Bulk Synchronous Parallel Scheduling of Uniform Dags

Radu Calinescu

Programming Research Group, Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford OX1 3QD, England

Abstract. This paper addresses the dag scheduling problem, proposing the bulk synchronous parallel (BSP) model as a framework for the derivation of general purpose parallel computer schedules of uniform dags, i.e., of dags that stand for tightly-nested loops with computable distance vectors. A general technique for the BSP scheduling of normalised uniform dags is introduced and analysed in terms of the BSP cost model, and methods for the normalisation of generic uniform dags are briefly overviewed in the paper.

1 Introduction

During the last two decades a great deal of research effort has been devoted to the identification and scheduling of potential parallelism. Despite the criticism pointing out its slow pace of progress, this research has led to remarkable advances. Data dependence analysis [2], loop transformation [1, 13], potential parallelism identification [6, 13], and dag scheduling [4, 7] are but a few examples of fields whose tremendous development has provided techniques successfully incorporated into nowadays' optimising compilers and tools.

Nevertheless, there is a justifiable dissatisfaction about the overall progress in the scheduling area; one of its main causes is the lack of unity in approaching the problem, especially the lack of a standard and realistic parallel programming and cost model. Therefore, the researchers have had either to restrict their interest to the analysis of virtual parallelism—when they wanted to remain general, or to address the problem for a specific parallel architecture—if they aimed at being practical. Clearly, both options have their disadvantages. This paper avoids limiting to one of the two directions by proposing the bulk synchronous parallel (BSP) programming and cost model [11, 8] as a target platform for the derivation of scalable dag schedules for general purpose parallel computers.

The paper is organised as follows. In Sect. 2, the BSP model is briefly described. Then, in Sect. 3, a new technique for the BSP scheduling of *normalised uniform dags*, i.e., of dags that stand for tightly-nested loops whose data dependences are expressible as distance vectors with all non-negative components, is devised and analysed in terms of the BSP cost model. Several methods for the conversion of generic uniform dags (or, equivalently, of generic tightly-nested loops with computable distance vectors) to normalised uniform dag form are mentioned in Sect. 4. A final section including a short summary and further work directions concludes the paper.

2 The Bulk Synchronous Parallel Model

The existence of a standard model is the only way to fully impose parallel computing as a viable alternative to sequential computing. The BSP model [11, 8] provides such a unifying framework for the design and programming of general purpose parallel computers. A BSP computer consists of a set of processor-memory pairs, a communication network for point-to-point message delivery, and a mechanism for efficient barrier synchronisation of all processors or of a subset of processors. No specialised broadcasting or combining facilities are available.

The performance of a BSP computer is characterised by three parameters: p, the number of processors; L, the minimal number of time steps between successive synchronisation operations, or the *synchronisation periodicity*; and g, the ratio between the total number of local operations performed by all processors in one second and the total number of words delivered by the communication network in one second. The parameter L is a measure of the network latency, whereas the parameter g is related to the time required to complete a so-called *h-relation*, i.e., a routing problem where any processor has at most h packets to send to various processors in the network, and where at most h packets are to be received by any processor; practically, g is the value such that gh is an upper bound for the number of time steps required to perform an h-relation.

A BSP computation consists of a sequence of supersteps; in any superstep, the processors may execute operations on locally held data and/or initiate read/write requests for non-local data. However, these non-local memory accesses take effect only when all the processors reach the barrier synchronisation that ends that superstep. To assess the complexity of a BSP algorithm, one has to count the costs of its constituent supersteps. The cost of a superstep S depends on the synchronisation cost (L), on the maximum number of local computation steps executed by any processor during S (w), and on the maximum number of messages sent/received by any processor during S (h_s, respectively h_r)[1]:

$$cost(S) = \max\{L, w, gh_s, gh_r\} \tag{1}$$

Accordingly, the complexity of a BSP algorithm depends not only on the problem size and on the number of processors, but also on the BSP parameters L and g.

3 BSP Scheduling of Normalised Uniform Dags

A normalised uniform dag is a directed acyclic graph used to represent a *fully permutable* [12] loop nest (Fig. 1). The dag is uniform in the sense that edges (data dependences) are uniform across the vertex space (iteration space); the uniform dag is normalised if any edge head points to a vertex (point in the iteration space) with all coordinates greater than or equal to those corresponding to the tail of the edge. A formal definition follows.

[1] several expressions have been proposed for the cost of a superstep; since all these expressions are equivalent within a small multiplicative constant, we chose the one in [8], which best accounts for the overlapping of computation and communication

$$
\begin{array}{l}
\text{for } i_1 = 0, n-1 \text{ do} \\
\quad \text{for } i_2 = 0, n-1 \text{ do} \\
\qquad \vdots \\
\qquad \text{for } i_K = 0, n-1 \text{ do} \\
\qquad\quad (a_1[i], a_2[i], \ldots, a_r[i]) = f(a_{\alpha_1}[i-d_1], a_{\alpha_2}[i-d_2], \ldots, a_{\alpha_q}[i-d_q])
\end{array}
$$

Fig. 1. A fully permutable loop nest: $i = (i_1, i_2, \ldots, i_K)$; a_j, $1 \leq j \leq r$ is a K-dimensional array; $a_{\alpha_j} \in \{a_1, a_2, \ldots, a_r\}$ and d_j is a K-dimensional distance (dependence) vector with non-negative elements, $1 \leq j \leq q$; $f : \mathbb{R}^q \to \mathbb{R}^r$.

Definition 1. A K-dimensional normalised uniform dag (NU-dag for short) is a directed acyclic graph $G = (V, E)$ with:

$$
\begin{array}{l}
V = (0..n-1)^K, \; n \in \mathbb{N}^*, \\
E = \{ < x, x+d > : < V, V > \mid \; d \in \{d_1, d_2, \ldots, d_q\} \},
\end{array}
\tag{2}
$$

where $d_j : \mathbb{N}^K \backslash \{(0,0,\ldots,0)\}$, $1 \leq j \leq q$. Each vertex $i \in V$ stands for a computation

$$(a_1[i], a_2[i], \ldots, a_r[i]) = f(a_{\alpha_1}[i-d_1], a_{\alpha_2}[i-d_2], \ldots, a_{\alpha_q}[i-d_q]),$$

where $a_j[i] : \mathbb{R}$, $1 \leq j \leq r$, $a_{\alpha_j} \in \{a_1, a_2, \ldots, a_r\}$ for $1 \leq j \leq q$, and $f : \mathbb{R}^q \to \mathbb{R}^r$, with appropriate boundary values for the arrays a_j, $1 \leq j \leq r$.

Although Definition 1 requires that all loops iterate from 0 to $n-1$, the results in this section can be easily extended to the case when $V = (0..n_1 - 1) \times (0..n_2 - 1) \times \cdots \times (0..n_K - 1)$. It is also worth noticing that, if any identical vectors $d_j = d_{j'}$, $1 \leq j < j' \leq q$ exist, $D = \{d_1, d_2, \ldots, d_q\}$ must be considered a bag instead of a set; multiple edges between some pairs of vertices exist in this case.

In the following, we shall consider the case when D contains K linearly independent vectors; otherwise, a partition of the dag into p independent sub-dags of equivalent complexity exists and can be used to perform the loop in a single computation superstep. To derive a p-processor BSP program for the computation of an NU-dag, the dag G is first partitioned into $p^{K/(K-1)}$ identical sub-dags[2] $G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}$, $0 \leq \hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K < p^{1/(K-1)}$, each of which is isomorphic with G:

$$
\begin{array}{rl}
G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K} & = (V^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}, E^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}) = \\
& = (\{(i_1, i_2, \ldots, i_K) : V \mid \forall j : 1..K \bullet \hat{i}_j \frac{n}{p^{1/(K-1)}} \leq i_j < (\hat{i}_j + 1) \frac{n}{p^{1/(K-1)}} \}, \\
& \quad \{ < x, y > : E \mid \; x, y \in V^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K} \}).
\end{array}
$$

The computation of each sub-dag $G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}$—also called a *tile*—will be assigned to one of the p processors of the parallel machine. Data dependences codified by edges in an $E^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}$ set are internalised by this approach (i.e., they are solved locally by the processor computing $G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}$), whereas edges in E but

[2] assume that $K > 1$ and $p^{1/(K-1)}$ divides n

not in any $E^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}$ stand for data dependences that must be satisfied through inter-processor communication. The choice of the tile size (i.e., $n/p^{1/(K-1)}$) is justified later by Theorem 3.

The graph $\widehat{G} = (\widehat{V}, \widehat{E}) = ((0..p^{1/(K-1)} - 1)^K, \{< x, y >:< \widehat{V}, \widehat{V} > \mid x \neq y \wedge \exists < \alpha, \beta >: E \bullet \alpha \in V^x \wedge \beta \in V^y\})$ is itself an NU-dag having a vertex for each tile in the original dag; we shall call it the *tile dag*. The hyperplane method of Lamport [6] is used to schedule the tile dag for parallel execution. This widely applied method (also known as *wavefront* scheduling [1, 13]) consists of successively scheduling for concurrent execution the intersections of the iteration space ($(0..p^{1/(K-1)} - 1)^K$ in our case) with a family of $(K-1)$-dimensional parallel hyperplanes. The number of hyperplanes in this family gives the number of supersteps required to accomplish the computation (i.e., the *schedule length*).

Theorem 2. *Let $G = (V, E)$ be a K-dimensional NU-dag and $\widehat{G} = (\widehat{V}, \widehat{E})$ be its associated tile dag. Assume that the tile size is large enough (i.e., larger than the distance vector sizes) for the dependences not internalised by the tiling to occur only between neighbour tiles, and for any tile $G^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}$ to depend (at least) on each of the tiles $G^{\hat{i}_1-1,\hat{i}_2,\ldots,\hat{i}_K}$, $G^{\hat{i}_1,\hat{i}_2-1,\ldots,\hat{i}_K}$, $\ldots$, $G^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K-1}$. Then, the set of $(K-1)$-dimensional hyperplanes given by*

$$\{\hat{i}_1 + \hat{i}_2 + \cdots + \hat{i}_K = t \mid 0 \leq t \leq Kp^{1/(K-1)} - K\} \tag{3}$$

defines a minimum-length legal schedule for the tile dag $\widehat{G}$.

Proof. It is a well-known result (see for instance [5]) that when a family of hyperplanes $\alpha_1 i_1 + \alpha_2 i_2 + \cdots + \alpha_K i_K = t$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_K) : \mathbb{N}$ is used for the (linear) scheduling of a size x hypercube whose distance vector set is D, the schedule is legal if and only if $\forall d \in D \bullet \alpha \cdot d > 0$, and requires a number of $(\alpha_1 + \alpha_2 + \cdots + \alpha_K)x + 1$ supersteps. In our case, D includes $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0), \ldots, (0, 0, \ldots, 1)$ (and possibly other distance vectors $d : \{0, 1\}^K$), so the schedule is correct if and only if $\alpha_j \geq 1$, $1 \leq j \leq K$. Therefore, the schedule in (3), which corresponds to $\alpha_j = 1$, $1 \leq j \leq K$, is legal and has minimum length. □

Thus, the BSP schedule of a K-dimensional NU-dag (Fig. 2) requires $Kp^{1/(K-1)} - K + 1$ supersteps, with the following actions taking place in each superstep t, $0 \leq t \leq Kp^{1/(K-1)} - K$: (1) each tile $G^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}$ with $\hat{i}_1 + \hat{i}_2 + \cdots + \hat{i}_K = t$ is computed by a distinct processor; (2) the results required for the computation of other tiles are sent to the processors that will compute those tiles in later supersteps.

Theorem 3. *At most p processors are required in any superstep of the BSP schedule in Fig. 2.*

Proof. Let $x = p^{1/(K-1)}$, and let $N_K : 0..Kx - K \to \mathbb{N}$ be a function such that for any $0 \leq t \leq Kx - K$, $N_K(t)$ gives the number of tiles scheduled for concurrent execution in superstep t. We shall prove by induction that, for any $K > 1$, the following proposition equivalent to the theorem's statement is true:

$$P(K): \quad \forall t : 0..Kx - K \bullet N_K(t) \leq x^{K-1}.$$

for $t = 0, Kp^{1/(K-1)} - K$ do
 forall $(\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K) : (0..p^{1/(K-1)} - 1)^K$ such that $\hat{i}_1 + \hat{i}_2 + \cdots + \hat{i}_K = t$ do in parallel
  (1) compute $G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}$
   for $i_1 = \hat{i}_1 \frac{n}{p^{1/(K-1)}}, (\hat{i}_1 + 1) \frac{n}{p^{1/(K-1)}} - 1$ do
    for $i_2 = \hat{i}_2 \frac{n}{p^{1/(K-1)}}, (\hat{i}_2 + 1) \frac{n}{p^{1/(K-1)}} - 1$ do
     $\vdots$
     for $i_K = \hat{i}_K \frac{n}{p^{1/(K-1)}}, (\hat{i}_K + 1) \frac{n}{p^{1/(K-1)}} - 1$ do
      $(a_1[i], a_2[i], \ldots, a_r[i]) = f(a_{\alpha_1}[i - d_1], a_{\alpha_2}[i - d_2], \ldots, a_{\alpha_q}[i - d_q])$
  (2) send data required by other tile computations

Fig. 2. The pseudocode for the p-processor BSP scheduling of a K-dimensional NU-dag. An initial superstep in which input data is provided for the boundary tiles must precede the whole computation.

The base step is immediate. For the induction step, assume that $P(n)$ is true and let $K = n + 1$. Then, $N_{n+1}(t)$ can be computed as a sum counting the solutions for the cases when $\hat{i}_{n+1} = \tau$ and $(\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_n)$ is allowed to take any value such that $\hat{i}_1 + \hat{i}_2 + \cdots + \hat{i}_n = t - \tau$, for $\tau = 0, 1, \ldots, t$. Formally,

$$\forall t : 0..(n+1)x - (n+1) \bullet N_{n+1}(t) = \alpha_0 N'_n(t) + \alpha_1 N'_n(t-1) + \cdots + \alpha_t N'_n(0),$$

where $\alpha_\tau = \begin{cases} 1, & \text{if } \tau \leq x - 1 \\ 0, & \text{otherwise} \end{cases}$ and $N'_n(\tau) = \begin{cases} N_n(\tau), & \text{if } \tau \leq nx - n \\ 0, & \text{otherwise} \end{cases}, 0 \leq \tau \leq t.$

Indeed, we must count only the cases when a legal value is assigned to i_{n+1}. Since i_{n+1} may be assigned only x legal values, $\alpha_0 + \alpha_1 + \cdots + \alpha_t = \min\{x, t\} \leq x$, or

$$\forall t : 0..(n+1)x - (n+1) \bullet N_{n+1}(t) \leq x \max_{0 \leq \tau \leq t} N'_n(\tau) \leq x\, x^{n-1} = x^n,$$

which is $P(n+1)$ (the induction hypothesis was used for the last step). □

The last result needed for the computation of the cost of a superstep is provided by the following theorem[3].

Theorem 4. *The amount of data sent by a processor after computing an inner tile $G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_d}$ is given by*

$$Com = \sum_{d \in D} \left[\left(\frac{n}{p^{1/(K-1)}} \right)^K - \prod_{j=1}^{K} \left(\frac{n}{p^{1/(K-1)}} - d_j \right) \right] = \frac{n^{K-1}}{p} \left(\sum_{d \in D} \sum_{j=1}^{K} d_j + o(1) \right) \quad (4)$$

The amount of data received by any processor during a superstep is at most Com.

Proof. Let $d \in D$; this vector implies data dependences between pairs of vertices of the form $< i, i + d >$. For a given tile $G^{\hat{i}_1, \hat{i}_2, \ldots, \hat{i}_K}$, we are interested in finding

[3] results similar to the first part of this theorem have been presented in [3, 9, 10]

the number of such pairs whose first element belongs to $G^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}$, but whose second element does not:

$$
\begin{aligned}
&Com(d) = \#\{<i, i+d>:<V, V> \mid i \in V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} \wedge (i+d) \in V \setminus V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}\} \\
&= \#(V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} \setminus \{i : V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} \mid \forall j : 1..K \bullet i_j + d_j < (\hat{i}_j + 1)\tfrac{n}{p^{1/(K-1)}}\}) \\
&= \#V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} - \#\{i : V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} \mid \forall j : 1..K \bullet i_j + d_j < (\hat{i}_j + 1)\tfrac{n}{p^{1/(K-1)}}\} \\
&= \#V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} - \#\{i : \mathbb{N}^K \mid \forall j : 1..K \bullet \hat{i}_j \tfrac{n}{p^{1/(K-1)}} \leq i_j < (\hat{i}_j + 1)\tfrac{n}{p^{1/(K-1)}} - d_j\} \\
&= \left(\tfrac{n}{p^{1/(K-1)}}\right)^K - \prod_{j=1}^{K}\left(\tfrac{n}{p^{1/(K-1)}} - d_j\right).
\end{aligned}
$$

As each distance vector brings a similar contribution to the amount of data sent by a processor after computing $G^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}$, and does this independently of other distance vectors, (4) is now straightforward.

For the second part of the theorem, let us consider the worst case, i.e., the case of a processor P_x which is assigned a tile in each of the K supersteps that may depend on data sent during the current superstep t. Consider now all the vertices in the K isomorphic tiles assigned to P_x during the supersteps $t+1, t+2, \ldots, t+K$, and let $x = (x_1, x_2, \ldots, x_K)$ be one of these vertices such that $x - d \in V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K}$ with $\hat{i}_1 + \hat{i}_2 + \cdots + \hat{i}_K = t$. Then, the distance vector d does not generate dependences on data computed during the superstep t for any of the other $K-1$ vertices isomorphic with x. Indeed, assume that such a vertex $y = (y_1, y_2, \ldots, y_K) = (x_1 + \alpha_1 \frac{n}{p^{1/(K-1)}}, x_2 + \alpha_2 \frac{n}{p^{1/(K-1)}}, \ldots, x_K + \alpha_K \frac{n}{p^{1/(K-1)}})$ existed with $\sum_{j=1}^{K} \alpha_j \neq 0$ (the two vertices must belong to tiles computed in different supersteps). Then, we have:

$$
\begin{aligned}
&x - d \in V^{\hat{i}_1,\hat{i}_2,\ldots,\hat{i}_K} \Longrightarrow \forall j : 1..K \bullet \hat{i}_j \tfrac{n}{p^{1/(K-1)}} \leq x_j - d_j < (\hat{i}_j + 1)\tfrac{n}{p^{1/(K-1)}} \Longrightarrow \\
&\forall j : 1..K \bullet (\hat{i}_j + \alpha_j)\tfrac{n}{p^{1/(K-1)}} \leq x_j + \alpha_j \tfrac{n}{p^{1/(K-1)}} - d_j < (\hat{i}_j + \alpha_j + 1)\tfrac{n}{p^{1/(K-1)}} \Longrightarrow \\
&\forall j : 1..K \bullet (\hat{i}_j + \alpha_j)\tfrac{n}{p^{1/(K-1)}} \leq y_j - d_j < (\hat{i}_j + \alpha_j + 1)\tfrac{n}{p^{1/(K-1)}} \Longrightarrow \\
&y - d = (y_1 - d_1, y_2 - d_2, \ldots, y_K - d_K) \in V^{\hat{i}_1+\alpha_1,\hat{i}_2+\alpha_2,\ldots,\hat{i}_K+\alpha_K}.
\end{aligned}
$$

Since both $x - d$ and $y - d$ are supposed to be computed in the same superstep $t = \sum_{j=1}^{K} \hat{i}_j = \sum_{j=1}^{K} (\hat{i}_j + \alpha_j)$, the assumption that $\sum_{j=1}^{K} \alpha_j \neq 0$ is contradicted. We can therefore "superpose" the K isomorphic tiles, and for any vertex for which the distance vector d involves an external data dependence, at most one vertex from the superposed set of tiles will receive this data from a tile computed in superstep t. Hence, the maximum amount of data received by P_x during superstep t due to the distance vector d can be computed as in the first part of the proof, and the desired result is immediate. □

Consequently, the cost of a superstep t of the schedule is

$$
cost(t) = \max\left\{L, \left(\frac{n}{p^{1/(K-1)}}\right)^K cost(f), gCom\right\},
$$

with Com given by (4), and $cost(f)$ representing the computational cost for a single vertex of the dag. After simplifications, the cost of the entire schedule is

$$
\max\left\{(Kp^{\frac{1}{(K-1)}} - K + 1)L, (K + o(1))\frac{n^K}{p} cost(f), (Kp^{\frac{1}{(K-1)}} - K + 1) gCom\right\}. \quad (5)
$$

Thus, K-optimality[4] is obtained if the BSP parameters L and g are low enough (i.e., $L < n^K cost(f)/p^{K/(K-1)}$ and $g < n\ cost(f)/(p^{1/(K-1)} \sum_{d \in D} \sum_{j=1}^{K} d_j)$).

4 Iteration Space Normalisation

Since not all tightly-nested loops can be directly mapped onto an NU-dag, the BSP schedule proposed in the Sect. 3 would be really useful if a procedure existed to transform a generic uniform iteration space into a normalised one. As proved by Wolf and Lam [12], such a procedure does exist in the general case; indeed, the authors show in [12] that any perfect loop nest with computable distance vectors can be converted into a fully permutable loop nest (which they call the *canonical form* of the loop nest) by using an affine transformation of the iteration space. Formally, if $\mathbf{D}$ is the $K \times q$ matrix whose columns are the distance vectors of the original loop nest, a $O(K^2q)$-time algorithm that finds a unimodular lower triangular transformation matrix $\mathbf{T} = (t_{ij})_{1 \leq i,j \leq K}$ such that $\mathbf{D}^+ = \mathbf{T} \cdot \mathbf{D} \geq \mathbf{0}$ (where the inequality is applied component-wise) is developed in [12]. However, Wolf and Lam pay no attention to choosing a transformation that would result in a minimum communication overhead. Still, if the NU-dag BSP schedule is to be extended for generic uniform dags, one needs to know this communication overhead for the transformed loop.

Corollary 5. *The amount of data exchanged by a processor computing a size x hypercubic tile of the transformed iteration space is*

$$Com' = x^{K-1} \left(\sum_{i=1}^{K} \sum_{j=1}^{q} \sum_{k=1}^{K} t_{ik} d_{kj} + o(1) \right). \tag{6}$$

Proof. Since the distance vectors of the transformed loop are $(d_{1j}^+, d_{2j}^+, \ldots, d_{Kj}^+)$, $1 \leq j \leq q$, with $d_{ij}^+ = \sum_{k=1}^{K} t_{ik} d_{kj}$, (6) directly follows from Theorem 4. □

Several approaches to find a transformation $\mathbf{T}$ that minimises (6) have been proposed so far. Ramanujam and Sadayappan [9] have formulated a linear programming problem whose solution is an optimal unimodular lower triangular transformation $\mathbf{T}$: find $(t_{ij})_{1 \leq i,j \leq K}$ which minimises $\sum_{i=1}^{K} \sum_{j=1}^{q} \sum_{k=1}^{K} t_{ik} d_{kj}$ subject to $t_{ii} = 1$, $1 \leq i \leq K$; $t_{ij} = 0$, $1 \leq i < j \leq K$; $\sum_{k=1}^{K} t_{ik} d_{kj} \geq 0$, $1 \leq i, j \leq K$. Although such a transformation leads to a particularly simple rewriting of the loop nest, better results can be obtained when no restriction are placed on the affine transformation $\mathbf{T}$. Thus, Schreiber and Dongarra [10] have developed a heuristic which aims at maximising the computation to communication ratio for the transformed loop nest. Also, in [3], Boulet et al. devise a method that yields, for any fixed amount of computation, a legal tiling of the iteration space which minimises the communication overhead[5].

[4] remember that K is rarely larger than 3 or 4

[5] if $\mathbf{D}$ is square, the matrix of the affine transformation induced by this method is $\mathbf{D}^{-1}$ (recall that $\mathbf{D}$ is non-singular), which is optimal under all circumstances

5 Conclusions

This paper has proposed a strategy for the BSP scheduling of loop nests representable as uniform dags, attempting to clarify the relation between tiling and scheduling for real computers—a currently open problem raised in [3]. The resulting schedule is portable across any parallel platform, and attains K-optimality (where K is the depth of the loop nest) if the BSP parameters L and g of the target machine are low enough (or, equivalently, if the problem size is large enough).

Further work is required to asses the effectiveness of varying the tile size across the computation in order to improve the load balancing, while maintaining the synchronisation and communication overheads within acceptable bounds. Also, a slight modification of the basic NU-dag scheduling technique must be considered after the normalisation of a generic uniform dag, since the transformed iteration space is no longer rectangular. Finally, subsequent efforts must focus on the design of techniques for the scheduling of untightly-nested loops onto general purpose parallel computers.

Acknowledgments The author would like to thank Dr. W.F. McColl for his constant advice and encouragement. The author is also grateful to an anonymous referee whose comments resulted in a significant improvement of the paper.

References

1. D.F. Bacon et al., Compiler transformations for high-performance computing. In: *ACM Comp. Surv.* **26**(4), Dec. 1994, 346-420.
2. U. Banerjee, *Dependence Analysis for Supercomputing.* Kluwer Acad. Publ., 1988.
3. P. Boulet et al., (Pen)-ultimate tiling?. In: *Integration, the VLSI Journal* **17**(1), Aug. 1994, 33-51.
4. A. Gerasoulis and T. Yang, A comparison of clustering heuristics for scheduling DAGs on multiprocessors. In: *J. Par. Distr. Comp.* **14**(4), Dec. 1992, 276-291.
5. F. Irigoin and F. Triolet, Supernode partitioning. In: *Conf. Rec. 15th ACM Symp. Princ. Progr. Lang.*, ACM Press, 1988, 319-329.
6. L. Lamport, The parallel execution of DO loops. In: *Comm. ACM* **17**(2), Febr. 1974, 83-93.
7. B.A. Malloy et al., Scheduling DAG's for asynchronous multiprocessor execution. In: *IEEE Trans. Par. Distr. Syst.* **5**(5), May 1994, 498-508.
8. W.F. McColl, General purpose parallel computing. In: A.M. Gibbons, P. Spirakis (eds.), *Lectures on Parallel Computation*, Cambridge Univ. Press, 1993, 337-391.
9. J. Ramanujam and P. Sadayappan, Tiling multidimensional iteration spaces for multicomputers. In: *J. Par. Distr. Comp.* **16**(2), Oct. 1992, 108-120.
10. R. Schreiber, J.J. Dongarra, Automatic blocking of nested loops. Technical Report 90-38, Univ. of Tennessee at Knoxville, May 1990.
11. L.G. Valiant, A bridging model for parallel computation. In: *Comm. ACM* **33**(8), Aug. 1990, 103-111.
12. M.E. Wolf and M.S. Lam, A loop transformation theory and an algorithm to maximize parallelism. In: *IEEE Trans. Par. Distr. Syst.* **2**(4), Oct. 1991, 452-470.
13. M.J. Wolfe, *Optimizing Supercompilers for Supercomputers.* Research Monographs in Parallel and Distributed Computing, MIT Press, 1989.