A systematic design of parallel program for Dirichlet convolution

A Systematic Design of a Parallel Program for Dirichlet Convolution

by

Pieter Struik

89/7

May, 1989

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

A Systematic Design of a Parallel Program for Dirichlet Convolution

Pieter Struik
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

May 11, 1989

0 Introduction

In this paper we design a parallel program for computing the Dirichlet Convolution of two arithmetical functions. We believe that the program derivation we present is easy to understand and is, thereby, a nice example of our method for designing parallel programs. Programs are derived from their formal specification in a calculational manner. Correctness by design is our main objective, whereas other methods afterwards require a verification of the constructed algorithm.

The problem of designing a parallel program for Dirichlet Convolution was originally posed by Tom Verhoeff in [5]. Solutions for computing the Dirichlet Convolution can be found in [0] and [3]. These solutions resemble our solution; the program derivations, however, are completely different.

The program we derive is a program with fine-grained parallelism. We do not discuss methods to enlarge the grain-size. Our main goal is to show that our design method can be applied to non-trivial examples as easily as to simple examples.

This paper is organized as follows. In section 1 the Dirichlet Convolution is defined and a parallel program is derived for computing the Dirichlet Convolution of two arbitrary arithmetical functions. Section 2 deals with the inverse convolution problem. A parallel program for the inverse convolution problem turns out to be almost identical to the program for Dirichlet Convolution. The Möbius function is an instance of the inverse convolution problem. A parallel program for this function is discussed in section 3. Our program for the Möbius function differs from Tom Verhoeff's program [5].

The notation we use has been adopted from [4].

1 Dirichlet Convolution

In this section we give a definition of Dirichlet Convolution. Next, we generalize this definition and obtain an expression for which we derive a recurrence relation. The program we derive

consists of a network of cells that communicate with each other by message passing over unidirectional channels. By applying the above mentioned recurrence relation, we derive relations for the individual communications along the channels. These relations impose requirements upon the communication behavior of the cells. After finding a communication behavior that satisfies these requirements and that introduces minimal buffering, we present the program text. A short complexity analysis of the program concludes this section.

For an introduction to the theory of arithmetical functions we refer to [2]. We consider arithmetical functions to be functions defined on the positive integers and that have the integers as their range. The Dirichlet Convolution of two arithmetical functions F and G, denoted by $F \star G$, is defined as

$$(F \star G)(n) = (\mathbf{S}\, p,q : p * q = n \ \wedge\ 1 \leq p \ \wedge\ 1 \leq q : F(p) * G(q))$$

for $n \geq 1$.
In this definition, the summation ranges over a non-empty domain that is symmetric in p and q. In the program derivation, we shall maintain this symmetry. We do so because other problems (e.g. dynamic programming [1]) show that destroying symmetry often leads to inefficient programs.

For the derivation of our program we prefer a slightly different (but equivalent) definition of $F \star G$:

$$(F \star G)(n) = (\mathbf{S}\, p,q : p * q = n \ \wedge\ (\sqrt{n} \leq p \ \vee\ \sqrt{n} \leq q) : F(p) * G(q))$$

We generalize this expression by introducing an additional variable. For $0 \leq m \leq n$, expression $Q(m,n)$ is defined as

$$Q(m,n) = (\mathbf{S}\, p,q : p * q = n \ \wedge\ (\sqrt{n} \leq p \leq m \ \vee\ \sqrt{n} \leq q \leq m) : F(p) * G(q))$$

Notice that expression $Q(m,n)$ is defined in the context of arithmetical functions F and G.

Taking $m = n$, we then have $Q(n,n) = (F \star G)(n)$. Hence, computing the Dirichlet Convolution of two arithmetical functions can be done by evaluating expression Q. We now derive a recurrence relation for $Q(m,n)$, since evaluating $Q(n,n)$ involves evaluation of partial sums $Q(m,n)$.

For $0 \leq m < \sqrt{n}$

$$Q(m,n) = 0$$

and for $\sqrt{n} \leq m+1 \leq n$, we derive

$$\begin{array}{ll} & Q(m+1,n) \\ = & \quad \{\text{def. } Q\} \\ & (\mathbf{S}\, p,q : p * q = n \ \wedge\ (\sqrt{n} \leq p \leq m+1 \ \vee\ \sqrt{n} \leq q \leq m+1) : F(p) * G(q)) \\ = & \quad \{\sqrt{n} \leq m+1\} \\ & (\mathbf{S}\, p,q : p*q = n \wedge (\sqrt{n} \leq p \leq m \vee \sqrt{n} \leq q \leq m \vee p = m+1 \vee q = m+1) : F(p)*G(q)) \\ = & \quad \{\text{domain split}\} \end{array}$$

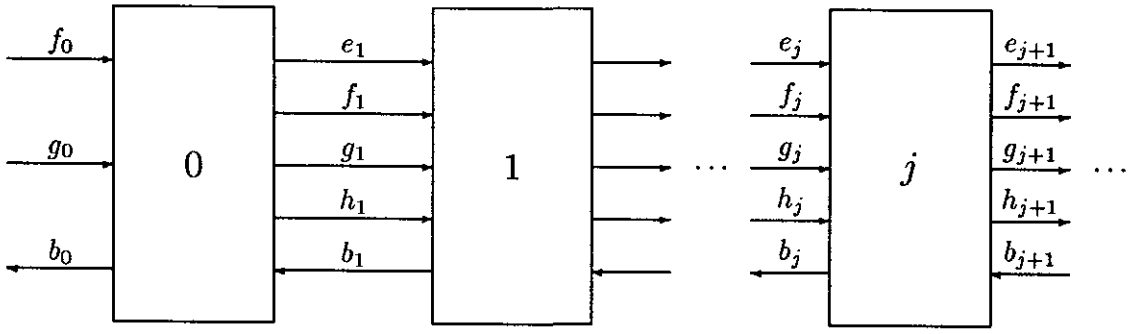

Figure 0: linear network of cells

$$
\begin{array}{ll}
 & (\mathbf{S}\, p,q : p*q = n \ \wedge\ (\sqrt{n} \le p \le m \ \vee\ \sqrt{n} \le q \le m) : F(p) * G(q)) \\
 & +(\mathbf{S}\, p,q : p*q = n \ \wedge\ p = m+1 \ \wedge\ q = m+1 : F(p) * G(q)) \\
 & +(\mathbf{S}\, p,q : p*q = n \ \wedge\ p \neq q \ \wedge\ (p = m+1 \ \vee\ q = m+1) : F(p) * G(q)) \\
= & \quad \{\, \text{def. } Q \,\} \\
 & Q(m,n) \\
 & +\ \mathbf{if}\ \ (m+1)^2 = n \qquad\qquad\qquad \rightarrow\ F(m+1) * G(m+1) \\
 & \quad [\!]\ \ (m+1)^2 \neq n \ \wedge\ (m+1 \mid n) \ \rightarrow\ F(m+1) * G(n/(m+1)) \\
 & \qquad\qquad\qquad\qquad\qquad\quad +\ F(n/(m+1)) * G(m+1) \\
 & \quad [\!]\ \ \neg(m+1 \mid n) \qquad\qquad \rightarrow\ 0 \\
 & \quad \mathbf{fi}
\end{array}
$$

where $k \mid n$ denotes k divides n, i.e. $n \ \mathbf{mod}\ k = 0$.

We rewrite this recurrence relation for $Q(m,n)$

$$
\begin{array}{rcl}
Q(0,n) & = & 0 \\
Q(m+1,n) & = & Q(m,n) \\
 & & +\ \mathbf{if}\ \ (m+1)^2 < n \ \vee\ \neg(m+1 \mid n) \ \rightarrow\ 0 \\
 & & \quad [\!]\ \ (m+1)^2 = n \qquad\qquad\qquad\quad \rightarrow\ F(m+1) * G(m+1) \\
 & & \quad [\!]\ \ (m+1)^2 > n \ \wedge\ \ (m+1 \mid n) \ \rightarrow\ F(m+1) * G(n/(m+1)) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad +\ F(n/(m+1)) * G(m+1) \\
 & & \quad \mathbf{fi}
\end{array}
$$

for $0 \le m < n$.

We now have a recurrence relation for $Q(m,n)$ which we use in the program derivation that follows.

The program we derive consists of a linear network of cells (see fig. 0). Cell 0 is fed with two arithmetical functions along two input channels f_0 and g_0:

$$
\begin{array}{rcl}
f_0(i) & = & F(i+1) \\
g_0(i) & = & G(i+1)
\end{array}
$$

for $i \ge 0$.
Cell 0 also communicates with the environment by means of output channel b_0, which satisfies

$$b_0(i) = (F \star G)(i+1)$$

for $i \geq 0$.

Given this definition, the first communication along channel b_0 satisfies

$$\begin{array}{ll} & b_0(0) \\ = & \quad \{\, \text{def. } b_0 \,\} \\ & (F \star G)(1) \\ = & \quad \{\, \text{def. } F \star G \,\} \\ & F(1) * G(1) \\ = & \quad \{\, \text{def. } f_0 \text{ and } g_0 \,\} \\ & f_0(0) * g_0(0) \end{array}$$

and for $i \geq 0$ we have

$$\begin{array}{ll} & b_0(i+1) \\ = & \quad \{\, \text{def. } b_0 \,\} \\ & (F \star G)((i+1)+1) \\ = & \quad \{\, \text{def. } Q \,\} \\ & Q((i+1)+1, (i+1)+1) \\ = & \quad \{\, \text{recurrence relation for } Q\text{; using } (i+2)^2 > i+2 \text{ and } i+2 \mid i+2 \,\} \\ & Q(i+1, (i+1)+1) + F((i+1)+1) * G(1) + F(1) * G((i+1)+1) \\ = & \quad \{\, \text{def. } f_0 \text{ and } g_0 \,\} \\ & Q(i+1, (i+1)+1) + f_0(i+1) * g_0(0) + f_0(0) * g_0(i+1) \end{array}$$

On account of this expression, we decide that cell 1 computes $Q(i+1,(i+1)+1)$ and sends the result to cell 0 along channel b_1. Generalizing, output channel b_j of cell j ($j \geq 1$) satisfies

$$b_j(i) = Q(i+1, i+1+j)$$

for $i \geq 0$. Notice that this relation also holds for $j = 0$. On account of this observation, we expect that, later on, matching the communication behaviors of cell 0 and cell 1 will not cause any problem.

Summarizing, the values communicated along channel b_0 satisfy

$$b_0(0) = f_0(0) * g_0(0) \tag{0}$$

$$b_0(i+1) = b_1(i) + f_0(i+1) * g_0(0) + f_0(0) * g_0(i+1) \tag{1}$$

From now on we consider cell j for $j \geq 1$.

The first communication along channel b_j satisfies

$$\begin{array}{ll} & b_j(0) \\ = & \quad \{\, \text{def. } b_j \,\} \\ & Q(1, 1+j) \end{array}$$

$$
\begin{array}{ll}
= & \{\text{recurrence relation for } Q;\ Q(0,1+j)=0;\ 1^2 < j+1\} \\
\quad 0 &
\end{array}
$$

and for $i \geq 0$ we have

$$
\begin{array}{ll}
 & b_j(i+1) \\
= & \quad \{\text{def. } b_j\} \\
 & Q((i+1)+1,(i+1)+1+j) \\
= & \quad \{\text{recurrence relation for } Q;\ (i+2 \mid i+j+2) \equiv (i+2 \mid j)\} \\
 & Q(i+1,i+1+(j+1)) \\
 & + \textbf{ if } \ (i+2)^2 < i+j+2 \ \vee \ \neg(i+2 \mid j) \ \rightarrow \ 0 \\
 & \quad [\!] \ (i+2)^2 = i+j+2 \qquad\qquad\quad \rightarrow \ F(i+2) * G(i+2) \\
 & \quad [\!] \ (i+2)^2 > i+j+2 \ \wedge \ (i+2 \mid j) \ \rightarrow \ F(i+2) * G((i+j+2)/(i+2)) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\quad +F((i+j+2)/(i+2)) * G(i+2) \\
 & \quad \textbf{fi}
\end{array}
$$

Thus, for the $(i+1)$-th communication along channel b_j cell j should have at its disposal the values of: $Q(i+1,i+1+(j+1))$, $F(i+2)$, $G(i+2)$, $F(1+j/(i+2))$, and $G(1+j/(i+2))$. On account of the definition of channel b_j, the i-th communication along channel b_{j+1} equals $Q(i+1,i+1+(j+1))$.

For $F(1+j/(i+2))$, $F(i+2)$, $G(i+2)$, and $G(1+j/(i+2))$ we introduce four input channels for cell j: respectively e_j, f_j, g_j, and h_j. Just like the definition of input channels f_0 and g_0, we define

$$
\begin{array}{lcl}
f_j(i) & = & F(i+1) \\
g_j(i) & = & G(i+1)
\end{array}
$$

for $i \geq 0$.
Although, according to above derivation for $b_j(i+1)$, $f_j(i)$ (and $g_j(i)$ similarly) need only to be specified for indices i satisfying $(i+1)^2 \geq i+j+1$, we have specified $f_j(i)$ for all natural i. For the specification of channel e_j and h_j, however, we are more liberal, viz.

$$
\begin{array}{lcl}
e_j(i) & = & F(1+j \ \textbf{div} \ (i+1)) \\
h_j(i) & = & G(1+j \ \textbf{div} \ (i+1))
\end{array}
$$

for all natural i satisfying $(i+1)^2 > (i+j+1)$.
Actually, we have restricted ourselves a little, since $e_j(i)$ (and $h_j(i)$ similarly) need only be specified for indices i that also satisfy $(i+1 \mid j)$. In the sequel, we explicitly use the fact that $e_j(i)$ is specified only for i satisfying $(i+1)^2 > (i+j+1)$.

Now, communications along channel b_j are implemented by

$$
\begin{array}{rcll}
b_j(0) & = & 0 & (2) \\
b_j(i+1) & = & b_{j+1}(i) & \\
 & & + \textbf{ if } \ (i+2)^2 < i+j+2 \ \vee \ \neg(i+2 \mid j) \ \rightarrow \ 0 & (3) \\
 & & \quad [\!] \ (i+2)^2 = i+j+2 \qquad\qquad\quad \rightarrow \ f_j(i+1) * g_j(i+1) & \\
 & & \quad [\!] \ (i+2)^2 > i+j+2 \ \wedge \ (i+2 \mid j) \ \rightarrow \ f_j(i+1) * h_j(i+1) & \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad +e_j(i+1) * g_j(i+1) & \\
 & & \quad \textbf{fi} &
\end{array}
$$

Next, we turn our attention to the implementation of input channels e_j, f_j, g_j, and h_j. On behalf of the symmetry between channels e_j and h_j, and between channels f_j and g_j, we only discuss the implementation of channels e_j and f_j.

We are free to choose from which cell, either from cell $(j-1)$ or from cell $(j+1)$, cell j receives inputs along channels e_j and f_j. It turns out that the first choice, values along channel e_j are sent from cell $(j-1)$ to cell j, is a good one. In particular, the fact that cell 0 can easily generate the values to be sent along channels e_1 and f_1 often indicates an appropriate choice.

Communications along channels e_1 and f_1 are sent by cell 0 and received by cell 1. Therefore, cell 0 must be able to compute both $e_1(i)$ and $f_1(i)$ for all natural i.

By definition, for all natural i: $f_1(i) = f_0(i)$.
The value of $e_1(i)$ is only specified for natural i satisfying $(i+1)^2 > i+2$, i.e. for $i \geq 1$. We are free to choose an appropriate value for $e_1(0)$. For $i \geq 1$:

$$\begin{array}{ll} & e_1(i) \\ = & \quad \{\text{def. } e_j\} \\ & F(1 + 1 \textbf{ div } (i+1)) \\ = & \quad \{i \geq 1 \text{ implies } 1 \textbf{ div } (i+1) = 0\} \\ & F(1) \\ = & \quad \{\text{def. } f_0\} \\ & f_0(0) \end{array}$$

An appropriate choice for the value of $e_1(0)$, now, is $e_1(0) = f_0(0)$, of course.

We proceed by calculating e_{j+1} and f_{j+1} for all $j \geq 1$ and $i \geq 0$. Since communications along channel f_{j+1} are very easy to implement, viz. $f_{j+1}(i) = f_j(i)$, we focus our attention on $e_{j+1}(i)$.

In the calculation of e_{j+1} we use two properties of the **div**-operator:

Property:
For $j \geq 1$ and $i \geq 0$:

0. If $\neg(i+1 \mid j+1)$, then $(j+1) \textbf{ div } (i+1) = j \textbf{ div } (i+1)$.

1. If $(i+1 \mid j+1)$ and, moreover, $(i+1)^2 > i+j+2$, then $(j+1) \textbf{ div } (i+1) = j \textbf{ div } i$.

Proof:
Let $q = (j+1) \textbf{ div } (i+1)$ and $r = (j+1) \textbf{ mod } (i+1)$.
Then, by definition, $(j+1) = q*(i+1) + r \ \wedge \ 0 \leq r < i+1$.

0. We derive

$$\begin{array}{ll} & j+1 = q*(i+1) + r \ \wedge \ 0 \leq r < i+1 \\ = & \quad \{\text{arithmetic}\} \\ & j = q*(i+1) + (r-1) \ \wedge \ -1 \leq (r-1) < i \\ = & \quad \{\neg(i+1 \mid j+1) \text{ implies } r \neq 0\} \\ & j = q*(i+1) + (r-1) \ \wedge \ 0 \leq (r-1) < i \end{array}$$

Hence, $(j+1)\ \mathbf{div}\ (i+1) = q = j\ \mathbf{div}\ (i+1)$.

1. We derive

$$\begin{array}{ll} & (j+1)/(i+1) \\ = & \quad \{\text{arithmetic}\} \\ & (i+j+2)/(i+1) - 1 \\ < & \quad \{(i+1)^2 > i+j+2\} \\ & i \end{array}$$

From $j+1 \neq 0$ and $j+1 = q*(i+1)$, we infer $q \neq 0$. Hence, q satisfies $1 \leq q < i$. Since in this case $j = q*i+(q-1)$ and $0 \leq (q-1) < i$, we conclude that $(j+1)\mathbf{div}(i+1) = q = j\ \mathbf{div}\ i$.

□

Notice that the second premise in the second property reflects the condition which we imposed on the specification of e_{j+1}.

We now derive a relation for $e_{j+1}(i)$. Since $e_{j+1}(i)$ has only been specified for indices i satisfying $(i+1)^2 > i+j+2$ we have

$$\begin{array}{ll} & e_{j+1}(i) \\ = & \quad \{\text{def. } e_j\} \\ & F(1+(j+1)\ \mathbf{div}\ (i+1)) \\ = & \quad \{\text{property above}\} \\ & \mathbf{if}\ \ \neg(i+1 \mid j+1) \ \rightarrow\ F(1+j\ \mathbf{div}\ (i+1)) \\ & [\!]\ \ \ (i+1 \mid j+1) \ \rightarrow\ F(1+j\ \mathbf{div}\ i) \\ & \mathbf{fi} \\ = & \quad \{\text{def. } e_j;\ (i+1)^2 > i+j+2 > i+j+1\} \\ & \mathbf{if}\ \ \neg(i+1 \mid j+1) \ \rightarrow\ e_j(i) \\ & [\!]\ \ \ (i+1 \mid j+1)\ \wedge\ i^2 > i+j \ \rightarrow\ e_j(i-1) \\ & [\!]\ \ \ (i+1 \mid j+1)\ \wedge\ i^2 \leq i+j \ \rightarrow\ F(1+(j+1)/(i+1)) \\ & \mathbf{fi} \end{array}$$

Note that $e_j(i-1)$ is only specified for $i^2 > i+j$. For i and j satisfying both $(i+1 \mid j+1)$ and $i^2 \leq i+j$ and $(i+1)^2 > i+j+2$, we therefore have to determine $F(1+(j+1)/(i+1))$. Since

$$\begin{array}{ll} & i^2 \leq i+j\ \wedge\ (i+1)^2 > (i+1)+(j+1) \\ = & \quad \{\text{arithmetic}\} \\ & i^2 - i + 1 \leq j+1 < i*(i+1) \\ = & \quad \{\text{arithmetic}\} \\ & (i-2) + 3/(i+1) \leq (j+1)/(i+1) < i \\ \Rightarrow & \quad \{(i+1 \mid j+1)\} \\ & (j+1)/(i+1) = i-1 \end{array}$$

we conclude that in this case $F(1+(j+1)/(i+1)) = F(1+(i-1)) = f_j(i-1)$. For indices i that do not satisfy $(i+1)^2 > i+j+2$ we are free to assign any appropriate value to $e_{j+1}(i)$.

The communications along channels e_j and f_j, $j \geq 1$, can now be implemented by

$$e_1(i) = f_0(0) \tag{4}$$

$$f_1(i) = f_0(i) \tag{5}$$

$$\begin{array}{lll} e_{j+1}(i) = & \textbf{if} & \neg(i+1 \mid j+1) \ \vee \ i=0 \quad \rightarrow \ e_j(i) \\ & [\!] & (i+1 \mid j+1) \ \wedge \ i^2 > i+j \quad \rightarrow \ e_j(i-1) \\ & [\!] & (i+1 \mid j+1) \ \wedge \ 0 < i^2 \leq i+j \quad \rightarrow \ f_j(i-1) \\ & \textbf{fi} & \end{array} \tag{6}$$

$$f_{j+1}(i) = f_j(i) \tag{7}$$

Recapitulating, we have introduced a number of channels for which we have derived relations that express the dependencies of the individual communications along these channels. These relations give rise to a partial order on the communications along the channels. We now turn our attention to finding a communication behavior for the cells that is consistent with this partial order.

Given relations (0) through (7) we are able to express the requirements for the communication behavior of the cells. We turn our attention to cell j ($j \geq 1$). For the sake of convenience, we drop the indices of the channel names and denote channels b_j, b_{j+1}, e_j, e_{j+1}, f_j, and f_{j+1} by b, $\bar{b}$, e, $\bar{e}$, f, and $\bar{f}$, respectively.

Relations (2) and (3) give rise to a partial order on the communications along b, $\bar{b}$, e, f (on account of symmetry we temporarily do not consider channels g, $\bar{g}$, h, and $\bar{h}$). A communication behavior that is consistent with this partial order is

$$e, f, b \ ; \ (\bar{b}, e, f \ ; \ b)^* \tag{8}$$

Notice that $e, f, b; \bar{b}, e, f; (\bar{b}, e, f; b)^*$ is, among other possibilities, also an appropriate choice. This communication behavior, however, requires extra buffering of three values. Since we aim at minimal buffering we prefer communication behavior (8).

Relation (6) gives rise to

$$e \ ; \ \bar{e} \ ; \ (e, f \ ; \ \bar{e})^* \tag{9}$$

And, finally, from relation (7) we infer

$$(f \ ; \ \bar{f})^* \tag{10}$$

These relations can be combined into the following (overall) communication behavior, CB, of cell j (including channels g, $\bar{g}$, h, and $\bar{h}$)

$$e, f, g, h \ ; \ (b, \bar{e}, \bar{f}, \bar{g}, \bar{h} \ ; \ \bar{b}, e, f, g, h)^* \tag{11}$$

Notice that the inputs along f occur earlier in (11) than in (9). Hence, we introduced extra buffering. Also notice the alternation between input actions and output actions.

Since

$$CB\lceil\{b,e,f,g,h\} = (e,f,g,h\ ;\ b)^*$$

and

$$CB\lceil\{\bar{b},\bar{e},\bar{f},\bar{g},\bar{h}\} = (\bar{e},\bar{f},\bar{g},\bar{h}\ ;\ \bar{b})^*$$

match, we conclude that the computation we derive does not suffer from deadlock (cf. [6]).
The reader is invited to verify that

$$f,g\ ;\ (b,\bar{e},\bar{f},\bar{g},\bar{h}\ ;\ \bar{b},f,g)^* \tag{12}$$

is a possible communication behavior for cell 0.

From relations (0), (1), (4), (5), and (12) we easily derive a program for cell 0:

$$\begin{array}{l}
[\ \textbf{var}\ vf0, vf, vg0, vg, vb : \text{ int};\\
\quad f?vf0, g?vg0\\
;\ b!(vf0 * vg0), \bar{e}!vf0, \bar{f}!vf0, \bar{g}!vg0, \bar{h}!vg0\\
;\ (\bar{b}?vb, f?vf, g?vg\\
\quad ;\ b!(vb + vf * vg0 + vf0 * vg), \bar{e}!vf0, \bar{f}!vf, \bar{g}!vg, \bar{h}!vg0\\
\quad)^*\\
]
\end{array}$$

And from relations (2), (3), (6), (7), and (11) we derive the following program for cell j:

$$\begin{array}{lll}
[\ \textbf{var} & ve, vee, vf, vff, vg, vgg, vh, vhh, vb : \text{ int}; & \\
& p, q, r, i : \text{ int}; & \\
\quad e?vee, f?vff, g?vgg, h?vhh & & \\
;\ b!0, \bar{e}!vee, \bar{f}!vff, \bar{g}!vgg, \bar{h}!vhh, i := 0 & & \\
;\ (\bar{b}?vb, e?ve, f?vf, g?vg, h?vh & & \\
\quad ;\ \textbf{if} & (i+2)^2 < i+j+2 \ \vee\ \neg(i+2 \mid j) & \rightarrow\ r := 0\\
\quad\ [\!] & (i+2)^2 = i+j+2 & \rightarrow\ r := vf * vg\\
\quad\ [\!] & (i+2)^2 > i+j+2 \ \wedge\ (i+2 \mid j) & \rightarrow\ r := vf * vh + ve * vg\\
\quad\ \textbf{fi} & & \\
\quad ,\ \textbf{if} & \neg(i+2 \mid j+1) & \rightarrow\ p, q := ve, vh\\
\quad\ [\!] & (i+2 \mid j+1) \ \wedge\ (i+1)^2 > i+j+1 & \rightarrow\ p, q := vee, vhh\\
\quad\ [\!] & (i+2 \mid j+1) \ \wedge\ (i+1)^2 \leq i+j+1 & \rightarrow\ p, q := vff, vgg\\
\quad\ \textbf{fi} & & \\
\quad ;\ b!(vb + r), \bar{e}!p, \bar{f}!vf, \bar{g}!vg, \bar{h}!q & & \\
\quad ;\ vee, vff, vgg, vhh, i := ve, vf, vg, vh, i+1 & & \\
\quad)^* & & \\
] & &
\end{array}$$

The above program does not meet the 'modularity constraint' of [3], i.e. j occurs in the program text and as a consequence the operation of a cell depends on the location of that cell in the network. This problem can be eliminated by introducing additional input channels for each cell (this technique has also been applied in [5]). By applying this technique it is possible to implement the evaluation of the guards efficiently. Without going into further detail, we

suggest to introduce three additional input channels u_j, v_j, and w_j, which are specified as follows

$$\begin{aligned} u_j(i) &= (i+1)^2 - (i+j+1) \\ v_j(i) &= j \bmod (i+1) \\ w_j(i) &= i \end{aligned}$$

for $i \geq 1$.
For example,

$$(i+2)^2 < i+j+2 \ \vee \ \neg(i+2 \mid j)$$

can now be replaced by

$$u_j(i+1) < 0 \ \vee \ v_j(i+1) \neq 0$$

For the sake of completeness, the transformed program texts read

$$\begin{array}{l} [\ \textbf{var}\ vf0, vf, vg0, vg, vb, vu, vv, vw : \text{int}; \\ \quad f?vf0, g?vg0 \\ \quad ; vu, vv, vw := -1, 0, 0 \\ \quad ; b!(vf0 * vg0), \bar{e}!vf0, \bar{f}!vf0, \bar{g}!vg0, \bar{h}!vg0, \bar{u}!vu, \bar{v}!vv, \bar{w}!vw \\ \quad ; (\bar{b}?vb, f?vf, g?vg \\ \quad\ ; vu, vv, vw := vu + 2 * vw + 2, 1, vw + 1 \\ \quad\ ; b!(vb + vf * vg0 + vf0 * vg), \bar{e}!vf0, \bar{f}!vf, \bar{g}!vg, \bar{h}!vg0, \bar{u}!vu, \bar{v}!vv, \bar{w}!vw \\ \quad)^* \\ ] \end{array}$$

and

$$\begin{array}{l} [\ \textbf{var}\ \ ve, vee, vf, vff, vg, vgg, vh, vhh, vb, vu, vuu, vv, vvn, vw : \text{int}; \\ \qquad\ p, q, r : \text{int}; \\ \quad e?vee, f?vff, g?vgg, h?vhh, u?vuu, v?vv, w?vw \\ \quad ; b!0, \bar{e}!vee, \bar{f}!vff, \bar{g}!vgg, \bar{h}!vhh, \bar{u}!(vuu - 1), \bar{v}!0, \bar{w}!vw \\ \quad ; (\bar{b}?vb, e?ve, f?vf, g?vg, h?vh, u?vu, v?vv, w?vw \\ \quad\ ; \ \textbf{if}\ vv \neq ww \rightarrow vvn := vv + 1 \ [\!]\ vv = ww \rightarrow vvn := 0\ \textbf{fi} \\ \quad\ ; \ \textbf{if}\ \ vu < 0 \ \vee \ vv \neq 0 \ \rightarrow \ r := 0 \\ \qquad [\!]\ \ vu = 0 \qquad\qquad\ \rightarrow \ r := vf * vg \\ \qquad [\!]\ \ vu > 0 \ \wedge \ vv = 0 \ \rightarrow \ r := vf * vh + ve * vg \\ \qquad \textbf{fi} \\ \quad\ , \textbf{if}\ \ vvn \neq 0 \qquad\qquad\ \rightarrow \ p, q := ve, vh \\ \qquad [\!]\ \ vvn = 0 \ \wedge \ vuu > 0 \ \rightarrow \ p, q := vee, vhh \\ \qquad [\!]\ \ vvn = 0 \ \wedge \ vuu \geq 0 \ \rightarrow \ p, q := vff, vgg \\ \qquad \textbf{fi} \\ \quad ; b!(vb + r), \bar{e}!p, \bar{f}!vf, \bar{g}!vg, \bar{h}!q, \bar{u}!(vu - 1), \bar{v}!vvn, \bar{w}!vw \\ \quad ; vee, vff, vgg, vhh, vuu := ve, vf, vg, vh, vu \\ \quad)^* \\ ] \end{array}$$

We are now done with the construction of our program and conclude this section with a short complexity analysis.

The response time of the program (consider the original program, not the transformed program) is analysed by introducing sequence functions σ_j for each cell j. For a channel a and natural i, $\sigma_j(a,i)$ denotes the time slot in which the i-th communication along channel a of cell j can be scheduled. From the communication behavior of the cells, (11) and (12), the following possible sequence function can be inferred (without loss of generality we only consider channels f, $\bar{f}$, b, and $\bar{b}$)

$$\begin{aligned}\sigma_j(f,i) &= 2*i+j\\ \sigma_j(\bar{f},i) &= 2*i+j+1\\ \sigma_j(b,i) &= 2*i+j+1\\ \sigma_j(\bar{b},i) &= 2*i+j+2\end{aligned}$$

For cell 0 we have $\sigma_0(b,i) = 2*i+1$. Hence, the computation we derived has constant response time. In the same time slot in which $b_0(i)$ is produced by cell 0 cell $(2*i+1)$ receives $f_{2*i+1}(0)$. Thus, computing $(F\star G)(n)$, for $1 \leq n \leq N$, involves $\mathcal{O}(N)$ cells and $\mathcal{O}(N)$ time. A sequential solution for computing $(F \star G)(n)$, for $1 \leq n \leq N$, has time complexity at least $\mathcal{O}(N \log N)$.

2 Inverse Convolution Problem

In this section we present a parallel program for the inverse convolution problem. It turns out that this parallel program is identical to the parallel program for Dirichlet Convolution, except for the design of cell 0.

The inverse convolution problem is stated as follows: given two arithmetical functions, G and H, one has to determine (arithmetical) function F such that $F \star G = H$, i.e.

$$H(n) = (\mathrm{S}\, p,q : p*q = n \;\wedge\; 1 \leq p \;\wedge\; 1 \leq q : F(p)*G(q))$$

for $n \geq 1$. Assume $G(1) \neq 0$.

The computation we derive consists of a linear network of cells where cell 0 is fed with the two given arithmetical functions along two input channels, g_0 and h_0:

$$\begin{aligned}g_0(i) &= G(i+1)\\ h_0(i) &= H(i+1)\end{aligned}$$

for $i \geq 0$.
Communication with the environment is established by means of output channel b_0, which satisfies

$$b_0(i) = F(i+1)$$

for $i \geq 0$ and F satisfying $F \star G = H$.
Since F is defined implicitly we derive relations for $F(n)$ and, next, extract $F(n)$ from these.

From $H(1) = F(1) * G(1)$, we readily conclude

$$b_0(0)$$

$$
\begin{array}{ll}
= & \{\text{def. } b_0\} \\
& F(1) \\
= & \{\text{relation above; } G(1) \neq 0\} \\
& H(1)/G(1) \\
= & \{\text{def. } g_0 \text{ and } h_0\} \\
& h_0(0)/g_0(0)
\end{array}
$$

For $n \geq 1$, we have

$$
\begin{array}{ll}
& H(n+1) \\
= & \{F * G = H\} \\
& (\mathbf{S}\, p,q : p * q = n+1 \ \wedge\ (\sqrt{n+1} \leq p \ \vee\ \sqrt{n+1} \leq q) : F(p) * G(q)) \\
= & \{\text{domain split; } 1 < \sqrt{n+1}\} \\
& F(1) * G(n+1) + F(n+1) * G(1) \\
& +(\mathbf{S}\, p,q : p * q = n+1 \ \wedge\ (\sqrt{n+1} \leq p \leq n \ \vee\ \sqrt{n+1} \leq q \leq n) : F(p) * G(q))
\end{array}
$$

Since $G(1) \neq 0$, we conclude that function F is unique.

Now, recall the definition of $Q(m,n)$ from the previous section. $Q(m,n)$ has been defined in the context of arithmetical functions F and G. Therefore, it is possible to substitute $Q(n,n+1)$ for the quantified summation in the derivation above, giving

$$H(n+1) = F(1) * G(n+1) + F(n+1) * G(1) + Q(n,n+1)$$

Cell 0 should have at its disposal the value of $Q(n,n+1)$ for each $n \geq 1$. For this purpose we can use the cells with $j \geq 1$ that already have been implemented in the previous section. Then $b_1(i) = Q(i+1,i+2)$ for $i \geq 0$, provided that cell 1 is supplied with the proper values. For $i \geq 0$ we derive

$$
\begin{array}{ll}
& b_0(i+1) \\
= & \{\text{def. } b_0\} \\
& F(i+2) \\
= & \{\text{above relation for } H(n+1);\ G(1) \neq 0\} \\
& (H(i+2) - F(1) * G(i+2) - Q(i+1,i+2))/G(1) \\
= & \{\text{def. } g_0,\ h_0,\ b_0, \text{ and } b_1\} \\
& (h_0(i+1) - b_0(0) * g_0(i+1) - b_1(i))/g_0(0)
\end{array}
$$

Summarizing:

$$b_0(0) = h_0(0)/g_0(0) \tag{13}$$

$$b_0(i+1) = (h_0(i+1) - b_0(0) * g_0(i+1) - b_1(i))/g_0(0) \tag{14}$$

A possible communication behavior for cell 0 is (cf. (12))

$$g,h\ ;\ (b,\bar{e},\bar{f},\bar{g},\bar{h}\ ;\ \bar{b},g,h)^*$$

The corresponding program for cell 0 reads

$$
\begin{array}{l}
[\ \textbf{var}\ vf0, vf, vg0, vg, vb, vh : \text{int};\\
\quad g?vg0, h?vh\\
\quad ; vf0 := vh/vg0\\
\quad ; b!vf0, \bar{e}!vf0, \bar{f}!vf0, \bar{g}!vg0, \bar{h}!vg0\\
\quad ; (\bar{b}?vb, g?vg, h?vh\\
\quad\quad ; vf := (vh - vf0 * vg - vb)/vg0\\
\quad\quad ; b!vf, \bar{e}!vf0, \bar{f}!vf, \bar{g}!vg, \bar{h}!vg0\\
\quad\quad)^*\\
\quad]
\end{array}
$$

3 The Möbius Function

The Möbius function μ is the arithmetical function defined by

$$\mu(n) = \begin{cases} 0 & \text{if } (\mathbf{E}\, m : m > 1 : m^2 \mid n) \\ (-1)^{\pi(n)} & \text{otherwise} \end{cases}$$

for $n \geq 1$, where $\pi(n)$ denotes the number of prime divisors of n.

It is well-known that the Möbius function is an instance of the inverse convolution problem, viz.

$$\mu \star E = U$$

where E is the all-one function, and U is defined by $U(1) = 1$, and $U(n) = 0$ for all $n > 1$.

A parallel program that computes the Möbius function can be obtained from the program for the (general) inverse convolution problem by feeding cell 0 with input streams g and h that satisfy $g(i) = E(i+1)$ and $h(i) = U(i+1)$ for $i \geq 0$. By exploiting knowledge about functions E and H it is possible, however, to eliminate a number of communication actions from the program texts of the cells. By doing so, the input channels of cell 0 can be omitted which results in a parallel program that only produces output.

After elimination of redundant statements we obtain the following program texts. For cell 0 we get

$$
\begin{array}{l}
[\ \textbf{var}\ vb : \text{int};\\
\quad b!1, \bar{e}!1, \bar{f}!1\\
\quad ; (\bar{b}?vb\\
\quad\quad ; b!(-vb-1), \bar{e}!1, \bar{f}!(-vb-1)\\
\quad\quad)^*\\
\quad]
\end{array}
$$

and for cell j $(j \geq 1)$

$$
\begin{array}{l}
[\ \textbf{var}\ \ ve, vee, vf, vff, vb : \text{int};\\
\qquad p, r, i : \text{int};\\
\ e?vee, f?vff\\
;\, b!0, \bar{e}!vee, \bar{f}!vff, i := 0\\
;\, (\bar{b}?vb, e?ve, f?vf\\
\ ;\ \textbf{if}\ \ (i+2)^2 < i+j+2 \ \vee\ \neg(i+2 \mid j) \ \rightarrow\ r := 0\\
\quad [\!]\ \ (i+2)^2 = i+j+2 \ \rightarrow\ r := vf\\
\quad [\!]\ \ (i+2)^2 > i+j+2 \ \wedge\ \ (i+2 \mid j) \ \rightarrow\ r := vf + ve\\
\quad \textbf{fi}\\
\ ,\ \textbf{if}\ \ \neg(i+2 \mid j+1) \ \rightarrow\ p := ve\\
\quad [\!]\ \ (i+2 \mid j+1) \ \wedge\ (i+1)^2 > i+j+1 \ \rightarrow\ p := vee\\
\quad [\!]\ \ (i+2 \mid j+1) \ \wedge\ (i+1)^2 \leq i+j+1 \ \rightarrow\ p := vff\\
\quad \textbf{fi}\\
\ ;\, b!(vb + r), \bar{e}!p, \bar{f}!vf\\
\ ;\, vee, vff, i := ve, vf, i+1\\
\ )^*\\
]
\end{array}
$$

Our program for generating the Möbius function differs from the program presented in [5]. This is mainly caused by the fact that in [5] there was no need for a 'symmetric solution'. Such a solution even would not have been obvious.

4 Concluding Remarks

We have derived parallel programs for Dirichlet Convolution and for the inverse convolution problem in a calculational, rather straightforward manner. A key issue in the derivation was the decision to maintain the symmetry of the problem specification in the generalized expression $Q(m, n)$. It is our experience that destroying symmetry in the derivation of parallel programs often yields inefficient solutions. In fact, this observation has also been made in [0, section 3]. Another important step in the derivation was the fact that we did not specify the additional input channels e_j and h_j for all natural i. In this way we made it possible to apply the second property that we derived for the **div** -operator.

We believe that our derivation is much clearer than the program derivations given in [3] and [0], which are, in a sense, based on similar but less explicit observations as our solution is based on. In [3], a rather intricate routing scheme is given for the routing of 'F-coefficients' and 'G- coefficients', which can be compared to the input channels e_j and h_j in our solution. We, however, refrained from giving an operational explanation for the behavior of the values communicated along channels e_j and h_j: such an explanation would only complicate the reasoning about our program. In [0], 'domain contraction' has been applied in order to obtain an efficient (symmetric) solution. This technique seems to be a little magical and hard to understand if one is not familiar with the method.

Starting from a parallel program for Dirichlet Convolution it turned out to be very simple to derive a parallel program for the inverse convolution problem: both programs are identical except for the design of cell 0. We have already come across this phenomenon in the design of systolic arrays for polynomial multiplication and division (cf. [4]).

Finally, we have presented a parallel program for computing the Möbius function. Our pro-

gram differs from the program presented in [5], which is mainly caused by the fact that in [5] there was no need for a 'symmetric solution'. Such a solution even would not have been obvious.

Acknowledgements

Acknowledgements are due to Martin Rem, Tom Verhoeff, and Joost P. Katoen for making comments on an earlier version of this paper.

References

[0] Marina Chen, Young-il Choo, *Synthesis of a Systolic Dirichlet Product Using Non-Linear Domain Contraction*, Tech. Rep. , YALEU/DCS/TR-664, Yale University, Dec. 1988.

[1] Rudolf Mak, Pieter Struik, *A Systolic Design for Dynamic Programming*, Computing Science Note 89/2, Eindhoven University of Technology, The Netherlands (1989).

[2] P.J. McCarthy, *Introduction to Arithmetical Functions*, Springer-Verlag, 1986, ch. 1.

[3] Patrice Quinton, Yves Robert, *Systolic Convolution of Arithmetic Functions*, IRISA Research Report nr. 449, Jan. 1989.

[4] Martin Rem, *Trace Theory and Systolic Computations*, in PARLE: Parallel Architectures and Languages Europe, Proceedings 1987 (J.W. de Bakker et al., eds.), Lecture Notes in Computer Science 258, Springer-Verlag, Berlin, 1987, pp. 14–33.

[5] Tom Verhoeff, *A Parallel Program That Generates the Möbius Sequence*, Computing Science Note 88/01, Eindhoven University of Technology, The Netherlands (1988).

[6] Gerard Zwaan, *Parallel Computations*, Ph.D.-thesis, Eindhoven University of Technology, The Netherlands (1989).

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.J.L. Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes

86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns, searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language

87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples
87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to handle Structured Information
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling

88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases
88/12	A.E. Eiben	Abstract theory of planning
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees
88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.