications are saved.

Although the expression is algebraically simpler, the RHS of Eq. (3) is still not optimal for generating parallel code. Assume that the (i,j,k)-grid domain of the problem is block-wise distributed in the j-direction. Then, excessive communication results because the parallel scan operation is executed before the serial reduction. When the summations are interchanged, which is an algebraically valid transformation, the data volume communicated in the parallel scan will be significantly reduced, see Fig. 1.



step 1: $\sum_{j=j+1}^{m}$
step 2: $\sum_{k=1}^{\ell}$
$\uparrow k$ $j\rightarrow$
$\Downarrow$ (interchange $\sum$)
step 1: $\sum_{k=1}^{\ell}$
step 2: $\sum_{j=j+1}^{m}$
$\uparrow k$ $j\rightarrow$

Figure 1. Reducing communication between two processors *P0* and *P1* by interchanging the order of the summations.

The interchange of the summations is automatically performed by GPAS while in other SACs this can only be accomplished by writing ad-hoc procedures. To this end, GPAS uses the abstract notion of commutativity defined by the class of commuting operators as briefly mentioned in Section 2.1. More specifically, two operator instances of this class commute if and only if an explicit commutativity relationship between the operators is defined by the system or (re)defined by the user. The commutativity relationships induce a default functional composition order on the operator instances of the class of commuting operators. This way, the application order can be controlled while still allowing the functional composition of the operators to be interchanged which is necessary for application of rewrite rules for optimization and for finding CSEs. Commutativity is allowed only in the absence of a cross-iteration dependency between two aggregate operations, e.g. the sums $\sum_{j=1}^{m}\sum_{i=j}^{n}$ cannot be interchanged without changing the semantics (however, using e.g. Fourier-Motzkin elimination, the iteration space of the summation can be reshaped giving $\sum_{i=1}^{n}\sum_{j=1}^{i}$). The default order for the functional composition of scans and reductions is the application of the reduction first. We will further refer to this transformation as 'scan over reduction optimization' when applied.

3.1. Extending the Example Model

Many scientific models, like the HIRLAM dynamics, consists of sets of PDEs. The RHSs of the PDEs often contain CSEs which can be removed by introducing new equations for the subexpressions found. This is performed after discretization of the PDEs, because partial derivatives need to be replaced first with e.g. different difference quotients depending on the type of grid on which they operate. In general, the difference quotients result in expressions comprising variables which are offset in their index. Therefore, for effectively removing CSEs, similar expressions which are shifted with respect to each other on the computational domain should be recognized as a CSE. As an example, reconsider the example model consisting of Eq. (1). Assume that the model is extended by adding the equation

$$q = \int_0^1 g\,\frac{\partial u}{\partial x}\,dy\bigg|_{z=1} \qquad \forall x \in \Omega_x \tag{4}$$

where $g(x)$ is a dependent variable and $\Omega_x = [0,1]$. Using a backward difference, the discretization of Eq. (4) results in

$$q_i = \frac{g_i}{h}\sum_{j=1}^{m}(u_{i,j,\ell} - u_{i-1,j,\ell}) \qquad \forall i \in D(q) \tag{5}$$

where $D(q)$ denotes a discretized grid domain of q. By close inspection, it is clear that subexpressions are shared between Eqs. (3) and (5). An optimal strategy for finding CSEs is by trying all factorized forms possible of the discrete equations and choosing the form in which the largest (number) of CSEs is obtained. This however, is computationally infeasible. Instead, the strategy employed by CTADEL is to check the algebraically expanded form and factorized form for CSEs. The discrete Eqs. (3) and (5) are translated into the expanded form first

$$\begin{aligned} p_{i,j} &= \frac{1}{h}\sum_{j=j+1}^{m}\sum_{k=1}^{\ell}u_{i+1,j,k} - \frac{1}{h}\sum_{j=j+1}^{m}\sum_{k=1}^{\ell}u_{i,j,k} \\ q_i &= \frac{g_i}{h}\sum_{j=1}^{m}u_{i,j,\ell} - \frac{g_i}{h}\sum_{j=1}^{m}u_{i-1,j,\ell} \end{aligned} \tag{6}$$

Then, the result is checked for CSEs. For each CSE found, a new variable is introduced

$$\begin{aligned} &s_{i,j,k} = \sum_{j=j+1}^{m}u_{i,j,k} \quad ; \quad t_{i,j} = \frac{1}{h}\sum_{k=1}^{\ell}s_{i,j,k} \\ &p_{i,j} = t_{i+1,j} - t_{i,j} \quad ; \quad q_i = \frac{g_i}{h}\left(s_{i,0,\ell} - s_{i-1,0,\ell}\right) \end{aligned} \tag{7}$$

where s and t are new temporary variables. The result is not trivial: observe that DICE 'discovers' that the total sum in the equation for q is redundant and therefore replaces the sum with the result of the partial sum computed in the RHS of Eq. (3) stored in variable s. That is, the result of the scan operation is reused for a reduction thereby completely avoiding the need for computing the reduction. We will further refer to this transformation as 'scan-reduction reuse'.

The CSEs requiring a large amount of memory while yielding a small cost reduction in the amount of computations, are substituted back into the expressions where the CSEs were found. Evaluating the complexity changes of introducing a CSE are based on heuristics describing the relative cost of load, store, and arithmetic operations of the hardware of the target computer architecture. Using these machine characteristics, it is decided if the CSE is worthwhile to maintain or to be replaced back. Since a placement back into an expression also changes the complexity of the affected expression, an iterative scheme is applied for the back-placements until a fixed point is reached [3].

In the example problem, the CSE-variable s is too expensive to maintain and is placed back. Then, local CSEs are removed for each equation, giving

$$\begin{aligned} &s_i = \sum_{j=1}^{m}u_{i,j,\ell} \quad ; \quad t_{i,j} = \frac{1}{h}\sum_{k=1}^{\ell}\sum_{j=j+1}^{m}u_{i,j,k} \\ &p_{i,j} = t_{i+1,j} - t_{i,j} \quad ; \quad q_i = \frac{g_i}{h}\left(s_i - s_{i-1}\right) \end{aligned} \tag{8}$$

After scan over reduction optimization, we finally obtain

$$\begin{aligned} &s_i = \sum_{j=1}^{m}u_{i,j,\ell} \quad ; \quad t_{i,j} = \frac{1}{h}\sum_{j=j+1}^{m}\sum_{k=1}^{\ell}u_{i,j,k} \\ &p_{i,j} = t_{i+1,j} - t_{i,j} \quad ; \quad q_i = \frac{g_i}{h}\left(s_i - s_{i-1}\right) \end{aligned} \tag{9}$$

3.2. Results

We have evaluated the performance of three codes corresponding to Eqs. (7), (8), and (9) of the example PDE problem to measure the effect of the transformations on performance. Fig. 2 depicts the main parts of three programs generated in CRAY Fortran with HPF '`FORALL`' statements to represent do-all loops. The vectorization and parallelization were performed at the statement-level by CTADEL.

```
      PROGRAM P1
      REAL u(0:n+1,m,l),g(0:n+1),h,p(0:n+1,m),q(0:n+1)
      REAL s(0:n+1,0:m,l),t(0:n+1,m)
      ...
      DO 2330 j = m,1,-1
       FORALL(i=0:n+1,k=1:l) s(i,j-1,k)=s(i,j,k)+u(i,j,k)
2330  CONTINUE
CMIC$ PARALLEL ...
CMIC$ CASE
      DO 2340 k = 1,l
       FORALL(i=1:n+1,j=1:m) t(i,j)=s(i,j,k)+t(i,j)
2340  CONTINUE
CMIC$ CASE
      FORALL(i=1:n) q(i)=g(i)*(s(i,0,l)-s(i-1,0,l))/h
CMIC$ END CASE
CMIC$ END PARALLEL
      FORALL(i=1:n+1,j=1:m) t(i,j)=t(i,j)/h
      FORALL(i=1:n,j=1:m) p(i,j)=t(i+1,j)-t(i,j)
```

```
      PROGRAM P2
      REAL u(0:n+1,m,l),g(0:n+1),h,p(0:n+1,m),q(0:n+1)
      REAL s(0:n+1),t(0:n+1,m),T1(1:n+1,m)
      ...
CMIC$ PARALLEL ...
CMIC$ CASE
      DO 2270 j = 1,m
       FORALL(i=0:n+1) s(i)=s(i)+u(i,j,l)
2270  CONTINUE
CMIC$ CASE
      DO 2280 k = 1,l
       FORALL(i=1:n+1) T1(i,m)=0
       DO 2290 j = m,2,-1
        FORALL(i=1:n+1) T1(i,j-1)=T1(i,j)+u(i,j,k)
2290   CONTINUE
       FORALL(i=1:n+1,j=1:m) t(i,j)=t(i,j)+T1(i,j)
2280  CONTINUE
CMIC$ END CASE
CMIC$ END PARALLEL
CMIC$ PARALLEL ...
CMIC$ CASE
      FORALL(i=1:n) q(i)=g(i)*(s(i)-s(i-1))/h
CMIC$ CASE
      FORALL(i=1:n+1,j=1:m) t(i,j)=t(i,j)/h
CMIC$ END CASE
CMIC$ END PARALLEL
      FORALL(i=1:n,j=1:m) p(i,j)=t(i+1,j)-t(i,j)
```

```
      PROGRAM P3
      REAL u(0:n+1,m,l),g(0:n+1),h,p(0:n+1,m),q(0:n+1)
      REAL s(0:n+1),t(0:n+1,m)
      ...
CMIC$ PARALLEL ...
CMIC$ CASE
      DO 2300 j = 1,m
       FORALL(i=0:n+1) s(i)=s(i)+u(i,j,l)
2300  CONTINUE
CMIC$ CASE
      DO 2310 j = m,2,-1
       FORALL(i=1:n+1) t(i,j-1)=t(i,j)
       DO 2320 k = 1,l
        FORALL(i=1:n+1) t(i,j-1)=t(i,j-1)+u(i,j,k)
2320   CONTINUE
2310  CONTINUE
CMIC$ END CASE
CMIC$ END PARALLEL
CMIC$ PARALLEL ...
CMIC$ CASE
      FORALL(i=1:n) q(i)=g(i)*(s(i)-s(i-1))/h
CMIC$ CASE
      FORALL(i=1:n+1,j=1:m) t(i,j)=t(i,j)/h
CMIC$ END CASE
CMIC$ END PARALLEL
      FORALL(i=1:n,j=1:m) p(i,j)=t(i+1,j)-t(i,j)
```

Figure 2. Three alternative programs, P1, P2, and P3, for solving the example problem.

Program P1 corresponds to Eq. (7), P2 to Eq. (8), and P3 to Eq. (9). Note that the type of source-code level transformations required to perform analogous transformations between the programs shown in Fig. 2 requires a very powerful restructuring compiler capable of recognizing reduction and scan operations and capable of data-structure transformations for reshaping array variables (array `s` in Fig. 2).

With $n = 254$, $m = 255$, and $\ell = 20$, performance results on a HP 712/60, SGI Indy, CRAY-C98, and a MasPar-MP1 (SIMD, 1024 PEs) are shown in Table 2.

On the HP 712, page swapping significantly degrades performance of program P1; P1 requires the largest amount of memory. On CRAY-C98 with 2 CPUs, the parallel sections are executed in parallel, but unfortunately, overhead prohibits performance gain with respect to 1 CPU. For

	P1	P2	P3
HP 712/60	2.30	0.502	0.361
SGI Indy	0.622	0.553	0.362
CRAY-C98, 1 CPU	0.011	0.012	0.006
2 CPUs	0.013	0.014	0.008
MasPar-MP1, i-distrib.	0.523	0.340	0.179
j-distrib.	237.	243.	164.

Table 2. Total elapsed time (sec).

the MasPar code with the j-direction distributed, excessive communication between the front-end and the DPU processor mesh results due to the fact that the j-reduction/scan are not parallelized. The performance is disappointingly poor, especially for P1 and P2 in which the scan over reduction optimization is prohibited by scan-reduction reuse (P1) or not applied (P2). Program P3 has the best performance compared to programs P1 and P2 on all platforms investigated; P3 is also the program generated by CTADEL.

4. Conclusions

In this paper we have briefly described methods for generating efficient codes for PDE-based problems. The presented techniques have been used for the generation of efficient codes for the HIRLAM weather forecast system using a prototype version of the CTADEL application driver. The optimization of multiple reductions and scans, however, is a new approach. Since parallel reductions and scans require expensive collective communications, the presented approach may yield a significant speedup of the generated code. This is mainly due to the use of a high-level problem description in which all of the high-level information is readily available for exploitation by the algebraic simplifier and common-subexpression eliminator within CTADEL. This information may be difficult to retrieve or may even be lost in the low-level architecture-specific code, thereby hampering the effective use of a restructuring compiler to obtain similar results.

References

[1] G.O. Cook, Jr. and J.F. Painter. ALPAL: A Tool to Generate Simulation Codes from Natural Descriptions, *Expert Systems for Scientific Computing*, E.N. Houstis, J.R. Rice, and R. Vichnevetsky (eds), Elsevier, 1992.

[2] R. van Engelen and L. Wolters. A Comparison of Parallel Programming Paradigms and Data Distributions for a Limited Area Numerical Weather Forecast Routine, *proc. of the* 9th *ACM Int'l Conf. on Supercomp.*:357–364, ACM Press, New York, July 1995.

[3] R. van Engelen, L. Wolters, and G. Cats. Ctadel: A Generator of Multi-Platform High Performance Codes for PDE-based Scientific Applications, *proc. of the* 10th *ACM Int'l Conf. on Supercomp.*:86–93, ACM Press, New York, May 1996.

[4] E.N. Houstis *et al.* //ELLPACK: A Numerical Simulation Programming Environment for Parallel MIMD Machines, *proc. of the* $4^{\rm th}$ *ACM Int'l Conf. on Supercomp.*:96–107, ACM Press, New York, 1990.

[5] E. Kållén (editor), *HIRLAM Documentation Manual System 2.5*, SMHI, S–60179 Norrköping, Sweden, June 1996.

[6] L. Wolters, G. Cats, and N. Gustafsson. Data-Parallel Numerical Weather Forecasting, *Scientific Prog.* 4:141–153, 1995.