CONCURRENCY: PRACTICE AND EXPERIENCE, VOL. 2(4), 289–313 (DECEMBER 1990)

Load balancing and Poisson equation in a graph

J.E. BOILLAT

Institute for Informatics and applied Mathematics
University of Berne
Länggassstrasse 51
CH-3012 Berne, Switzerland

SUMMARY

We present a fully distributed dynamic load balancing algorithm for parallel MIMD architectures. The algorithm can be described as a system of identical parallel processes, each running on a processor of an arbitrary interconnected network of processors. We show that the algorithm can be interpreted as a Poisson (heath) equation in a graph. This equation is analysed using Markov chain techniques and is proved to converge in polynomial time resulting in a global load balance. We also discuss some important parallel architectures and interconnection schemes such as linear processor arrays, tori, hypercubes, etc. Finally we present two applications where the algorithm has been successfully embedded (process mapping and molecular dynamic simulation).

1. INTRODUCTION

To achieve a high performance from a parallel computer it is imperative to balance the processor's work-load. The balancing problem must take two aspects into consideration:

- *the underlying architecture of the system used.* This includes the topology of the multi-processor system and the processor interconnections. In the simplest case there is a predefined, fixed and usually homogeneous topology, e.g. a hypercube scheme[1]—more flexible systems allow a dynamic or static reconfiguration of the interprocessor connections[2]. Clearly if the topology is fixed and homogeneous during a computation the load balancing problem will be easier to solve[3].
- *the application to be run on the target system.* The load balancing problem is relatively easy to solve in the case of a homogeneous computational problem—i.e. where each member of the underlying data domain is associated with a similar complexity in time (i.e. calculation time), and where the member distribution in the domain does not change during the computation (i.e. a static data distribution). For problems where, either the data distribution or the member/time complexity, or any change during the computation, more complex methods have to be devised: these could *dynamically* reallocate the work-load of the processors in order to achieve a well-balanced overall load of the system and thus high efficiency.

In this paper a new approach is described to solve the load balancing problem for parallel programs. We present a fully distributed load balancing algorithm, consisting of the same process running in parallel on each processor of a given network. No assumption has to be made concerning the structure of the underlying network. We prove that local exchange of load can result in global load balancing in polynomial time.

1040-3108/90/040289-25$12.50

Received August 1989
Revised June 1990

2. LOAD BALANCING ALGORITHM

To achieve the best possible overall performance of a parallel system over the entire duration of the application a load balancer process is usually run concurrently with the main application[4]. This feature is generally assumed as essential for the design of a dynamic load balancer[5], which then can be implemented according to many different schemes[4]. In most cases, load balancing is achieved by optimizing a global cost function, i.e. the load balancing process needs to know information from all processors in order to make balancing decisions. This usually results in a centralistic algorithm which causes excessive synchronization on MIMD systems with a very large number of processors, resulting in poor efficiency. These approaches contradict the philosophy of developing distributed applications with as little global interaction as possible. Furthermore, it has been shown that simple load balancing policies yield high performance at low costs[6].

Considering the above, we have investigated a new approach to load balancing:

1. A load balancer should be distributed among all the processors of a given network.
2. The interactions between the distributed parts of the load balancer should be as local as possible.
3. For reasons of simplicity, we developed a load balancer consisting of the same procedure running on all the processors.
4. The synchronization of the load balancing processes is restricted to direct neighbours.
5. Every load balancing procedure decides whether to give load to the neighbouring processors depending only on local information, i.e. by comparing the load of the underlying processor with the load of the neighbouring processors and thereby trying to reach a *local* equilibrium.

After a number of diffusion cycles, which in the worst case is quadratic to the number of processors, a *global* equilibrium of the load will be reached. This approach was suggested from the heat diffusion phenomenon from physics, in particular from discrete numeric schemes for solving the Poisson equation.

The proposed load balancing algorithm has been implemented and tested in two different applications:

1. a distributed mapping algorithm[3]
2. a distributed molecular dynamics simulation algorithm[7]

which will be discussed briefly in Section 6.

The load balancing algorithm has exactly the same structure as the underlying architecture. It consists of the same process running on all processors of the machine. Each process is a loop consisting of a synchonization and information exchange step and a computation step. We call programs consisting of the same procedure running on all the processors of a network Single Program Multiple Data (SPMD) programs, analogous to the classification of Flynn for parallel computers[8,9]. Every procedure of a SPMD program has the **occam** pseudocode form shown in Figure 1.

In most cases a load balancer has to balance the load of a system of communicating processes. An arbitrary balancing strategy may result in high communication costs between the distributed processes, and for this reason the processes to be exchanged

```
WHILE run
  SEQ
    ... exchange informations with direct neighbours
    ... compute one step
```

Figure 1: Typical procedure of an SPMD application

must satisfy additional communication costs criteria. In Reference 3 we show that a suboptimum of the communication costs may be reached by optimizing local costs (running the load balancer together with a communication strategy always leads to a global load balancing and communication cost suboptimum (see also Section 6)). In this paper we concentrate on the load balancing problem and leave the communication optimization.

For reasons of model consistency (see Sections 2.1 and 3), we assume that the domain of the load balancer consists in a very large number of processes, and furthermore that all processes of the underlying domain have the same load. This last condition may appear to be restrictive but is necessary for proving the convergence of the algorithm. Since there is a very large number of processes (i.e. the ratio of processors to processes is small), we consider the load of each processor as a real number—the sum of the process loads running on each processor. Thus we may ignore the number of processes running on each processor. Tests have shown that the load balancer behaves well if the processes have true discrete loads, provided there are many more processes than processors[3].

Remark 1.

A discrete modelling of the problem, i.e. working with individual processes, would lead to cellular automata over graphs[10,11].

2.1 Formal description of the load balancing problem

Formally a MIMD system can be represented by a graph $H = (T,E)$, T the set of processors, and E the set of interprocessor links. We will assume that the processors have a maximal number (`max.links`) of interconnection links. Let P be the set of the processes to be distributed among the processors.

- A *mapping* is a function
$$\pi : P \longrightarrow T$$
assigning each process to a processor.
- The *load* of the processor i relative to a mapping π is defined as
$$\text{load}_\pi(i) = \sum_{\pi(\mu)=i} \text{time}(\mu)$$
where $\text{time}(\mu)$ denotes the time complexity of the process μ.
- A global cost function $\Gamma(\pi)$ may be defined as

$$\Gamma(\pi) = \sum_{i \in T} (\bar{l} - \text{load}_\pi(i))^2$$

where

$$\bar{l} = \frac{\sum_{i \in T} \text{load}_\pi(i)}{|T|}$$

The load balancing problem can be defined as follows:

Find a mapping π which minimizes the global cost function $\Gamma(\pi)$

We show that the global minimum can be reached by the parallel optimization of the local load, i.e. by locally optimizing

$$\Gamma_i(\pi) = \sum_{(i,j) \in E} (\text{load}_\pi(i) - \text{load}_\pi(j))^2 \qquad \forall i \in T$$

2.2 Load balancing algorithm

The proposed algorithm is based on the discrete Poisson equation in a graph, i.e. the network graph of the MIMD system. The algorithm consists in the process (Figure 2) running in parallel on each processor of the machine.

One of the main problems with highly parallel algorithms is that of convergence and complexity. We will assume that the load balancing process does not change the load of the processors, i.e. computing and searching the processes to be moved and moving the processes is costless with respect to the load of the processes. Under this hypothesis, the proposed load balancing algorithm is equivalent to a discrete Poisson equation in a graph.

3. DISCRETE POISSON EQUATION IN A GRAPH

3.1. Notation

In this paper, a graph $G = (N,E)$ is a non directed, connected finite graph without loops. We will denote the vertices of a graph G by numbers ($N = \{1,2,...,n\}$) and the adjacency matrix of G by A_G. The valency of the vertex i (i.e. the number of edges incident to i), will be denoted by v_i.

3.2. Laplacean operator in a graph

A *valuated graph* is a graph $G = (N,E)$ to which each edge (i,j) a positive number $c_{ij} = c_{ji}$ is associated. According to Fiedler[12], the *Laplacean operator* Δ_G^C of a valuated graph $G = (N,E)$ with n vertices, is defined as the matrix of the quadratic form

$$\langle \Delta_G^C x, x \rangle = \sum_{(i,j) \in E} c_{ij}(x_i - x_j)^2$$

```
--------------------------------------------------------------------
PROC load.balancer ([]CHAN OF p in,       -- channels from neighbour
                    []CHAN OF p out,      -- channels to neighbours
                    VAL INT processors,   -- number of processors
                    VAL INT id)           -- processor identifier
  ------------------------------------------------------------------
  INT links                   : -- # active links
  INT iterations              : -- # of iterations
  [max.links]INT weight       : -- valuation of the links
  [max.links]REAL receive     : -- processes received from neighbours
  [max.links]REAL give        : -- processes given to neighbours
  [max.links]REAL nb.load     : -- load of the neighbours
  REAL my.load                : -- load of processor
  ------------------------------------------------------------------
  SEQ
    ... initialize iterations, weight, my.load and give
    SEQ i = 0 FOR iterations
      SEQ
        ... exchange load information with direct neighbours
        ... iteration step of application
        VAL init.load IS my.load                :
        SEQ j = 0 FOR links
          VAL to.give IS (init.load - nb.load[j]) / weight[j] :
          IF
            to.give > 0.0
              SEQ
                my.load := my.load - to.give
                give[j] := to.give
            TRUE
              give[j] := 0.0
        ... exchange processes with direct neighbours
        ... update load
  ------------------------------------------------------------------
:
--------------------------------------------------------------------
```

Figure 2. Load balancing algorithm

i.e. $\Delta_G^C = (\delta_{ij})$, where

$$\delta_{ij} = \delta_{ji} = -c_{ij} \qquad \text{if } (i,j) \in E$$
$$\delta_{ij} = \delta_{ji} = 0 \qquad \text{if } i \neq j \text{ and } (i,j) \notin E$$
$$\delta_{ii} = -\sum_{(i,k)\in E} c_{ik}$$

Let $\mathbf{e}$ be the vector $(1, 1, \ldots, 1)^T$. Note that since $\Delta_G^C \mathbf{e} = 0$ and Δ_G^C is positive semi-definite, zero is the smallest eigenvalue of Δ_G^C. Since G is connected, Δ_G^C is irreducible and $\mathbf{e}$ is the only linearly independent solution of $\langle \Delta_G^C x, x \rangle = 0$ i.e. 0 is a simple eigenvalue of the Laplacean.

3.3. Poisson equation in a graph

Let $G = (N, E)$ be a graph. The valuation

$$c_{ij} = \frac{1}{\max(v_i, v_j) + 1} \qquad \text{if } (i,j) \in E$$

is called the canonical valuation of G.

The time-discrete Poisson equation in a graph $G = (N, E)$ with canonical valuation $\mathbf{C}$ is defined as

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{x}(t+1) - \mathbf{x}(t) = -\Delta_G^C \mathbf{x}(t)$$

The Poisson operator associated with the graph $G = (N, E)$ is defined as the matrix

$$\mathbf{P}_G = \mathbf{I} - \Delta_G^C$$

where $\mathbf{I}$ is the identity matrix. Thus a solution $\mathbf{x}$ of the Poisson equation is a fixed point of the following equation:

$$\mathbf{x} = \mathbf{P}_G \mathbf{x}$$

In this Section, we show that the iteration

$$\mathbf{x}(t) = \mathbf{P}_G^t \mathbf{x}(0) \tag{1}$$

where $\mathbf{x}(0) \geq 0$, converges towards

$$\mathbf{x} = \frac{1}{n}\mathbf{e} \tag{2}$$

i.e. the Poisson equation with initial condition $\mathbf{x}(0) \geq 0$, $\sum \mathbf{x}(0)_i = 1$, has a unique solution for every graph G. The solution is the uniform distribution.

Lemma 1. $\mathbf{P}_G$ is a non-negative matrix.

Proof: $p_{ii} = 1 - \sum_{(i,j)\in E} \frac{1}{\max(v_i, v_j)+1} \geq 1 - \sum_{(i,j)\in E} \frac{1}{v_i+1} \geq 1 - \frac{v_i}{v_i+1} > 0$. □

Lemma 2. $\mathbf{P}_G$ is an irreducible doubly stochastic matrix.

Proof: $\mathbf{P}_G$ is a stochastic matrix, i.e. the sum of the elements of each row of is 1. Since $\mathbf{P}_G$ is symmetric, $\mathbf{P}_G$ is doubly stochastic. Since Δ_G^C is irreducible, $\mathbf{P}_G$ is irreducible. □

Lemma 3. Let $\mathbf{C}$ be an irreducible, symmetric non-negative $n \times n$ matrix. If the diagonal of $\mathbf{C}$ is strictly positive, then

$$\mathbf{C}^{n-1} > 0$$

i.e. $\mathbf{C}$ is primitive.

Proof: It is sufficient to show that for every vector $\mathbf{y} \geq 0$ ($\mathbf{y} \neq 0$) the inequality $\mathbf{C}^{n-1}\mathbf{y} > 0$ holds. It remains to show that $\mathbf{z} = \mathbf{C}\mathbf{y}$ always has fewer zero co-ordinates than $\mathbf{y}$. Note that the non-zero co-ordinates of $\mathbf{y}$ are also non-zero co-ordinates of $\mathbf{z}$ since the diagonal is strictly positive. Without loss of generality, assume that $\mathbf{y}$ has the form

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ 0 \\ \vdots \end{pmatrix}$$

Since C is irreducible, $\exists j \leq i$ and $\exists k > i$ such that $c_{jk} = c_{kj} \neq 0$. It follows that $(\mathrm{Cy})_k = \sum_{l=1}^{n} c_{kl} y_l > 0$. The primitivity follows (Reference 13, theorem 8, page 422). □

Theorem 1. The Markov chain associated with the doubly stochastic matrix $\mathbf{P}_G$ is irreducible, regular and acyclic.

Proof: The irreducibility follows directly from lemma 2. Since $\mathbf{P}_G$ is primitive (lemma 3), $\mathbf{P}_G$ is acyclic. Since 0 is a simple eigenvalue of Δ_G^C, 1 is a simple eigenvalue of $\mathbf{P}_G$. -1 cannot be an eigenvalue because $\mathbf{P}_G$ is primitive; hence $\mathbf{P}_G$ has exactly one eigenvalue of maximal modulus 1. It follows that the Markov chain is regular. □

Theorem 2.

$$\lim_{t \to \infty} \mathbf{P}_G^t = \mathbf{P}_G^{\infty} \text{exists}$$

and

$$\mathbf{P}_G^{\infty} = \frac{1}{n}\mathbf{I}$$

Proof: The convergence follows directly from Gantmacher (Reference 13, theorem 11, page 436). It is well known that every column vector of $\mathbf{P}_G^{\infty}$ is a characteristic vector of the stochastic matrix P_G for the eigenvalue 1, i.e. αe. Since $\mathbf{P}_G^t$ is symmetric, all the columns of $\mathbf{P}_G^{\infty}$ are equal, i.e. $p_{ij}^{\infty} = \frac{1}{n} \quad \forall i, j$. □

Remark 2. The iteration of equation (1) converges for any valuation

$$c_{ij} = \frac{1}{\max(v_i, v_j) + \epsilon} \qquad \text{if } (i,j) \in E \text{ and } \epsilon > 0$$

because the diagonal of $\mathbf{P}_G$ still remains strictly positive.

3.4. Asymptotic behaviour

The following lemma is obvious:

Lemma 4. (Asymptotic Behaviour). Let $1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ be the eigenvalues of a Poisson operator P and let

$$\lambda_G = \max_{k \geq 2} \{ | \lambda_k | \}$$

If x is a vector with non negative co-ordinates such that

$$\sum_{i=1}^{n} x_i = 1$$

then for large k $\mathbf{P}^k\mathbf{x}$ satisfies the following asymptotic relation:

$$P^k\mathbf{x} \simeq \frac{1}{n}\mathbf{e} + \lambda_G^k e'$$

where $\mathbf{e}' \perp \mathbf{e}$.

Proof: Choose a base $\mathbf{B} = (e_1, e_2, \ldots, e_n)$ consisting in orthonormal eigenvectors of $\mathbf{P}_G$ associated with $\lambda_1 = 1, \lambda_2, \ldots, \lambda_n$ (this is possible since $\mathbf{P}_G$ is symmetric). Let α_i be the scalar product $< e_i, x >$, $i = 1, \ldots, n$. Then

$$\mathbf{x} = \sum_{i=1}^{n} \alpha_i e_i$$

Since $\mathbf{e}_1 = \frac{1}{\sqrt{n}}\mathbf{e}$ is the normalized eigenvector associated with the eigenvalue $\lambda_1 = 1$, $\alpha_1 = \frac{1}{\sqrt{n}}$. It follows that

$$\mathbf{x} = \frac{1}{n}\mathbf{e} + \sum_{i=2}^{n} \alpha_i e_i$$

and

$$\mathbf{P}_G^t\mathbf{x} = \frac{1}{n}\mathbf{e} + \sum_{i=2}^{n} \alpha_i \lambda_i^t e_i$$

The rest of the proof is straightforward. □

In the rest of the paper, the eigenvalue

$$\lambda_G = \max_{k \geq 2}\{|\lambda_k|\}$$

will be called the *convergence factor* of the Poisson operator associated with G.

Theorem 3. Let $\mathbf{P}_G$ be the n-dimensional Poisson operator in a graph **G=(N,E)** with eigenvalues $1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. If $n \geq 2$, the convergence factor

$$\lambda_G = \max_{k \geq 2}\{|\lambda_k|\}$$

satisfies the inequality

$$\lambda \leq 1 - \frac{4}{\nu + 1}\sin^2\frac{\pi}{2n}$$

where $\nu = \max \nu_i$

Proof: Denote by μ_G the measure of irreducibility of $\mathbf{P}_G$:

$$\mu_G = \min_{\emptyset \neq M \subset N} \sum_{i \in M, k \notin M} p_{ik}$$

Then, according to Reference 14, the second greatest eigenvalue of $\mathbf{P}_G$ satisfies

$$\lambda_2 \leq 1 - 4\mu_G \sin^2\left(\frac{\pi}{2n}\right)$$

Since $\mu_G \geq \frac{1}{\nu+1}$, this proves the lemma if all eigenvalues of $\mathbf{P}_G$ are non-negative. Suppose that $\lambda_n < 0$ and let $d = \min p_{ii}$. Since $d = 1 - \sum_{(i,j)\in E} p_{ij} = 1 - \sum_{(i,j)\in E} \frac{1}{\max(\nu_i,\nu_j)+1} \geq 1 - \sum_{(i,j)\in E} \frac{1}{\nu_i+1} = 1 - \frac{\nu_i}{\nu_i+1} \geq \frac{\nu}{\nu+1}$, it follows that $d \geq \frac{1}{\nu+1}$. It is well known that each eigenvalue λ satisfies $|\lambda| \leq 1 - 2d$. It follows that $|\lambda_n| \leq 1 - \frac{2}{\nu+1}$, and since $2\sin^2\frac{\pi}{2n} \leq 1$ for $n \geq 2$, the proof is complete. □

4. COMPLEXITY OF THE LOAD BALANCING ALGORITHM

The load balancing algorithm (see Figure 2) is equivalent to the Poisson equation in the hardware graph H. It will converge if the weight `weight[(i,j)]` of each link (i,j) is initialized to $\max(\nu_i, \nu_j) + 1$.

It follows from Theorem 3 that the convergence factor λ_H of the Poisson operator satisfies

$$\lambda_H \leq 1 - \frac{4}{\texttt{max.links} + 1} \sin^2 \frac{\pi}{2|T|}$$

where T is the set of processors. Thus after t iteration steps of the load balancing algorithm, `load(t)` will differ from the uniform distribution by a factor λ_H^t (see Lemma 4).

Let $\epsilon < 1$ be a suitable small positive constant. If t is chosen such that

$$\lambda_H^t \leq \epsilon$$

then `load(t)` will differ from the uniform distribution by a factor ϵ.

Theorem 4. The worst case time complexity of the load balancing algorithm is

$$O(|T|^2)$$

Proof: Note that for large $|T|$, i.e. for large networks,

$$\lambda_H \simeq 1 - \frac{4}{\texttt{max.links} + 1} \sin^2\left(\frac{\pi}{2|T|}\right) \simeq 1 - \frac{\pi^2}{|T|^2(\texttt{max.links} + 1)} \simeq 1 - \frac{1}{|T|^2}$$

Since

$$t = \frac{\ln \epsilon}{\ln \lambda_H}$$

it follows that

$$t = O\left(\frac{1}{\ln\left(1 - \frac{1}{|T|^2}\right)}\right)$$

Since

$$\ln\left(1 - \frac{1}{|T|^2}\right) \simeq \frac{1}{|T|^2}$$

the complexity of the load balancing algorithm is $O(|T|^2)$. □

5. EXAMPLES

In this section we apply the results of Section 3 to important parallel architectures such as linear processor array, hypercube, circuit, torus, etc. As may be expected, the diffusion is much faster in the case of hypercubes or a torus than for linear arrays or circuits.

5.1. Poisson equation in a path

Let $\mathbf{P}_n = (\mathbf{N},\mathbf{E})$ be a path with n vertices, i.e.

$$\mathbf{E} = \{(i, i+1) \mid i \in \mathbf{N} - \{\mathbf{n}\}\}$$

The associated Laplacean has the form

$$\Delta^{\mathbf{C}}_{\mathbf{P}_n} = \frac{1}{3}\begin{pmatrix} 1 & -1 & & & & & 0 \\ -1 & 2 & -1 & & & & \\ & -1 & . & . & & & \\ & & . & . & . & & \\ & & & . & . & -1 & \\ & & & & -1 & 2 & -1 \\ 0 & & & & & -1 & 1 \end{pmatrix}$$

Let

$$\mathbf{P}_{\mathbf{P}_n} = \frac{1}{3}\begin{pmatrix} 2 & 1 & & & & & 0 \\ 1 & 1 & 1 & & & & \\ & 1 & . & . & & & \\ & & . & . & . & & \\ & & & . & . & 1 & \\ & & & & 1 & 1 & 1 \\ 0 & & & & & 1 & 2 \end{pmatrix}$$

be the associated Poisson operator.

Remark 3. The matrix associated with the Poisson operator corresponds to an iteration scheme for a numerical solution of the heat equation in a bar.

Lemma 5[15]: The eigenvalues of $\mathbf{P}_n$ are

$$\lambda_k = \frac{1 + 2\cos\frac{(k-1)\pi}{n}}{3}, \qquad k = 1, 2, \ldots, n$$

and the characteristic vector $\mathbf{v}^{(k)}$ associated to λ_k has the following co-ordinates:

$$v_i^{(k)} = \cos(2i - 1)\frac{(k-1)\pi}{2n}$$

Proof: It can be shown that

$$\cos(2i-3)\frac{(k-1)\pi}{2n} + \cos(2i-1)\frac{(k-1)\pi}{2n} + \cos(2i+1)\frac{(k-1)\pi}{2n} =$$
$$\cos(2i+1)\frac{(k-1)\pi}{2n}\left(1 + 2\cos\frac{(k-1)\pi}{2n}\right)$$

$i = 2, 3, \ldots, n-1$, and

$$2\cos\frac{(k-1)\pi}{2n} + cos\frac{3(k-1)\pi}{2n} =$$
$$\cos\frac{(k-1)\pi}{2n}\left(2 + \cos\frac{(k-1)\pi}{n}\right) - \sin\frac{(k-1)\pi}{n}\sin\frac{(k-1)\pi}{2n} =$$
$$\cos\frac{(k-1)\pi}{2n}\left(2 + \cos\frac{(k-1)\pi}{n}\right) - 2\sin\frac{(k-1)\pi}{2n}\cos\frac{(k-1)\pi}{2n}\sin\frac{(k-1)\pi}{2n} =$$
$$\cos\frac{(k-1)\pi}{2n}\left(2 + \cos\frac{(k-1)\pi}{n} - 2\sin^2\frac{(k-1)\pi}{2n}\right) =$$
$$\cos\frac{(k-1)\pi}{2n}\left(1 + 2\cos\frac{(k-1)\pi}{n}\right)$$

etc. □

Lemma 6. The convergence factor $\lambda_{\mathbf{P}_n}$ of the Poisson operator in the path $\mathbf{P}_n$ with n vertices is

$$\lambda_{\mathbf{P}_n} = \frac{1 + 2\cos\frac{\pi}{n}}{3}$$

i.e.

$$\lambda_{\mathbf{P}_n} \simeq 1 - \frac{\pi^2}{3n^2}$$

□

5.2. Poisson equation in a hypercube

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The sum $G_1 + G_2$ of these two graphs is the graph $G = (V, E)$, where $V = V_1 \times V_2$ and E is defined as follows. Let $(x_1, x_2) \in V$

and $(y_1, y_2) \in V$. The vertices (x_1, x_2) and (y_1, y_2) are adjacent in the sum $G_1 + G_2$ if and only if either $x_1 = y_1$ and $(x_2, y_2) \in E_2$, or $(x_1, y_1) \in E_1$ and $x_2 = y_2$.

Let H_d be the d-dimensional hypercube. According to Cvetkovič[16, page 75] H_d can be represented as the sum $G_1 + G_2 + \ldots + G_d$ of d paths, P_2 with two vertices, i.e. $G_i \simeq P_2, i = 1, \ldots, d$. Let $\mathbf{A}_{H_d}$ be the adjacency matrix of H_d. It is well known that the spectrum of $\mathbf{A}_{H_d}$ consists of all numbers $\lambda_{1i_1} + \ldots + \lambda_{di_d}$, where λ_{ji_j} is an eigenvalue of the adjacency matrix of $\mathbf{P}_2$, i.e. $\lambda_{ji_j} \in \{1, -1\}$. It follows that the eigenvalues of $\mathbf{A}_{H_d}$ are

$$\alpha_j = d - 2j, \qquad j = 0, \ldots, d$$

with multiplicities

$$p_j = \binom{d}{j}, \qquad j = 0, \ldots, d$$

Theorem 5. The eigenvalues of the d-dimensional hypercube H_d are

$$\lambda_i = 1 - \frac{2j}{d+1}, \qquad i = 0, \ldots, d$$

with multiplicities

$$p_i = \binom{d}{i}, \qquad i = 0, \ldots, d$$

Proof: The d-dimensional hypercube is a regular graph of degree d, i.e. all vertices of the hypercube have the same valency d. The Poisson operator can be directly computed from the adjacency matrix of H_d, and it is easy to see that

$$\Delta^C_{H_d} = \frac{1}{d+1}(d\mathbf{I} - \mathbf{A}_{H_d})$$

Since $\mathbf{P}_{H_d} = \mathbf{I} - \Delta^C_{H_d}$, it follows that

$$\mathbf{P}_{H_d} = \frac{1}{d+1}(\mathbf{I} + \mathbf{A}_{H_d})$$

with eigenvalues

$$\lambda_i = \frac{1}{d+1}(1 + \alpha_i), \qquad i = 0, \ldots, d$$

where α_i is an eigenvalue of $\mathbf{A}_{H_d}$, i.e.

$$\lambda_i = 1 - \frac{2i}{d+1}, \qquad i = 0, \ldots, d$$

□

Lemma 7. The convergence factor λ_{H_d} of the Poisson operator in the d-dimensional hypercube H_d is

$$\lambda_{H_d} = 1 - \frac{2}{d+1}$$

Proof:

$$\max_{j=1,\dots,d} |\lambda_j| = \max_{j=1,\dots,d} \left|1 - \frac{2j}{d+1}\right| = 1 - \frac{2}{d+1}$$

□

Note that for large dimensions d and for $k = td$

$$\left(1 - \frac{2}{d+1}\right)^k \simeq e^{-2t}$$

5.3. Poisson equation in a circuit

According to Reference 14 the eigenvalues of $\mathbf{P}_{C_n}$ are

$$\frac{1 + 2\cos\frac{2(k-1)\pi}{n}}{3}, \qquad k = 1, 2, \dots, n$$

and correspond to the orthogonal eigenvectors

$$v_i^{(k)} = \cos(2i - 1)\frac{(k-1)\pi}{2n}$$

Lemma 8. The convergence factor λ_{C_n} of the Poisson operator in the circuit C_n with n vertices is

$$\lambda_{C_n} = \frac{1 + 2\cos\frac{2\pi}{n}}{3}$$

i.e.

$$\lambda_{C_n} \simeq 1 - \frac{4\pi^2}{3n^2}$$

□

5.4. Poisson equation in a torus

If $C_{n_1 n_2}$ is an $n_1 \times n_2$ torus, then the eigenvalues of $\Delta^C_{C_{n_1 n_2}}$ are

$$\frac{1 + 2\cos\frac{2(k_1-1)\pi}{n_1} + 2\cos\frac{2(k_2-1)\pi}{n_2}}{5} \qquad k_1 = 1, 2, \dots, n_1 \; k_2 = 1, 2, \dots, n_2$$

Lemma 9. The convergence factor $\lambda_{C_{n_1 n_2}}$ of the $n_1 \times n_2$ torus is

$$\lambda_{C_{n_1 n_2}} = \frac{3 + 2\cos\frac{2\pi}{n}}{5}$$

where $n = \max\{n_1, n_2\}$, i.e.

$$\lambda_{C_{n_1 n_2}} \simeq 1 - \frac{4\pi^2}{5n^2}$$

□

Thus the Poisson equation converges faster in an $n_1 \times n_2$ torus than in a path with $n_1 \times n_2$ vertices.

The result above may be generalized for an $n_1 \times n_2 \times \ldots \times n_m$ torus since

$$\mathbf{C}_{n_1 n_2 \ldots n_m} = \mathbf{C}_{n_1} + \mathbf{C}_{n_2} + \ldots + \mathbf{C}_{n_m}$$

5.5. Poisson equation in the complete graph

Let $F_n = (N, E)$ be the complete graph with n vertices, i.e.

$$E = \{(i,j) \mid i \neq j\}$$

Lemma 10. The eigenvalues of $\mathbf{P}_{F_n}$ are

$$\lambda_k = \begin{cases} 1 & \text{if } k = 1 \\ 0 & \text{else} \end{cases}$$

□

Lemma 11. The convergence factor λ_{F_n} of the Poisson operator in the complete graph F_n with n vertices is

$$\lambda_{F_n} = 0$$

i.e. the uniform load distribution is reached after only one iteration step. □

5.6. Poisson equation in the De Bruijn graph $UB(d,D)$

The nodes of the undirected De Bruijn graph $UB(d,D)$ are words of length D on an alphabet of d letters. The node $(x_1, x_2, \ldots, x_D)$ is adjacent to all the nodes $(x_2, \ldots, x_D, \alpha)$ and $(\alpha, x_1, \ldots, x_{D-1})$, where α is any letter.

The nodes have degree $2d - 2$, $2d - 1$ or $2d$ if $D \geq 3$ and the graph has diameter D. De Bruijn graphs can connect more nodes for comparable diameter and degree than the hypercube[17].

The following examples have been computed with the mathematical tool *Mathematica*[18].

5.6.1. Poisson equation in $UB(2,3)$

The Poisson operator in the graph $UB(2,3)$ (see Figure 3) has the form

$$\mathbf{P}_{UB(2,3)} = \frac{1}{60} \begin{pmatrix} 36 & 12 & 0 & 0 & 12 & 0 & 0 & 0 \\ 12 & 12 & 12 & 12 & 12 & 0 & 0 & 0 \\ 0 & 12 & 21 & 0 & 12 & 15 & 0 & 0 \\ 0 & 12 & 0 & 12 & 0 & 12 & 12 & 12 \\ 12 & 12 & 12 & 0 & 12 & 0 & 12 & 0 \\ 0 & 0 & 15 & 12 & 0 & 21 & 12 & 0 \\ 0 & 0 & 0 & 12 & 12 & 12 & 12 & 12 \\ 0 & 0 & 0 & 12 & 0 & 0 & 12 & 36 \end{pmatrix}$$

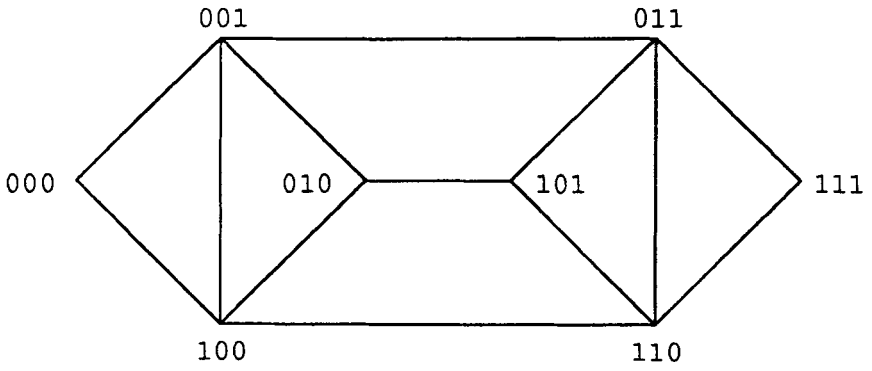

Figure 3. Undirected De Bruijn graph $UB(2,3)$

The nodes are labelled with binary numbers in the range $0, \ldots, 111$. The numerical eigenvalues of the Poisson operator are

$$(1.0, 0.775, 0.6, 0.309, 0.2, -0.184, -0.2)$$

with respective multiplicities

$$(1, 1, 1, 1, 2, 1, 1)$$

This results a convergence factor

$$\lambda_{UB(2,3)} = 0.775$$

5.6.2. Poisson equation in $UB(2,4)$

$UB(2,4)$ has the same diameter and the same number of nodes as the 4-dimensional hypercube. The numerical eigenvalues of the operator $\mathbf{P}_{UB(2,4)}$ are

$$(1.0, 0.847, 0.766, 0.6, 0.465, 0.304, 0.2, -0.047, -0.11, -0.2, -0.359, -0.366)$$

with respective multiplicities

$$(1, 1, 1, 2, 1, 1, 4, 1, 1, 1, 1, 1)$$

This results in a convergence factor

$$\lambda_{UB(2,4)} = 0.847$$

5.7. Poisson equation in the Kautz graph $UK(d,D)$

The nodes of the undirected Kautz graph $UK(d,D)$ are words of length D on an alphabet of d letters where two consecutive letters are different. The node $(x_1, x_2, \ldots, x_D)$ is adjacent to all the nodes $(x_2, \ldots, x_D, \alpha)$ and $(\beta, x_1, \ldots, x_{D-1})$, where α is any letter except x_D and β is any letter except x_1.

The nodes have degree $2d - 1$ or $2d$ if $D \geq 3$ and the graph has diameter D. Like

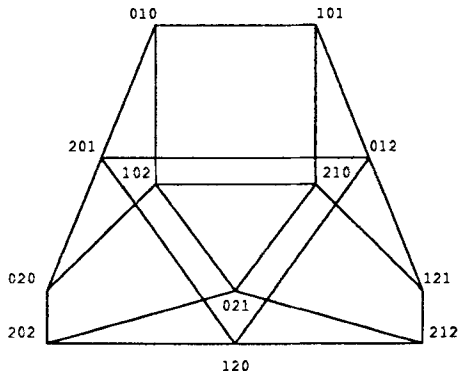

Figure 4. Undirected Kautz graph $UK(2,3)$

De Bruijn graphs, Kautz graphs can connect more nodes for comparable diameter and degree than the hypercube[17].

The following examples have been computed using *Mathematica*[18].

5.7.1. Poisson equation in $UK(2,3)$

The Poisson operator in the graph $UK(2,3)$ (see Figure 4) has the form

$$\mathbf{P}_{UK(2,3)} = \frac{1}{20}\begin{pmatrix} 7 & 0 & 0 & 0 & 5 & 4 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 4 & 0 & 4 & 4 & 4 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 4 & 0 & 0 & 4 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 4 & 0 & 0 & 0 & 4 & 4 & 4 \\ 5 & 4 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 4 & 0 & 4 & 4 & 0 & 4 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 4 & 0 & 4 & 4 & 0 & 4 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 4 & 5 \\ 4 & 4 & 4 & 0 & 0 & 0 & 4 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 4 & 0 & 0 & 4 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 & 4 & 0 & 4 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 4 & 5 & 0 & 0 & 0 & 7 \end{pmatrix}$$

The nodes are labelled in lexicographic order.

The numerical eigenvalues of the Poisson operator are

$$(1.0, 0.684, 0.6, 0.342, 0.2, 0.1, 0.0, -0.325, -0.325)$$

with respective multiplicities

$$(1, 2, 1, 2, 1, 1, 2, 1, 1)$$

This results in a convergence factor

$$\lambda_{UK(2,3)} = 0.689$$

5.7.2. Poisson equation in UK(2,4)

The numerical eigenvalues of the operator $\mathbf{P}_{UK(2,3)}$ are

$$(1.0, 0.8, 0.775, 0.6, 0.349, 0.309, 0.286, 0.2, -0.09, -0.184, -0.2, -0.445)$$

with respective multiplicities

$$(1, 2, 1, 3, 2, 1, 2, 4, 2, 1, 3, 2)$$

This results a in convergence factor

$$\lambda_{UK(2,4)} = 0.8$$

5.8. Discussion of the examples

Table 1 contains the convergence factor of some graphs for which it was possible to compute the eigenvalues exactly. Table 2 contains a summary of the asymptotic convergence factors for the graphs described above.

An architecture based on a complete graph is unrealistic, and thus its associated (fastest) convergence factor is of no interest. However, the convergence factor of the path $\mathbf{P}_n$ is particularly interesting. It represents the worst case convergence behaviour of the load balancer. The inequality of Theorem 3,

$$\lambda \leq 1 - \frac{4}{v+1} \sin^2\left(\frac{\pi}{2n}\right) = \frac{v - 1 + 2\cos\frac{\pi}{n}}{v+1}$$

Table 1. Convergence factor of the Poisson operator

Graph	Vertices	Diameter	Degree	Convergence factor
$\mathbf{F}_n$	n	1	$n-1$	0
$\mathbf{H}_d$	2^d	d	d	$1 - \frac{2}{d+1}$
$\mathbf{P}_n$	n	$n-1$	2	$\frac{1+2\cos\frac{\pi}{n}}{3}$
$\mathbf{C}_n$	n	$\lfloor\frac{n}{2}\rfloor$	2	$\frac{1+2\cos\frac{2\pi}{n}}{3}$
$\mathbf{C}_{n_1 n_2}$	$n_1 \times n_2$	$\lfloor\frac{n_1}{2}\rfloor + \lfloor\frac{n_2}{2}\rfloor$	4	$\frac{3+2\cos\frac{2\pi}{\max\{n_1,n_2\}}}{5}$
$\mathbf{C}_{n_1 \ldots n_m}$	$n_1 \times \ldots \times n_m$	$\lfloor\frac{n_1}{2}\rfloor + \ldots + \lfloor\frac{n_m}{2}\rfloor$	$2m$	$\frac{2m-1+2\cos\frac{2\pi}{\max\{n_1,\ldots,n_m\}}}{2m+1}$

Table 2. Asymptotic convergence factor of the Poisson operator

Graph	Asymptotic factor
$\mathbf{F}_n$	0
$\mathbf{H}_d$	$1 - \frac{2}{d+1}$
$\mathbf{P}_n$	$1 - \frac{\pi^2}{3n^2}$
$\mathbf{C}_n$	$1 - \frac{4\pi^2}{3n^2}$
$\mathbf{C}_{n_1 n_2}$	$1 - \frac{4\pi^2}{5(\max\{n_1, n_2\})^2}$
$\mathbf{C}_{n_1 \dots n_m}$	$1 - \frac{4\pi^2}{(2m+1)(\max\{n_1, \dots, n_m\})^2}$

is the best possible for paths, since $\nu = 2$. The remaining results indicate that the hypercube and hypertorus architectures are well suited for the load balancer.

An analysis of the above examples shows that the convergence factor could be a function of the diameter and the vertex degree of the graph. Note that for a specific non-reconfigurable architecture, the convergence factor can be computed directly from the hardware graph. For reconfigurable architectures like the Supernode[2], the Poisson operator may change during the execution of a program resulting in a new convergence factor. The inequality of Theorem 3 has the advantage of being fixed for a given reconfigurable architecture provided all communication links are used.

The results show that the Kautz graph may be an interesting competitor to the hypercube or the hypertorus. This architecture is of particular interest because some processors have a smaller vertex degree than the maximal one. This enables such a system to be connected to the environment (see also Reference 17). The performance of the load balancer in Kautz graphs seems to be better than in De Bruijn graphs. Both Kautz and De Bruijn graphs seem to be more flexible for mapping a process system with inhomogeneous communication structure than the hypercube.

Remark 4. During the revision of the paper, Cybenko[19] published a similiar paper on diffusion based load balancing. Cybenko proposes normalized linear transformations $\alpha \mathbf{P}_G + \beta \mathbf{I}$, $\alpha + \beta = 1$, for optimizing the convergence factor. In particular, he proves that the Poisson diffusion scheme is the best possible for the hypercube. Note that this method can only be applied if the graph G is known. Unfortunately, Cybenko does not give any bounds for the convergence factor.

Remark 5. In the discrete case, i.e. working with individual processes, our problem is equivalent to the random walk problem in graphs[20]. At the moment, only few results in this area have been published.

Table 3. Convergence factor of the Poisson operator for some graphs with maximal vertex degrees 3 and 4

Graph	Vertices	Diameter	Degree	Convergence factor
H_3	8	3	3	0.500
$C_{2\times4}$	8	3	4	0.600
$UB(2,3)$	8	3	4	0.775
C_8	8	4	2	0.804
P_8	8	7	2	0.949
$C_{3\times4}$	12	3	4	0.600
$UK(2,3)$	12	3	4	0.684
C_{12}	12	6	2	0.910
P_{12}	12	11	2	0.977
H_4	16	4	4	0.600
$C_{4\times4}$	16	4	4	0.600
$UB(2,4)$	16	4	4	0.847
C_{16}	16	8	2	0.949
P_{16}	16	15	2	0.987
$UK(2,4)$	24	4	4	0.800
$C_{4\times6}$	24	5	4	0.800

6. IMPLEMENTATIONS

In this Section we present briefly the results of two implementations of the load balancing algorithm. The first application concerns the mapping problem and the second a two-dimensional molecular dynamics simulation.

6.1. Mapping

The mapping problem[21,22] is fundamental in distributed computing. The problem occurs every time that a system of communicating processes has to be placed onto a parallel architecture. The mapping may be defined topologically as follows:

> Is there a mapping of a system of communicating processes onto a processor network such that the neighbouring processes are assigned to neighbouring processors?

Since mapping communicating processes onto processors networks is known to be NP-complete[2], it makes no sense to try and develop exact algorithms for solving the problem. Furthermore, mapping algorithms should be able to be integrated into distributed operating systems, and be able to produce suboptimal solutions in a short time. Our load balancing algorithm is well-suited for solving the mapping problem[3], and delivering good suboptimal solutions in a short time. The algorithm is fully distributed and has an SPMD structure. It has been implemented for mapping a static system of communicating processes onto a processor network. The mapper is implemented in **occam**[23,24] and can be run on any transputer network[25]. It is integrated in a tool called MARC for automatic configuration of **occam** programs[26–28]. The mapper is also used in a simulation program for high order petri nets[29].

The mapper is based on the load balancer (see Figure 2) and has the same structure. It works with partial communication information, i.e. each node of the mapper has a list of the probable locations of all the processes to be mapped.

```
SEQ i = 0 FOR iterations
  SEQ
    . . . exchange informations and processes with direct neighbours
    . . . choose process with high local communication costs
    . . . decide whether to send it to neighbouring processor
    . . . balance load locally
```

Figure 5. General structure of the mapping algorithm

At each information exchange step, the current knowledge about load and probable location of all processes are exchanged, and the local list of probable process locations can be updated. In the next step, a process causing high local communication costs, i.e. a process with very distant neighbours, or a process which has no local neighbours, is selected for communication cost reduction processing. Since the probable location of all processes is known, the process to be moved can be sent closer to the correct processor. This process will move towards its distant neighbour(s), resulting in a reduction of the communication costs. This step is executed randomly. The probability that such a process is chosen decreases. This corresponds somewhat to the improvement steps of a simulated annealing algorithm. The last step of the algorithm consists of the local load diffusion. If the mapper has to distribute some load to a given neighbouring processor, it will first try to exchange processes which have neighbours either placed on that processor or reachable through that processor (shortest path).

Tests have shown that uniform load is reached in more than 98% of cases (where possible). Table 4 shows the results of the mapping of a 5-dimensional hypercube of homogeneous communicating processes onto a 3 dimensional processor hypercube. In Table 4 all processes have the same load and all communication has the same costs. Hardware communication costs are set to 1 for processes mapped onto the same processor or onto neighbouring processors, and set to 2 × distance for the remaining communications. The cost of multiplexing communication channels are not taken in account. The algorithm was executed on a transputer cube (20 MHz); for each number of iterations 50 runs have been made.

The load equilibrium was reached in all but two cases. Tests have shown that a load equilibrium is reached after about 20 iteration steps of the mapper. The other steps are used for optimizing the network communication costs.

The results show that after 800 iteration steps, a suboptimal placement is reached, whose mean differs by only 3% from the optimal solution. 94% of the mappings are optimal. Tests done with the Petri net simulation program[29] show that the mapper is also well suited for mapping process systems with different loads onto multiprocessor systems.

Table 4. Mapping a 5-dimensional process hypercube onto a 3-dimensional processor hypercube

Number of iterations	Optimum costs	Mean costs	Variance	Success	Time used
200	80	99.22	22.82	54%	1.18 s
400	80	91.50	17.99	66%	2.31 s
600	80	85.18	14.31	88%	3.46 s
800	80	83.02	12.19	94%	4.62 s

6.2. Molecular dynamics

The load balancing algorithm has been implemented as part of a SPMD Molecular Dynamics (MD) application for simulating 4050 particles interacting by the Lennard–Jones 12–6 potential (short-range interactions)[7]. At initialization, the simulation box is decomposed in equal-area (i.e. well balanced) rectangular domains (see Figure 6). Every domain is assigned to one of the processors in a linear array.

In a distributed implementation of am MD application, information concerning boundary particles must be exchanged between neighbouring processors. After this step, the moves of the particles are computed and boundary particles leaving the region are sent to a neighbouring processor, resulting in a change of the load of the processors. Without load balancing, the execution speed of a distributed MD application depends upon the processor with the greatest load (greatest number of particles in the underlying region). This results in idle time for other processors in the system (see Figure 6).

The load balancing algorithm is particularly well suited for MD applications. Each domain can be divided up into different regions, each being assigned to a specific processor. The population of each processor is then maintained constant during the whole simulation. Geometrically, the exchange of particles between the processors can be interpreted in moving the strip limits of the domains (see Figure 6). With this method, the ratio of communication costs to computation costs is smaller than that of the scattered decomposition method proposed in Reference 4, page 322.

Note that load information exchange, boundary moves (particles exchange) and co-ordinates exchange of boundary particles may be achieved in the same communication step, resulting in a minimal communication overhead for the load balancer (Figure 7). Experiments have shown that the population of each processor remains very near to the ideal ratio of population over processors during the whole simulation (see Figure 8).

Embedding the MD application into the load balancer has resulted in the execution time per MD step being reduced considerably (see Figure 9). The implementation of the load balancer has shown that the convergence is much faster than the theoretical worst case. This can be explained by the fact that the load balancer works with discrete load units (in the theoretical case the load is a real number).

7. CONCLUSIONS

In conclusion, the proposed load balancer offers a good alternative to the traditional load balancing techniques. Indeed, this load balancer:

- is very efficient
- is very simple and easy to implement
- is distributed over the whole processor network and does not need any global synchronization. The synchronizations are restricted to local interactions between neighbouring processors.

We intend to apply our tool to reconfigurable architectures. The load balancer seems to be particularly well suited for such networks because the load balancer can adapt itself to every new processor topology resulting from the switching of interprocessor links.

Furthermore, together with a communication kernel, the load balancer is a good candidate for building in a truly distributed operation system.

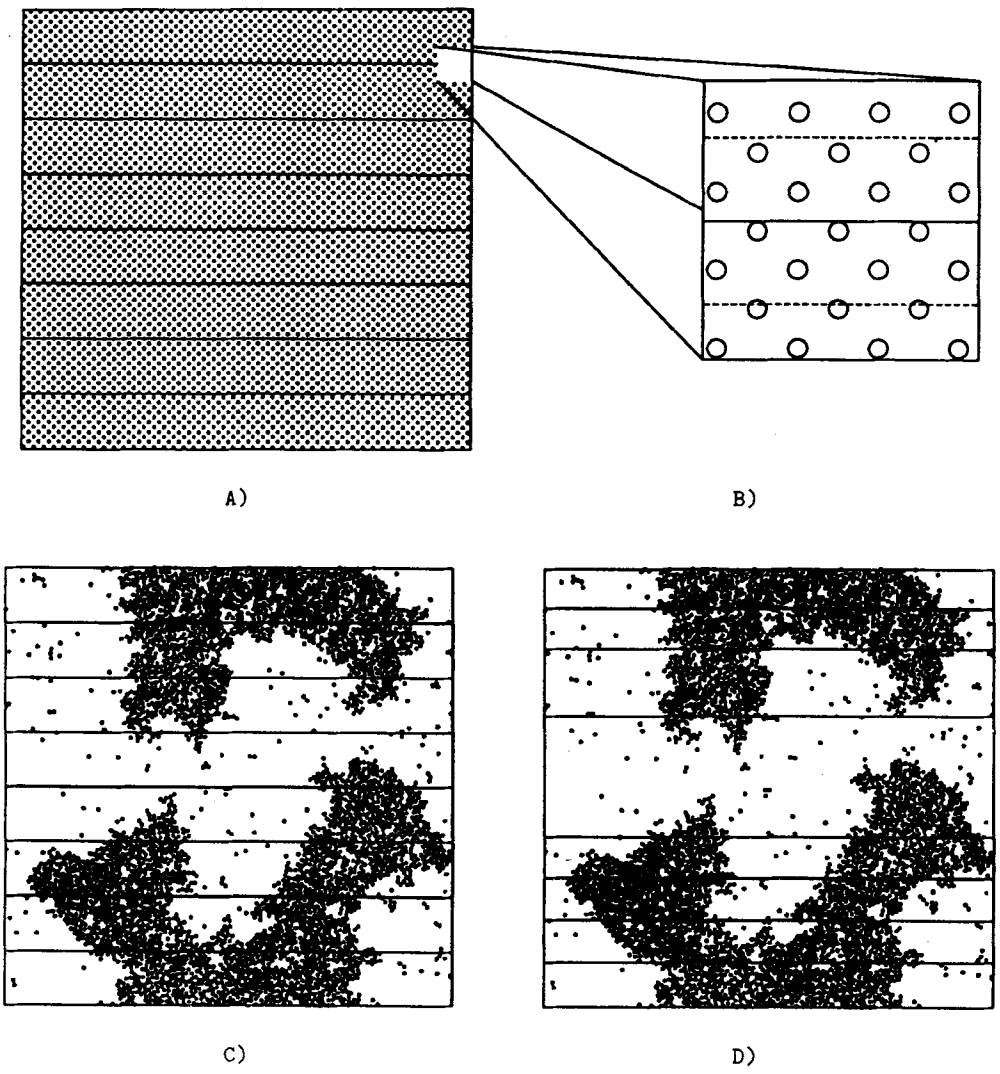

Figure 6. (A) Typical initial configuration for the molecular dynamics simulation of a two-dimensional system involving 4050 Lennard–Jones particles, which undergoes a spinodal phase separation. The simulation box is subdivided into eight rectangular domains which are attributed to the eight ring-connected Transputers constituting a MIMD computer; (B) exploded view of the boundary region between two adjacent domains; (C) final configuration reached after 100,000 integration steps (equivalent to a simulated time of 1 ns) keeping constant the domain limits; (D) the same final configuration as in (C) obtained while the load balancer is active, which causes a different domain sizing in order to optimize the processor work-load.

```
WHILE cycling
  SEQ
    . . . calculate particle interactions
    . . . update particle positions
    . . . exchange boundary and incoming - outgoing
          particles with neighbouring processors
    . . . adjust co-ordinate matrix to maintain the data structure
    . . . compute new boundary limits
```

Figure 7. MD simulation with integrated load balancer

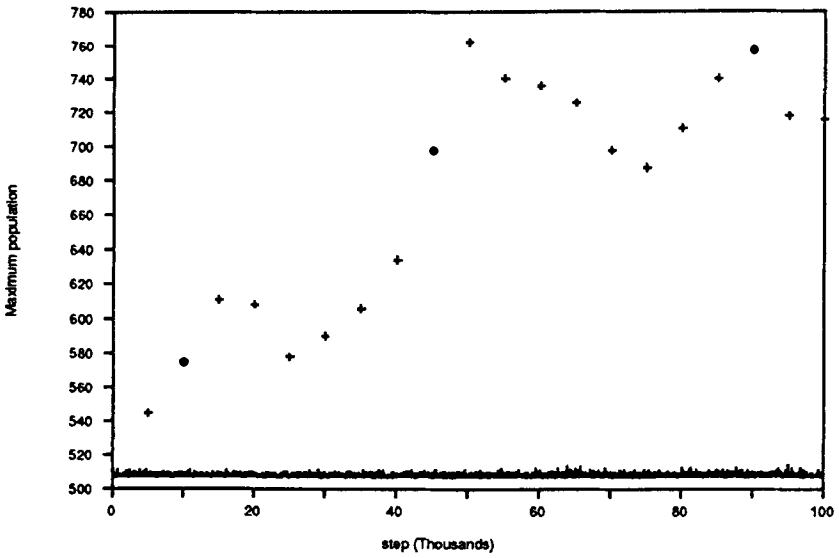

Figure 8. Maximum population over processors against MD step number with (continuous line) and without (crosses) load balancer

REFERENCES

1. R.W. Hockney and C.R. Jesshope. *Parallel Computers 2*, Adam Hilger, Bristol, 2 ed., 1988.
2. J.G. Harp, C.R. Jesshope, T. Muntean and C. Whitby-Strevens. 'Phase 1 of the development and application of a low cost high performance multiprocessor machine'. In *ESPRIT 86: Results and Achievments*, Directorate General XIII, ed., Elsevier, Amsterdam, 1987.
3. J.E. Boillat and P.G. Kropf. 'A fast distributed mapping algorithm', in CONPAR 90—VAPP IV Proceedings, Springer LNCS 457, 1990.
4. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors I*, Prentice–Hall, Englewood Cliffs, 1988.
5. G. Fox, A. Kolawa and R. Williams. 'The implementation of a dynamic load balancer'. In *Hypercube Multiprocessors*, M.T. Heath (ed.), SIAM, 1987.
6. D.E. Eager, E.D. Lazowska and J. Zahorjan. 'Adaptive load sharing in homogenous distributed systems', *IEEE Trans. Softw. Eng.*, **SE–12**(5), 662–675 (1986).
7. J.E. Boillat, F. Brugé and P.G. Kropf. 'A dynamic load balancing algorithm for molecular dynamics simulation on multi–processor systems' (Submitted for publication).
8. M.J. Flynn. 'Very high-speed computing systems'. *Proc. IEEE*, **54**(12), (1966).
9. M.J. Flynn. 'Some computer organizations and their effectiveness', *IEEE Trans. Comput.*, **C–21**(9), (1972).

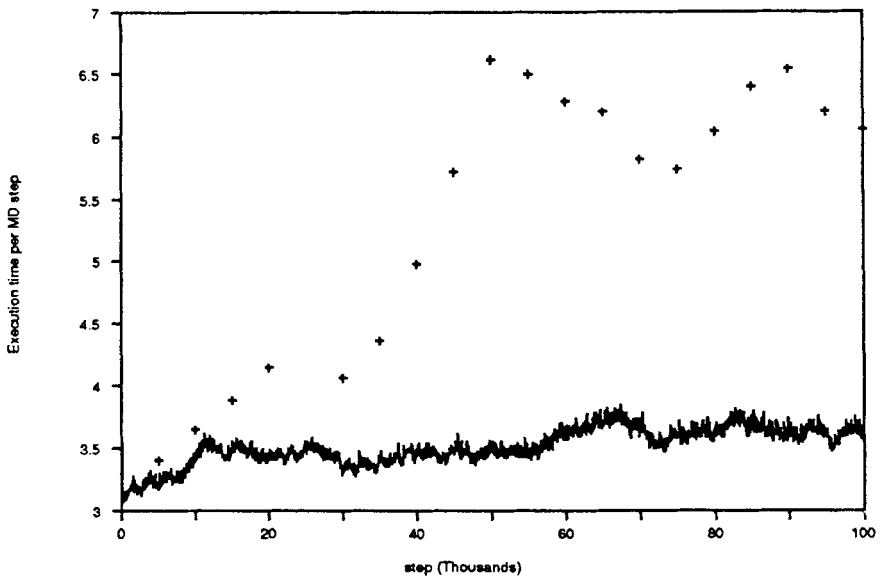

Figure 9. Execution time per MD step against step number with (continuous line) and without (crosses) load balancer

10. S. Wolfram. 'Statistical mechanics and cellular automata', *Rev. Modern Physics*, **55**, 601–644 (1983).
11. K. Sutner. 'Linear cellular automata and the garden-of-eden'. *The Mathematical Intelligencer*, **11**(2), 49–53 (1989).
12. M. Fiedler. 'A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory'. *Czes. Math. J.*, **25**(100), 619–633 (1975).
13. F.R. Gantmacher. *Matrizentheorie*, Springer, Berlin, 1986.
14. M. Fiedler. 'Bounds for eigenvalues of doubly stochastic matrices', *Linear Algebra Appl.*, **5**, 299–310 (1972).
15. H. Carnal. 1989. Private communication.
16. D.M. Cvetkovič, M. Doob and H. Sachs. *Spectra of Graphs*, Academic Press, New York, 1979.
17. F. Baude, F. Carré, P. Cléré and G. Vidal-Naquet. 'Topologies for large transputer networks: theoretical aspects and experimental results'. In *Applying Transputer Based Parallel Machines*, A. Bakkers (ed.), I.O.S., Amsterdam, 1989.
18. S. Wolfram. *Mathematica*, Addison–Wesley, Redwood City, 1988.
19. G. Cybenko. 'Dynamic load balancing for distributed memory multiprocessors', *J. Parallel Distrib. Comput.*, **7**, 279–301 (1989).
20. D. Aldous. 'An introduction to covering problems for random walks on graphs'. *J. Theoretical Probability*, **2**(1), 87–89 (1989).
21. S.H. Bokhari. 'On the mapping problem', *IEEE Trans. Comput.*, **C-30**(3), 550–557 (1981).
22. S.H. Bokhari. *Assignment Problems in Parallel and Distributed Computing*, Kluwer, Boston, 1988.
23. D. May. Occam. *ACM SIGPLAN Notices*, **18**(4), (1983).
24. INMOS. *OCCAM 2 Reference Manual*, Prentice–Hall, Englewood Cliffs, 1988.
25. INMOS. *Transputer Reference Manual*, Prentice–Hall, Englewood Cliffs, 1988.
26. J.E. Boillat, P.G. Kropf, D.Chr. Meier and A. Wespi. 'An analysis and reconfiguration tool for mapping parallel programs onto transputer networks'. In *OPPT*, T. Muntean (ed.), Grenoble, 1987.

27. J.E. Boillat, P.G. Kropf, D.Ch. Meier and A. Wespi. *Automatische Multiprozessorkonfiguration.* Technical Report IAM-PR-8687, University of Bern, Informatics, 1987.
28. J.E. Boillat, P.G. Kropf and S. Feller. *Automatische Multiprozessorkonfiguration.* Technical Report IAM-PR-8788, University of Bern, Informatics, 1987.
29. B. Bütler, R. Esser and R. Mattmann. 'A distributed simulator for high order petri nets'. In *Tenth International Conference on Application and Theory of Petri Nets*, W. Reisig and K. Voss (eds.), GMD, Bonn, 1989.