Preprint BUW-SC 03/3

Bergische Universität Wuppertal

Fachbereich C – Mathematik und Naturwissenschaften

Mathematik

Holger Arndt

Load Balancing: Dimension Exchange on Product Graphs

November 2003

http://www.math.uni-wuppertal.de/SciComp/

Load Balancing: Dimension Exchange on Product Graphs

Holger Arndt

Abstract

Load balancing on parallel computers aims at equilibrating some initial load which is initially different from one processor to another. Nearest neighbour load balancing algorithms can be divided basically into two classes: diffusion and dimension exchange. Whereas the first is appropriate for the so-called all-port-model where a processor can send tokens to all its neighbours at a time, the latter relies on the one-port-model.

In the last few years finite diffusion algorithms for general graphs as well as for product graphs like grids and tori have been developed. Recently finite dimension exchange algorithms have been proposed by the author. In the present paper we will introduce one new diffusion and two new dimension-exchange schemes for product graphs. We will show that the latter two can achieve nearly minimal communication operations and therefore short run-times. Additionally some modifications of the algorithms will be presented that reduce the flows so that only very few load items are moved via longer paths than necessary.

1 Introduction

We consider the problem of load balancing on parallel computers. The problem is modeled by an undirected connected graph G with a given initial load on every node. We suppose that the load consists of a number of equally sized items. This load distribution is to be balanced by moving load items along the edges of the graph.

The nodes of our graph G always represent the processors of the parallel computer. If the processors are connected by some special network (e. g. in

a torus) the graph should correspond to this topology. If the machine has a network so that the communication between any two processors is equally fast, a graph should be chosen that allows fast load balancing. Unless the number of processors is a power of two and you can take a hypercube, a torus is often a good choice. All algorithms we study are based on nearest neighbour communication: All information and load items are exchanged only via the edges of G.

Parallel computers can be modeled in two ways according to their communication hardware. The first one is the so-called all-port-model where a processor can send messages to or receive messages from all its neighbours at the same time. In the more realistic one-port-model, the communication is restricted to pairs of processors at a time.

In correspondence to the two communication models load balancing algorithms can be divided into two classes, diffusion and dimension exchange (see [3, 5]).

Load balancing algorithms usually proceed in two phases. During the first phase only information about load numbers is exchanged and a flow along the edges is computed, but no load is moved. In the second phase a scheduling algorithm is applied that moves load items according to the flow. In this paper we will only look at the first phase.

A good load balancing procedure should have three properties: it should be fast (as every call of the load balancer interrupts some major calculation), it should be numerically stable and it should produce small flows so that no more load is moved than necessary. The algorithms described in this paper will be examined with respect to these three aims.

The paper is organized as follows. In Section 2 some basic notation is introduced. Section 3 summarizes finite diffusion and dimension exchange algorithms. Section 4 addresses algorithms for product graphs. After reviewing one of the algorithms from [7] new diffusion and dimension exchange schemes for product graphs are presented. Additionally some heuristic flow reduction techniques are described. Numerical results are given in Section 5. We conclude the paper in Section 6 with a short summary.

2 Definitions

Let $G = (V, E)$ be a connected undirected graph consisting of $n = |V|$ nodes and $N = |E|$ edges. On each node $i \in V$ we are given an initial load $w_i^0 \geq 0$.

The load balancing algorithm has to determine the vector $\overline{w}$ of balanced loads, $\overline{w} = \left(\frac{1}{n}\sum_{i=1}^{n} w_i^0\right) \cdot (1, \ldots, 1)^T$.

Let $\alpha \in (0,1)$ be a constant edge weight. Then the matrix $M^{\mathsf{Diff}} = \left(m_{ij}^{\mathsf{Diff}}\right) \in \mathbb{R}^{n \times n}$ is defined by

$$m_{ij}^{\mathsf{Diff}} = \begin{cases} 1 - \alpha \deg(i) & \text{if } i = j \\ \alpha & \text{if } \{i,j\} \in E \\ 0 & \text{else}\,. \end{cases}$$

M^{Diff} is called a *diffusion matrix* if all elements m_{ij}^{Diff} are non-negative. Let $A \in \{-1,0,1\}^{n \times N}$ be the *node-edge incidence matrix* of G having in each column exactly two non-zero entries 1 and -1 which represent the nodes incident to the corresponding edge. The directions of the edges can be chosen arbitrarily. Now the *Laplace matrix* of G is defined as

$$L^{\mathsf{Diff}} = AA^T\,.$$

Then the diffusion matrix can also be expressed as $M^{\mathsf{Diff}} = I - \alpha L^{\mathsf{Diff}}$.

Let $x \in \mathbb{R}^N$ be a flow on G. The direction of the flow is determined by the directions of the edges in the incidence matrix A. The flow x is called a *balancing flow* if

$$Ax = w^0 - \overline{w}\ .$$

It is unique if and only if the graph is acyclic.

At last we give the definition of *product graphs*. Let $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$ be connected undirected graphs. Then the product graph $G = G^{(1)} \times G^{(2)}$ with $G = (V, E)$ is defined by

$$V = V^{(1)} \times V^{(2)}$$

and

$$E = \left\{\left((u^{(1)}, u^{(2)}), (v^{(1)}, v^{(2)})\right) : \left(u^{(1)} = v^{(1)} \wedge (u^{(2)}, v^{(2)}) \in E^{(2)}\right) \vee \left((u^{(1)}, v^{(1)}) \in E^{(1)} \wedge u^{(2)} = v^{(2)}\right)\right\}.$$

Grids and tori can be interpreted as products of two paths resp. cycles.

3 Load Balancing on General Graphs

In this section we state the most important diffusion and dimension exchange algorithms for general graphs and some of their properties.

3.1 Diffusion Algorithms

In [4, 7] two finite load balancing algorithms were introduced. Both of them are diffusion algorithms and require the knowledge of eigenvalues related to the underlying graph. Let $\mu_1^{\mathsf{Diff}} > \mu_2^{\mathsf{Diff}} > \ldots > \mu_m^{\mathsf{Diff}}$ be the distinct eigenvalues of the diffusion matrix M^{Diff}. The eigenvalue $\mu_1^{\mathsf{Diff}} = 1$ belongs to the eigenvector $\overline{w}$. Analogously let $\lambda_1^{\mathsf{Diff}} = 0 < \ldots < \lambda_m^{\mathsf{Diff}}$ be the eigenvalues of the Laplace matrix L^{Diff} where $\lambda_i^{\mathsf{Diff}} = 1 - \alpha\mu_i^{\mathsf{Diff}}$.

The simpler of the two aforementioned algorithms, OPT [7], can then be expressed in the following way:

for $k = 1, \ldots, m-1$ **do**
$w^k = \left(I - \frac{1}{\lambda_{k+1}^{\mathsf{Diff}}} L^{\mathsf{Diff}}\right) w^{k-1}$
$x^k = x^{k-1} + \frac{1}{\lambda_{k+1}^{\mathsf{Diff}}} A^T w^{k-1}$
end for

After $m-1$ steps we have $w^{m-1} = \overline{w}$ and x^{m-1} is a balancing flow. Unfortunately there are often stability problems with OPT. In particular, if the eigenvalues are inappropriately ordered the intermediate errors can become very large. Best numerical results are achieved by orderings based on weighted Leja points [12, 6, 1, 2]. We consider only the weight functions $\omega_g(z) = |z|^g$. Rather good numerical results can be achieved by using the special cases $\omega_0(z) = 1$ and $\omega_1(z) = |z|$.

Nevertheless the other algorithm, OPS [4], should be preferred. It is based on a three-term-recurrence and achieves small bounds of the l_2-norms of the errors after each step. Before the load balancing can be started, some parameters α_i, β_i and γ_i have to be computed once for a given graph. We define an inner product for polynomials p, q by

$$\langle p, q \rangle := \sum_{j=2}^{m} \omega_j p(\mu_j) q(\mu_j)$$

with $\omega_j = 1 - \mu_j$. For $k = 0, \ldots, m-1$ the polynomials p_k are given as the (scaled and shifted) so-called kernel polynomials, see [8]. They satisfy $p_0(t) = 1$, $p_1(t) = \frac{1}{\gamma_1}\left[(\alpha_1 - t)\, p_0(t)\right]$, $p_k(t) = \frac{1}{\gamma_k}\left[(\alpha_k - t)\, p_{k-1}(t) - \beta_k p_{k-2}(t)\right]$, $k = 2, \ldots, m-1$ and $\alpha_k = \frac{\langle t p_{k-1}, p_{k-1}\rangle}{\langle p_{k-1}, p_{k-1}\rangle}$, $\beta_k = \gamma_{k-1}\frac{\langle p_{k-1}, p_{k-1}\rangle}{\langle p_{k-2}, p_{k-2}\rangle}$, $\gamma_1 = \alpha_1 - 1$, $\gamma_k = \alpha_k - 1 - \beta_k$. The values for the γ_k are chosen such that $p(1) = 1$ which guarantees that the total amount of load is unchanged. Once these values are computed, OPS can be applied as follows:

$w^1 = \frac{1}{\gamma_1}\left[\alpha_1 w^0 - M^{\mathsf{Diff}} w_0\right]$
$x^1 = -\frac{1}{\gamma_1}\alpha A^T w^0$
for $k = 2, \ldots, m-1$ **do**
$w^k = \frac{1}{\gamma_k}\left[\alpha_k w^{k-1} - M^{\mathsf{Diff}} w^{k-1} - \beta_k w^{k-2}\right]$
$x^k = \frac{1}{\gamma_k}\left[(\alpha_k - 1)\, x^{k-1} - \alpha A^T w^{k-1} - \beta_k x^{k-2}\right]$
end for

Both OPT and OPS have the property that they need exactly $m-1$ steps where m is the number of distinct eigenvalues of the graph. It was shown in [4] that this is a substantial improvement compared to other iterative but non-finite methods like FOS or SOS (first / second order scheme) [3, 11].

A balancing flow for a given initial load distribution is usually not unique — unless the graph is a tree. It is known that the balancing flow produced by any diffusion algorithm (including OPS and OPT) has minimal l_2-norm [4, 10]. Such flows will be called *minimal flows* in the sequel.

3.2 Dimension Exchange

Whereas the diffusion algorithms shown so far are designed for the all-port-model, most parallel computers rely on the one-port-model. Therefore dimension exchange algorithms are more appropriate because in this case each step of the algorithm is divided into substeps — one substep per neighbour. In each substep the most recent load value (from the last substep) is then used. Dimension exchange was first suggested for the hypercube by Cybenko in [3], edge-colourings of graphs have been proposed in [9] and [13] and finite dimension exchange procedures have been introduced in [1] and [2].

An edge-colouring of the graph is needed for the definition of substeps. The edge-set E is divided into c disjoint non-empty sets E_i so that $E = E_1 \cup \ldots \cup E_c$ and for any pair of edges incident with one common node it holds that they are in different sets E_i and E_j.

The edge-colouring of G induces subgraphs $G_i = (V, E_i)$. The diffusion and Laplace matrices of these subgraphs are denoted by M_i and L_i.

Before we can precisely formulate dimension exchange algorithms we first have to introduce some additional notation:

$$M^{\mathsf{DE}} = M_c \cdots M_1$$
$$L^{\mathsf{DE}} = \frac{1}{\alpha}\left(I - M^{\mathsf{DE}}\right)$$

Here DE stands for dimension exchange. A multiplication of M^{DE} by a load vector w means precisely that c substeps are applied to w and the most recent information is used for each substep. In addition we will need the eigenvalues μ^{DE} and λ^{DE} of the matrices M^{DE} and L^{DE} respectively.

Next we need the incidence matrices $A_i \in \{-1, 0, 1\}^{n \times N}$ of the G_i. In contrast to the usual definition of an incidence matrix we do not delete columns which are not coloured with colour i from A but we replace the 1's and -1's in those columns with zeros.

Now we can construct new algorithms DE-OPT and DE-OPS as follows. We use the same iteration as for OPT resp. the same polynomials as for OPS, but we replace the matrices M^{Diff} resp. L^{Diff} by their dimension exchange counterparts M^{DE} and L^{DE}. Of course we also have to replace the eigenvalues by those of the new matrices. We start with the DE-OPT algorithm which computes $w^k = \left(I - \frac{1}{\lambda^{\mathsf{DE}}_{k+1}} L^{\mathsf{DE}}\right) w^{k-1}$. To show the substeps explicitely we rewrite this algorithm as follows:

for $k = 1, \ldots, m-1$ **do**
  $\hat{w}^0 = w^{k-1}$
  for $j = 1, \ldots, c$ **do** *{loop over the colours}*
    $\hat{w}^j = M_j \hat{w}^{j-1}$
    $x = x + \frac{1}{\lambda^{\mathsf{DE}}_{k+1}} A_j^T \hat{w}^{j-1}$
  end for
  $w^k = \left(1 - \frac{1}{\alpha \lambda^{\mathsf{DE}}_{k+1}}\right) w^{k-1} + \frac{1}{\alpha \lambda^{\mathsf{DE}}_{k+1}} \hat{w}^c$
end for

Finally we give the substep-version of DE-OPS.

$\hat{w}^0 = w^0$; $\hat{x}^0 = 0$
for $j = 1, \ldots, c$ **do** *{loop over the colours}*
  $\hat{w}^j = M_j \hat{w}^{j-1}$; $\hat{x}^j = \hat{x}^j + \alpha A_j^T \hat{w}^{j-1}$
end for
$w^1 = \frac{1}{\gamma_1} [\alpha_1 w^0 - \hat{w}^c]$; $x^1 = -\frac{1}{\gamma_1} \hat{x}^c$
for $k = 2, \ldots, m-1$ **do**
  $\hat{w}^0 = w^{k-1}$; $\hat{x}^0 = x^{k-1}$
  for $j = 1, \ldots, c$ **do** *{loop over the colours}*
    $\hat{w}^j = M_j \hat{w}^{j-1}$; $\hat{x}^j = \hat{x}^j + \alpha A_j^T \hat{w}^{j-1}$
  end for

$w^k = \frac{1}{\gamma_k}\left[\alpha_k w^{k-1} - \hat{w}^c - \beta_k w^{k-2}\right]$
$x^k = \frac{1}{\gamma_k}\left[\alpha_k x^{k-1} - \hat{x}^c - \beta_k x^{k-2}\right]$
end for

The edge weight α should always be chosen equal to $\frac{1}{2}$ because this usually results in the least number of different eigenvalues and therefore lowest run-times, cf. [2, 1]. This factor is assumed in all tables and figures in the sequel.

3.3 Comparison of Diffusion and Dimension Exchange

We summarize the most important results from [2]. Table 1 shows that dimension exchange often needs less communication steps than diffusion and is therefore faster. The diameter of the graph is a lower bound for the number of steps because the information about the initial load distribution has to be transported through the entire graph. This minimal number is (nearly) achieved by dimension exchange for paths, cycles of even lengths and hypercubes.

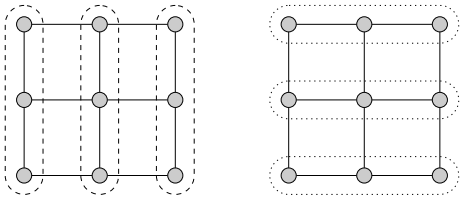
Whereas diffusion can have numerical problems for certain graphs, dimension exchange almost always converges well. The flows computed by dimension exchange are not minimal but it could be shown in [1, 2] that they are bounded.

4 Load Balancing on Product Graphs

From Table 1 one can see that for grids and tori the run-time of all algorithms differ from the minimum by a factor of order n. In this section algorithms with (nearly) optimal run-times will be introduced. We will restrict ourselves to square product graphs, but all algorithms can easily be adopted to the situation of two different factor graphs. The OPT-variants are even applicable to higher dimensional products.

4.1 Existing Approaches

If we are given a product graph the following strategy can always be used. First apply some load balancing algorithm to all copies of the first factor graph and after that balance along the second direction which results in a globally balanced state.



Unfortunately this approach yields too large flows. This problem can be reduced by applying the load balancing steps alternatingly to both directions.

Let $G = G^{(1)} \times G^{(1)}$ be a square product graph, $L^{\mathsf{Diff}^{(1)}}$ the Laplace matrix of $G^{(1)}$ and let $\lambda_1^{(1)}, \ldots, \lambda_m^{(1)}$ denote the eigenvalues of $L^{\mathsf{Diff}^{(1)}}$. The following ADI-OPT (ADI = alternating direction iteration) procedure has been introduced in [7].

for $k = 1, \ldots, m-1$ **do**

$$w^k = \left(I - \tfrac{1}{\lambda_{k+1}^{(1)}} L^{\mathsf{Diff}^{(1)}} \otimes I\right)\left(I - \tfrac{1}{\lambda_{k+1}^{(1)}} I \otimes L^{\mathsf{Diff}^{(1)}}\right) w^{k-1}$$

$$x^k = x^{k-1} + \tfrac{1}{\lambda_{k+1}^{(1)}}\left[\left(I \otimes A^{(1)^T}\right) + \left(A^{(1)^T} \otimes I\right)\left(I - \tfrac{1}{\lambda_{k+1}^{(1)}} I \otimes L^{(1)}\right)\right] w^{k-1}$$

end for

Even better flows are achieved by a slight modification called MDI-OPT (MDI = mixed direction iteration) in [7].

4.2 A New Diffusion Algorithm: ADI-OPS

It is known that OPS is numerically more stable than OPT. Thus for product graphs it would be desirable to have an ADI-OPS. Because OPS uses the last two iterates in each but the first step the construction of an ADI-version becomes more complicated and needs additional intermediate load values. We will now use double indices for load vectors. The indices r, c (row / column) indicate from which direction a new value is computed. The parameters $\alpha_k^{(1)}$, $\beta_k^{(1)}$ and $\gamma_k^{(1)}$ are associated with the factor graphs.

$$w^{0,0} = w^0$$

$$w_r^{k,1} = \frac{1}{\gamma_1^{(1)}}\left(\alpha_1^{(1)} I - M^{(1)} \otimes I\right) w^{k,0} \qquad k = 0, \ldots$$

$$w_c^{1,l} = \frac{1}{\gamma_1^{(1)}} \left(\alpha_1^{(1)} I - I \otimes M^{(1)} \right) w^{0,l} \qquad l = 0, \ldots$$

$$w_r^{k,l} = \frac{1}{\gamma_l^{(1)}} \left(\left(\alpha_l^{(1)} I - M^{(1)} \otimes I \right) w^{k,l-1} - \beta_l^{(1)} w^{k,l-2} \right) \quad l = 2, \ldots; k = 0, \ldots$$

$$w_c^{k,l} = \frac{1}{\gamma_k^{(1)}} \left(\left(\alpha_k^{(1)} I - I \otimes M^{(1)} \right) w^{k-1,l} - \beta_k^{(1)} w^{k-2,l} \right) \quad k = 2, \ldots; l = 0, \ldots$$

By induction on $k + l$ the following result can be shown:

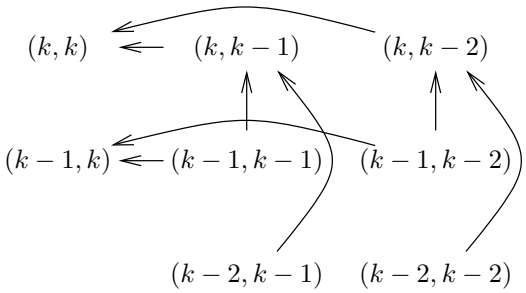
Theorem 1. *For the load vectors defined above it holds that* $w_r^{k,l} = w_c^{k,l} =: w_{k,l}$ *for* $k, l \geq 1$.

With this theorem convergence can easily be proven.

Corollary 2. *The vector* $w^{m-1,m-1}$ *equals the vector of balanced load* $\overline{w}$.

Proof. According to the last theorem it is irrelevant via which $w^{i,j}$'s the vector $w^{m-1,m-1}$ is computed. In particular we can choose the "sequential directions path" $w^{0,0}, w^{0,1}, \ldots, w^{0,m-1}, w^{1,m-1}, \ldots, w^{m-1,m-1}$. □

The path from the last proof generates unnecessarily large flows. Instead of that, in each but the first step we compute four new values $w^{k,k-2}$, $w^{k,k-1}$, $w^{k-1,k}$ and $w^{k,k}$ using four old values $w^{k-2,k-2}$, $w^{k-1,k-2}$, $w^{k-2,k-1}$ and $w^{k-1,k-1}$.



The number of communication operations per step is not higher than for ADI-OPT as the data needed for the computation of $w^{k,k-2}$ and $w^{k,k-1}$ can be sent together as well as those for $w^{k-1,k}$ and $w^{k,k}$.

The complete ADI-OPS algorithm is shown below.

$w11 = w^0$
$w01 = \frac{1}{\gamma_1} \left(\alpha_1 I - I \otimes M^{(1)} \right) w11$**;** $x01 = -\frac{1}{\gamma_1} \alpha \left(I \otimes A^{(1)^T} \right) w01$

$w10 = \frac{1}{\gamma_1}\left(\alpha_1 I - M^{(1)} \otimes I\right) w11$; $x10 = -\frac{1}{\gamma_1}\alpha\left(A^{(1)^T} \otimes I\right) w10$
$w = \frac{1}{\gamma_1}\left(\alpha_1 I - M^{(1)} \otimes I\right) w01$; $x = -\frac{1}{\gamma_1}\left[(\alpha_1 - 1)\, x01 - \alpha\left(A^{(1)^T} \otimes I\right) w01\right]$
for $k = 2, \ldots, m-1$ **do**
$w22 = w11$; $w12 = w01$; $w21 = w10$; $w11 = w$
$x22 = x11$; $x12 = x01$; $x21 = x10$; $x11 = x$
$w02 = \frac{1}{\gamma_k}\left[\left(\alpha_k I - I \otimes M^{(1)}\right) w12 - \beta_k w22\right]$
$x02 = \frac{1}{\gamma_k}\left[(\alpha_k - 1)\, x12 - \alpha\left(I \otimes A^{(1)^T}\right) w12 - \beta_k x22\right]$
$w01 = \frac{1}{\gamma_k}\left[\left(\alpha_k I - I \otimes M^{(1)}\right) w11 - \beta_k w21\right]$
$x01 = \frac{1}{\gamma_k}\left[(\alpha_k - 1)\, x11 - \alpha\left(I \otimes A^{(1)^T}\right) w11 - \beta_k x21\right]$
$w10 = \frac{1}{\gamma_k}\left[\left(\alpha_k I - M^{(1)} \otimes I\right) w11 - \beta_k w12\right]$
$x10 = \frac{1}{\gamma_k}\left[(\alpha_k - 1)\, x11 - \alpha\left(A^{(1)^T} \otimes I\right) w11 - \beta_k x12\right]$
$w = \frac{1}{\gamma_k}\left[\left(\alpha_k I - M^{(1)} \otimes I\right) w01 - \beta_k w02\right]$
$x = \frac{1}{\gamma_k}\left[(\alpha_k - 1)\, x01 - \alpha\left(A^{(1)^T} \otimes I\right) w01 - \beta_k x02\right]$
end for

4.3 Dimension Exchange on Product Graphs

As for the general case, dimension exchange algorithms for product graphs can simply be generated from diffusion algorithms by replacing the respective matrices and eigenvalues, e. g. by taking $L^{\mathsf{DE}^{(1)}}$ instead of $L^{\mathsf{Diff}^{(1)}}$. The new algorithms are called DE-ADI-OPT and DE-ADI-OPS. We show only the first algorithm where c is the number of colours of each factor graph and the λ_k are the eigenvalues of $L^{\mathsf{DE}^{(1)}}$.

for $k = 1, \ldots, m-1$ **do**
$\hat{w}^0 = w^{k-1}$
for $j = 1, \ldots, c$ **do** *{loop over the colours of* $G^{(1)}$*}*
$\hat{w}^j = (I \otimes M_j)\, \hat{w}^{j-1}$
$x = x + \frac{1}{\lambda_{k+1}}\left(I \otimes A_j^T\right) \hat{w}^{j-1}$
end for
$w^{k-\frac{1}{2}} = \left(1 - \frac{1}{\alpha\lambda_{k+1}}\right) w^{k-1} + \frac{1}{\alpha\lambda_{k+1}}\hat{w}^c$
$\tilde{w}^0 = w^{k-\frac{1}{2}}$
for $j = 1, \ldots, c$ **do** *{loop over the colours of* $G^{(1)}$*}*
$\tilde{w}^j = (M_j \otimes I)\, \tilde{w}^{j-1}$

$x = x + \frac{1}{\lambda_{k+1}} \left(A_j^T \otimes I\right) \tilde{w}^{j-1}$
end for
$w^k = \left(1 - \frac{1}{\alpha\lambda_{k+1}}\right) w^{k-\frac{1}{2}} + \frac{1}{\alpha\lambda_{k+1}} \tilde{w}^c$
end for

4.4 Comparison of Diffusion and Dimension Exchange on Product Graphs

It can be seen from Table 2 that the combination of ADI and dimension exchange yields a minimal or at least nearly minimal number of communication steps on grids and on tori of even dimension. An example for the convergence of different ADI- and non-ADI-algorithms is shown in Figure 1.

4.5 Flows for ADI algorithms

The flows computed by any ADI algorithm are not l_2-minimal, neither for diffusion nor dimension exchange — but there are several techniques that can reduce the flows.

The whole computation can be applied twice: Once the standard ADI-algorithm is used (starting each step with direction one and considering the second direction afterwards) and once starting with the second direction. This results in two different balancing flows. The average of these two is also a balancing flow and generally has a lower l_2-norm than the two original flows. The computational costs of course increase by a factor 2, but the dominating number of communication steps just increases by the substeps for one direction if both computations are done nearly at the same time – shifted by a half step. The following figure shows the sequence of the steps affecting the horizontal and vertical factor graphs.

computation 1:	H	V	H	V	H	V	
computation 2:		V	H	V	H	V	H

The resulting methods will be denoted by (DE-)ADC-OPS/T. The following two modifications can be combined with this technique.

Whereas the flows generated by (DE-)OPT are invariant under different orderings of the eigenvalues, they are highly dependent on this ordering for (DE-)ADI-OPT. Generally Leja orderings should be used because of stability

reasons. It can be observed by experiments that the exponent g in the weight function $\omega_g(z) = |z|^g$ has two different impacts. If g is increased the norm of the flow decreases whereas the error $e^{m-1} = w^{m-1} - \overline{w}$ after the last step increases, cf. Figure 2. A value of $g = 1.5$ seems reasonable for practical situations.

In (DE-)OPS the $\omega_j = 1 - \mu_j$ were chosen such that the error after each step is as small as possible. The fact that the error becomes zero after $m-1$ steps is independent of this choice. Whereas the flows are independent of the ω_j for (DE-)OPS they can be reduced for ADI-(DE-)OPS by using ω_j^η, $\eta > 1$ instead. Thus the new inner product is

$$\langle p, q \rangle = \sum_{j=2}^{m} \omega_j^\eta p(\mu_j) q(\mu_j) \, .$$

Experiments show that the flows decrease and the final errors increase if η is increased, cf. Figure 3. In practice, values for η between 3 and 4 yielded satisfying results.

5 Numerical Results

Table 3 shows results of numerical experiments done on a cluster of Linux-PCs. The initial load consisted of 6400 randomly distributed load items. The times are scaled such that the simplest algorithm, an OPT-implementation using synchronous MPI communication, equals 100. All other diffusion algorithms implement faster asynchronous communication, dimension exchange is inherently synchronous.

It is impossible to achieve minimal run times and minimal flows at the same time, but the two algorithms DE-ADC-OPT and DE-ADC-OPS are quite close to this optimum. Usually the first is the faster one and produces slightly lower flows.

6 Summary

We have presented several new load balancing algorithms designed for product graphs like grids and tori. The algorithms DE-ADC-OPT and DE-ADC-OPS are nearly time-optimal, produce flows which are usually only a few percent higher than the l_2-minimal flow and are numerically stable.

References

[1] H. Arndt. *Loadbalancing auf Parallelrechnern mit Hilfe endlicher Dimension-Exchange-Verfahren.* PhD thesis, Bergische Universität Wuppertal, Fachbereich Mathematik, July 2003.

[2] H. Arndt. On finite dimension exchange algorithms. *Linear Algebra and its Applications*, 2003. to appear.

[3] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[4] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25:789–812, 1999.

[5] R. Diekmann, B. Monien, and R. Preis. Load balancing strategies for distributed memory machines. In B. Topping, editor, *Parallel and Distributed Processing for Computational Mechanics: Systems and Tools*, pages 124–157. Saxe-Coburg Publications, 1999.

[6] R. Elsässer. *Spectral Methods for Efficient Load Balancing Strategies.* PhD thesis, Universität Paderborn, Fachbereich Mathematik / Informatik, Feb. 2002.

[7] R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction loadbalancing schemes. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *Euro-Par'99 Parallel Processing*, Lecture Notes in Computer Science, No. 1685, pages 280–290. Springer-Verlag, 1999.

[8] B. Fischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems.* Wiley-Teubner series in advances in numerical mathematics. Wiley-Teubner, Chichester, Stuttgart, 1996.

[9] S. H. Hosseini, B. Litow, M. I. Malkawi, J. McPherson, and K. Vairavan. Analysis of a graph coloring based distributed load balancing algorithm. *Journal of Parallel and Distributed Computing*, 10:160–166, 1990.

[10] Y. F. Hu and R. J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25:417–444, 1999.

[11] S. Muthukrishnan, B. Ghosh, and M. H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *SPAA '96. Proceedings of the 8th annual ACM symposium on Parallel algorithms and architectures*, pages 72–81, 1996.

[12] L. Reichel. The application of Leja points to Richardson iteration and polynomial preconditioning. *Linear Algebra and its Applications*, 154–156:389–414, 1991.

[13] C. Xu and F. C. M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16(4):385–393, Dec. 1992.

Figures and Tables

Graph			diameter	OPS/T	DE-OPS/T
path	P_n	$2 \mid n$	$n-1$	$2n-2$	n
cycle	C_n	$4 \mid n$	$\frac{1}{2}n$	n	$\frac{1}{2}n$
		$3 \mid n$	$\frac{1}{2}n-\frac{1}{2}$	n	$2n$
grid	G_n	$2 \mid n$	$2n-2$	$2n^2$	$\frac{1}{2}n^2+\mathcal{O}(n)$
torus	T_n	$4 \mid n$	n	$\frac{1}{2}n^2+\mathcal{O}(n)$	$\frac{1}{8}n^2+\mathcal{O}(n)$
hypercube	H_d		d	d^2	d

Table 1: Number of communication steps for some graphs and algorithms

Graph			diameter	ADI-OPS/T	DE-ADI-OPS/T
grid	G_n	$2 \mid n$	$2n-2$	$4n-4$	$2n$
		$2 \nmid n$	$2n-2$	$4n-4$	$2n+2$
torus	T_n	$4 \mid n$	n	$2n$	n
		$2 \mid k$, $4 \nmid k$	n	$2n$	$n+2$

Table 2: Number of communication steps for ADI on grids and tori

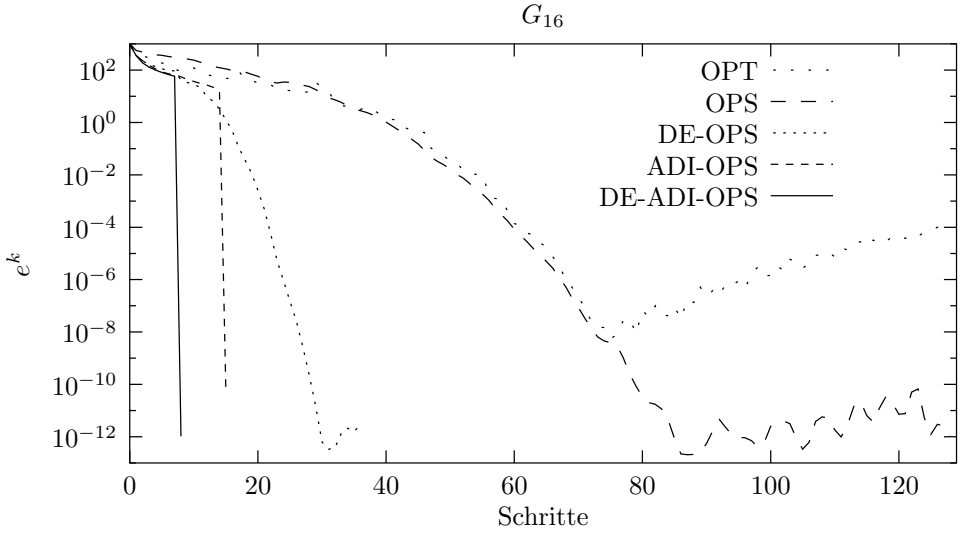


Figure 1: Convergence of ADI- and non-ADI-algorithms for the grid G_{16}

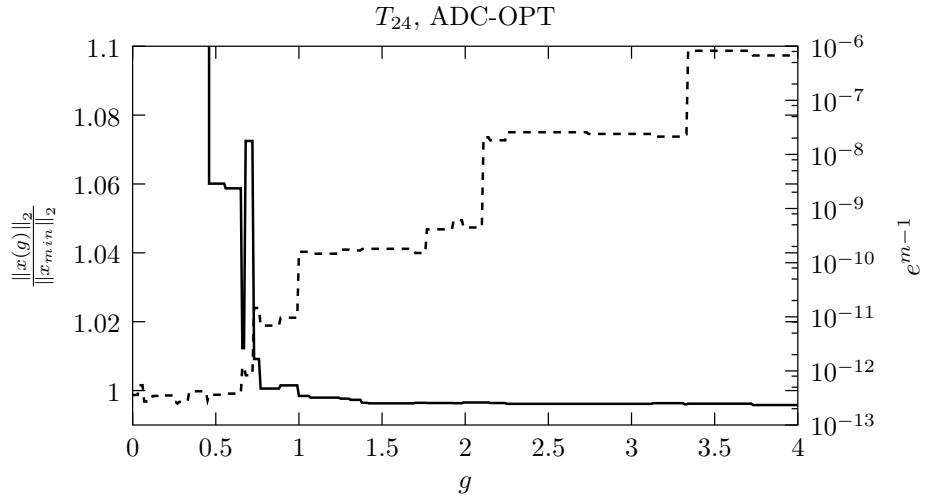


Figure 2: Dependency of ADC-OPT using Leja ordering on the exponent g of the weight function $\omega_g(z) = |z|^g$. It is shown, by which factor the flow is above the minimum (solid line, left axis) and the final error e^{m-1} (dashed line, right axis)

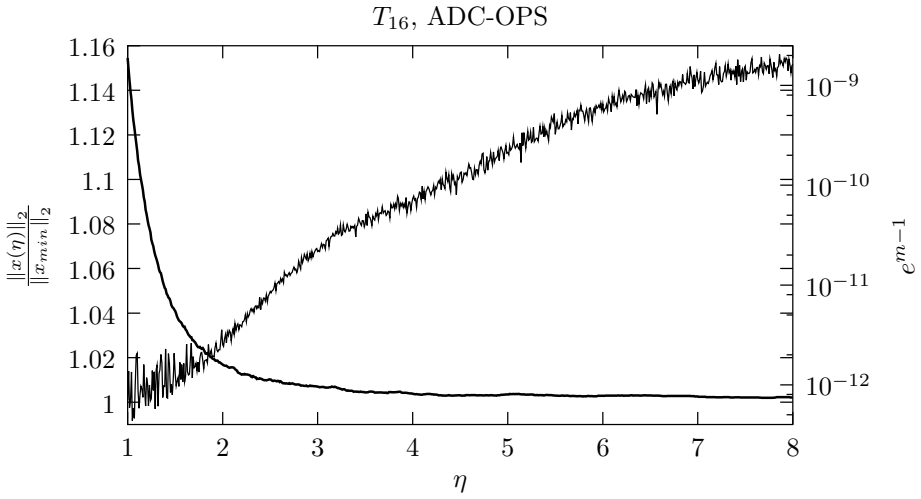


Figure 3: Dependency of ADC-OPS on η: It is shown, by which factor the flow is above the minimum (thick line, left axis) and the final error e^{m-1} (thin line, right axis)

algorithm	grid G_8		torus T_8	
	time	$\frac{\|x\|_2}{\|x_{\min}\|_2}$	time	$\frac{\|x\|_2}{\|x_{\min}\|_2}$
OPT (unopt)	100	1	100	1
OPS	79	1	90	1
DE-OPT	34	1.14	25	1.23
MDI-OPT, $g = 1.5$	23	1.02	37	1.03
ADC-OPT, $g = 1.5$	22	≈ 1	39	≈ 1
ADC-OPS, $\eta = 4$	25	1.006	40	1.005
DE-ADI-OPT, $g = 1.5$	16	1.16	21	1.24
DE-ADC-OPT, $g = 1.5$	16	1.037	22	1.059
DE-ADI-OPS, $\eta = 4$	16	1.20	21	1.28
DE-ADC-OPS, $\eta = 4$	19	1.047	24	1.07

Table 3: Results for the grid $G_8 = P_8 \times P_8$ and the torus $T_8 = C_8 \times C_8$