Mapping Strategies in Message Based Multiprocessor Systems

O. Krämer, H. Mühlenbein
Gesellschaft für Mathematik und Datenverarbeitung mbH
Postfach 1240, D-5205 St. Augustin

Abstract

Machines with distributed memory have the mapping problem — assigning processes to processors. In this paper we define the mapping problem as an optimization problem and discuss the question, how far is an optimum solution from an average or random solution.

The term robustness is introduced and explained in detail with two examples, the SUPRENUM and the Hypercube architecture. For the SUPRENUM architecture we show that a simple mapping strategy (optimal clustering of the processes) gives almost as good results as the optimal mapping. Optimal mapping is more important for the Hypercube architecture.

Mapping strategies are difficult to apply for inhomogeneous networks. For this networks adaptive routing seems promising.

1 Introduction

In the research for a new generation of computers a worldwide trend to parallel systems can be observed. The new machines make use of massive parallelism by the connection of some hundreds (or even thousands) of single processors to a "supercomputer".

The new machines can be divided into two classes, depending on their concept of memory access (see [Hock85]):

- multiprocessor systems with shared memory,
- multiprocessor systems with distributed memory.

Each of the classes has a very crucial problem. Without its solution the performance of the multiprocessor systems will be limited by a bottleneck. Shared memory systems have the **data synchronisation** problem, distributed memory systems have the **data transportation** problem.

Shared memory systems consist of a lot of processors, all working on the same memory. The central problem is the synchronisation when different processes are working on the same data. If too many processes work on the same data there is a potential bottleneck limiting the system performance (hot spot contention).

This problem does not exist in machines with distributed memory because each process can access only its local memory. Synchronisation and data transport is done by a message passing system. Now the central problem is a fast communication system and strategies to keep the amount of messages small.

In this paper we discuss how mapping strategies (assigning processes to processors) influence the amount of communication.

In chapter 2 we introduce a process model and the local memory abstract machine. In chapter 3 a mathematical model for the mapping problem is defined. Chapter 4 presents the three main approaches to solve the mapping problem: the topological mapping, a mathematical programming problem and an adaptive mapping strategy. In chapter 5 the term robustness is introduced

as a characterization of the system behavior when using different mapping strategies. The sensitivity of the SUPRENUM architecture against various mapping strategies is discussed in chapter 6. We analyse the worst-/best-case behaviour of important process systems and work out the main criteria for good heuristics. Chapter 7 contains the sensitivity analysis for the Hypercube architecture. The rationale for adaptive routing is discussed in chapter 8.

2 A parallel programming paradigm of abstract machines

Our analysis is based on a fairly general programming paradigm which consists of a network of abstract machines (for detail see [MKLMS87]). The central abstract machine is called LAM – the Local memory Abstract Machine. LAM provides the following operations for entities called processes:

(I) Processes are disjoint (no shared variables).

(II) Processes can create other processes and can terminate.

(III) Communication is done by asynchronous message passing between partners (processes which share some knowledge e.g. know each other).

LAM allows arbitrary communication and dynamic evolution.

In applications with regular networks more specific abstract machines can be defined by adding additional constraints. This will be explained by RLAM — the Ring Local memory Abstract Machine. RLAM is defined informally by (I) – (III) and

(IV) In RLAM every process has exactly two communication partners, called neighbors. Communication is restricted to neighbors only.

Similarly other topologies can be defined like tree, torus, hypercube and the totally interconnected abstract machines. These abstract machines have to be mapped onto real machines of a given connection structure, e.g. lattice, hypercube or butterfly.

The programming task can now be characterized as in figure 1. The problem is formulated in an

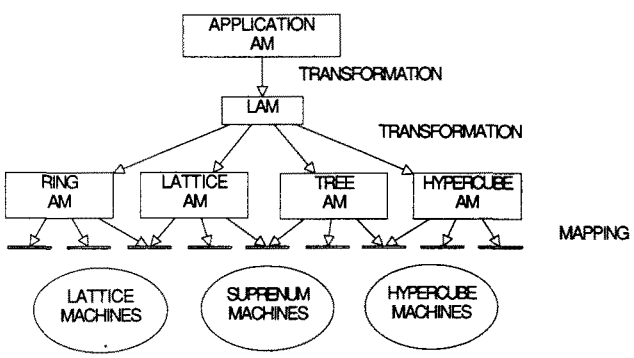

Figure 1: Transformation and mapping of the application to the real machine

application oriented abstract machine, then it is transformed (manually and/or automatically) to an specific abstract machine implementation. This machine will then be mapped onto the topology of the real machine (automatically by the operating system and/or by explicit mapping).

The program, running on a network of abstract or real machines, consists of independent processes, each communicating with partners. This program can be described by a process graph.

A process graph is given by a set of processes P and a neighbourship description K. It can be represented as digraph (P, K). An edge from process p to process q means that q is neighbour of p. So process p can send messages to his neighbour q.

A process structure is called

- **static**, if its number of nodes and the communication structure is fixed and known before runtime. The process–processor placement is done once at the program start and needs not to be changed during its run,
- **multi–phased**, if different communication structures are used during different phases of the computation (see figure 2),
- **dynamic**, if processes and communication links can be created during runtime. This model is more general and needs new concepts for process placement.

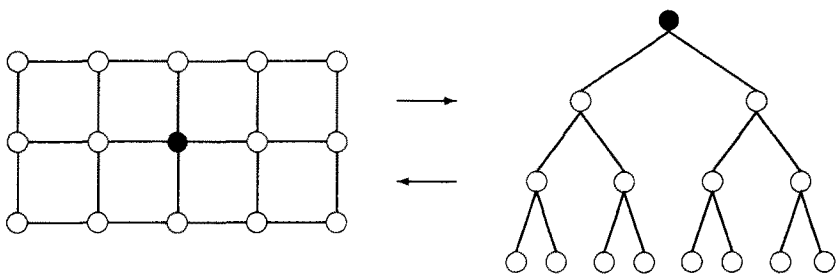

Figure 2: Multiphase computation

Some examples of multi-phased and dynamic networks are discussed in [HoMü86].

3 Mathematical Models for the Mapping Problem

Any distributed memory architecture can be described by the following model:

N	set of processors
C	communication cost
	$C_{m,n}$:= cost per information unit from processor $m \rightarrow n$, $(m, n \in N)$
S	size of storage for each processor, with $S = (S_1, \ldots, S_{\|N\|})$.

A static process structure is described similarly:

P	set of processes
K	communication amount,
	$K_{p,q}$:= information units passed from process $p \rightarrow q$, $(p, q \in P)$
R	amount of storage for each process with $R = (R_1, \ldots, R_{\|P\|})$

Common to the distributed architectures is the fact that the distances between pairs of processors are not equal. So the delay for sending a message depends on the locations of the sending and the receiving processor.

An optimal mapping is given by a function

$$\pi : P \rightarrow N$$

assigning each process to a processor, which minimizes the communication cost Γ_C

$$\Gamma_C^* = \min_{\pi} \Gamma_C = \min_{\pi} \sum_{p,q \in P} K_{p,q} \cdot C_{\pi(p),\pi(q)} \tag{1}$$

s.t.

$$\forall n \in N : \sum_{\substack{p \in P \\ \pi(p)=n}} R_p \leq S_n$$

Γ_C describes the total communication cost, which is defined as the product of the communication amount between all pairs of processes (p, q) and the cost to transport this message between the assigned processes $(\pi(p), \pi(q))$. The assignment has to satisfy the constraint that the storage amount of all processes getting assigned to one processor may not exceed the available storage of this node.

The cost function (1) is a very simple measure and does not take into account

- load balancing
- communication delay.

Several models which incorporate the above factors have been proposed. A weighted sum of CPU-load and communication cost is analyzed in [EBP86]. A communication delay and a time constraint is proposed in [HHR86]. The time T is used implicitely and limits the processing time of each processor. The reduction of T forces more and more processors to be used. All these models are complicated and the question of performance gain is difficult to answer.

A straightforward extension of model (1) is the following function:

$$\Gamma = \min_{\pi} (\Gamma_C + \omega_L \Gamma_L + \omega_B \Gamma_B)$$

Γ_C as defined in (1)
Γ_L penalty for unbalanced CPU-load
Γ_B penalty for network bottlenecks

ω_L and ω_B are constants to balance the weights of the three criteria.
Γ_L can be defined as follows: let T_p be the execution time of process p and L_n the load of processor n, then

$$\Gamma_L = \frac{1}{|N|} \sum_{n \in N} (\overline{L} - L_n)^2 \tag{2}$$

with

$$\overline{L} = \frac{\sum T_p}{|N|} \qquad L_n = \sum_{\substack{p \\ \pi(p)=n}} T_p$$

Γ_L is defined as the variance of the processor loads. Γ_B can be defined analogue.

We believe, however, that these complex models are of limited practical value. They need many parameters, which are seldom known a priori. Heuristics to solve this problem are difficult because three sometimes contradictory criteria influence the solution.

4 The Mapping Strategies

The strategies to solve the mapping problem can be divided into three methods:

- topological mapping,
- mapping by optimizing a cost function,
- adaptive mapping.

Given a machine model (N, C) and a process model (P, K), the following question will be called the **topological mapping** problem:

> Is there a mapping of the process system (P, K) on the machine structure (N, C) such that neighboring processes get assigned to neighboring processors ?

In this general formulation this problem is known to be equivalent to the graph isomorphism problem (see [Bokh81]). This is shown to be NP–complete, such that fast algorithms are not available.

Nevertheless many authors have examined this questions for regular graphs on specific machine architectures (for example [Snyd82] and [SaSc85]). This procedure is useful for the solution of the mapping problem for static process systems and can also be used to examine topological properties of LAMs.

In the case that a topological mapping is not possible, one has to develop fast suboptimal algorithms to get an acceptable solution.

To get a measure for the quality of a solution a mathematical model must be available which describes the communication behavior of the process system on the machine. Central in this **mathematical programming** formulations are the optimization criteria. They enable the algorithms to restrict their search area and lead them into the direction of local optima (heuristic search).

This procedure guarantees good solutions with low computational investments. A variety of heuristics have been developed in different areas where assignment problems arose ([ArRa80], [CHLE80], [ScGa85], [SAKA86]).

We will use the mathematical programming formulation of chapter 3 for the qualitative analysis of the mapping problem.

Both, the topological mapping and mapping as an optimization problem are *feed forward* techniques.

Systems with dynamic process creation need *feed back* strategies taking the actual system state into account. They have to adapt the processor selection to the actual load situation and assign a new process to a suitable processor.

Adaptive mapping can be implemented in a number of ways. In [NiSa86] a dynamic remapping strategy is analyzed and in [ScJo85] user directives are used.

5 Robustness

The tuple (process graph, processor graph) is called **robust**, when the communication costs do not differ significantly on different mapping strategies. If the tuple is robust, then we do not need to look for the optimum mapping.

In the **sensitivity analysis** we compare the results of the optimum mapping to "bad" mapping strategies. The evaluation is done by a comparison of the values of the objective functions of the mathematical model. This analysis can be used to get an answer to the question, how much expense should be invested in the development of mapping strategies.

As representative for the "bad" algorithms we take **random mapping**, which assigns each process to a randomly selected processor. This procedure is bad in the sense that it does not use information about the machine and the process structure. It is possible to find worse mappings, but to do this one has to use structural information. In other words it is as difficult to find a worst mapping than it is to find a best mapping.

In the next chapter we will make an **asymptotic sensitivity analysis** to get a more qualitative evaluation of mapping strategies. In this analysis we consider **graph families** (processor graph, process graph). A graph family is a graph with parameters, like a ring of length l, a $l \times l$ torus, etc.

We start with an important observation:

Lemma 1 *Any mapping strategy is optimal for any processor (process) structure if the process (processor) structure is completely connected.*

This lemma shows that mapping strategies are not very important if the graphs are very densely connected. Mapping strategies may be important for sparsely connected graphs. This will be investigated in the next two chapters.

6 Mapping strategies for SUPRENUM

SUPRENUM is a german supercomputer project. The architecture is described in [BGM86].

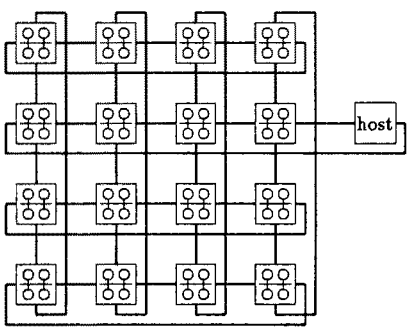

Figure 3: SUPRENUM architecture

The topology of a SUPRENUM multicomputer is shown in figure 3. It consists of $c_1 \times c_2$ clusters, where each cluster has d processors. The processors in a cluster are connected by a bus. The clusters are connected by row and column rings.

The estimation of the communication cost for this architecture is complicated because the communication media are shared. The communication cost is depending on the load of the communication medium, number of nodes and bus protocoll. A queuing network analysis of bus architectures has been done in [MBC83].

Such a complex model is unnecessary for our analysis. We will use a first order approximation taking three parameters into account.

We estimate the communication cost between processor m and processor n for a $(c \times c, d)$ configuration as follows:

$$C_{m,n} = \begin{cases} \varepsilon \approx 0 & m = n \\ d & m,n \text{ in the same cluster} \\ 2d + \alpha c & m,n \text{ in different clusters} \\ 2d + 2\alpha c & m,n \text{ in different rows/columns} \end{cases} \tag{3}$$

ε is the speed of the intraprocessor communication and α is the quotient of the speed of the intracluster bus and the intercluster ring.

The amount of communication K is application dependent. To keep the model simple we investigate regular and homogeneous communication structures (i.e. $K_{p,q} = const$, or $K_{p,q} = 0$).

Definition 1 *Let δ be the expected communication cost between two randomly selected processors.*

Lemma 2 *The communication cost δ for a $(c \times c, d)$ SUPRENUM architecture is given by:*

$$\delta = \frac{1}{c^2}((d-1) + 2(c-1)(2d+\alpha c) + (c-1)^2(2d+2\alpha c)) + \frac{\varepsilon}{c^2 d} \tag{4}$$

Proof : Two randomly selected processors are with probability $1/c^2d$ equal, with probability $(d-1)/c^2d$ within a cluster, with probability $2(c-1)/c^2$ within a row/column and with probability $(c-1)^2/c^2$ on different rows/columns. If we weight the probabilities with the communication cost (3), we obtain (4). □

With $c \to \infty$ or $d \to \infty$, we obtain $\delta \to \infty$. This shows the limited communication bandwidth of this architecture. In the SUPRENUM architecture we have $\alpha \approx 10$ if a dual clusterbus is used, or $\alpha \approx 5$ otherwise.

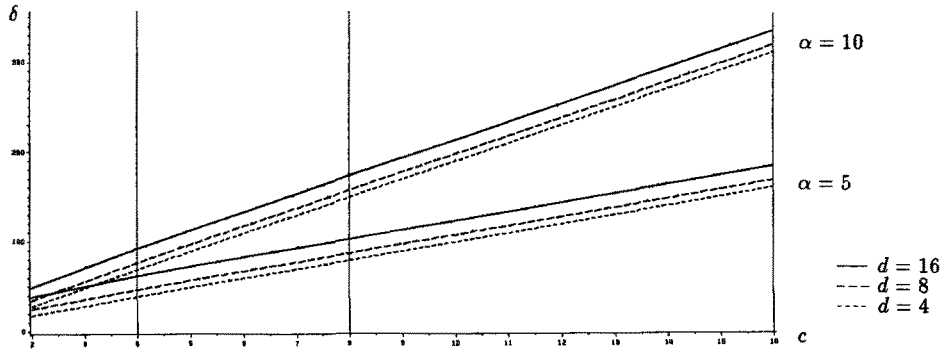

Figure 4: Expected communication cost δ for different SUPRENUM configurations

We compare the communication cost of four different mapping strategies:

OPT	Optimal mapping.
PART	Optimal partition of the process set into as many partitions as processors are available; the mapping is done randomly.
PARTS	Optimal partition of the process set into as many partitions as SUPRENUM clusters are available; optimal mapping within a cluster; the mapping of the partitions is done randomly.
RANDOM	Every process is mapped randomly.

Optimal partitioning minimizes the inter-partition communication.

Table 1 gives the communication cost for some standard process families. The costs are normalized, so that

$$K_{p,q} \cdot C_{\pi(p),\pi(q)} = d,$$

if the two communicating processes are on different processors in the same cluster. ε is assumed to be 0.

	Ring l	2-D Torus $l \times l$
OPT	$c^2d(d-1)+c^2(2d+\alpha c)$	$2cd(\sqrt{d}-1)l+2c(2d+\alpha c)l$
PART	$c^2d\,\delta$	$2c\sqrt{d}\,\delta\,l$
PARTS	$c^2d\,(d-1)+c^2\delta$	$2cd\,(\sqrt{d}-1)\,l+2\,c\,\delta\,l$
RANDOM	$\delta\,l$	$2\,\delta\,l^2$

	3-D Torus $l \times l \times l$	Binary Tree 2^l
OPT	$d^2\,l^2+2c\,(2d+\alpha c)\,l^2$	$c^2d\,(d-1)+(c^2-1)(2d+\alpha c)$
PARTS	$d^2\,l^2+2c\,\delta\,l^2$	$c^2d\,(d-1)+(c^2-1)\,\delta$
RANDOM	$3\,\delta\,l^3$	$(2^l-2)\,\delta$

Table 1: SUPRENUM communication cost

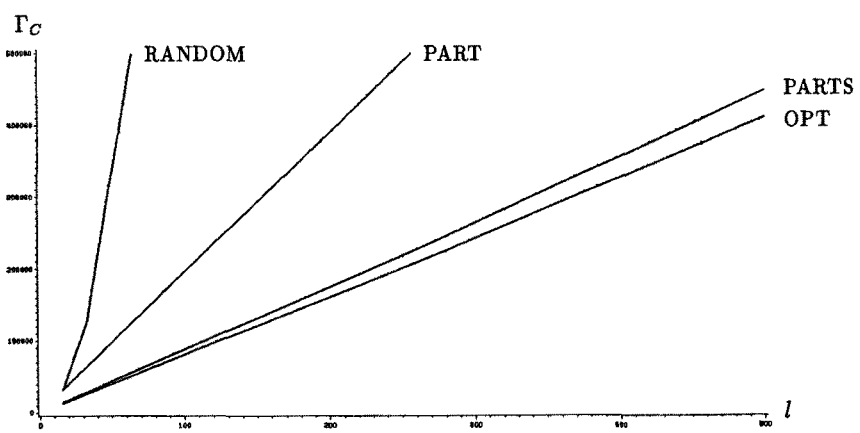

Figure 5: Communication cost for a 2-D torus on SUPRENUM

Proof : We take the 2-D torus as an example. The processors of a SUPRENUM cluster can be interpreted as a $\sqrt{d} \times \sqrt{d}$ grid. This makes it possible to arrange the SUPRENUM processors virtually in a torus of sidelength $c\sqrt{d}$.

The optimal partitioning of the process grid is the regular division into squares. Since $c^2 d$ processors are available, each dimension of the process grid has to be divided by $c\sqrt{d}$ cuts, each containing l communication paths. Each partition gets mapped onto one processor. Since ε is assumed to be 0, communication cost arise only between processes on different processors.

OPT: The mapping is done one-to-one between the partitions of the process-graph and the (virtual) SUPRENUM torus. $2c$ cuts lie on bounds between two neighboring SUPRENUM clusters while $2c(\sqrt{d}-1)$ cuts lie within the clusters. Multiplication of the communication cost (3) and the length l gives

$$\Gamma_C = 2c(\sqrt{d}-1) \cdot d \cdot l + 2c \cdot (2d + \alpha c) \cdot l$$

PART: Optimal partitioning gives $2c\sqrt{d} \cdot l$ paths between partitions. Because of random assignment of these partitions the paths have to be multiplied by the average communication cost

$$\Gamma_C = 2c\sqrt{d} \cdot \delta \cdot l$$

PARTS: The process torus is partitioned into c^2 clusters. $2c(\sqrt{d}-1)$ cuts, each consisting of l links can be handled within the clusters, while the other $2c$ cuts are mapped on random processor pairs, such that the following cost arise:

$$\Gamma_C = 2c(\sqrt{d}-1) \cdot d \cdot l + 2c \cdot \delta \cdot l$$

RANDOM: The 2-D torus of sidelength l contains $2l^2$ links. The processes get mapped randomly, that each link produces the mean communication cost δ

$$\Gamma_C = 2l^2 \cdot \delta$$

The proofs for the other process structures are similar. □

Figure 5 shows the communication cost for a 2-D torus on a $(4 \times 4, 16)$ SUPRENUM machine. We see that OPT and PARTS differ only slightly.

Table 1 shows that this is true for all considered process families. The cost for OPT and PARTS differ in the second term by a factor, the communication between two clusters $(2d + \alpha c)$ for OPT and δ for PARTS. The difference between these two functions increases with growing configuration size. It is about 17% for the average SUPRENUM configuration $(c = 4, d = 16, \alpha = 5)$ and about 41% for the large configuration $(c = 8, d = 16, \alpha = 5)$.

The communication cost for all mapping strategies lie between the values of OPT and RANDOM. OPT and RANDOM differ by a factor which is dependent on the problemsize l. For example the cost for the optimally mapped torus are in $\mathcal{O}(l)$ while the random mapping produces costs in $\mathcal{O}(l^2)$.

We believe however that the SUPRENUM architecture is fairly robust. Optimal mapping is not necessary, good partitioning is sufficient. The partitioning problem can be solved more easily and is discussed in [MüKr87]. A new partitioning algorithm for static process structures with inhomogeneous communication is described in [MGK87].

7 Mapping strategies for Hypercube Architectures

The mapping strategies considered so far for hypercubes concentrate on the topological mapping (see [SaSc85], [BrSc86]). We will analyse different mapping strategies with the cost function (1).

The communication cost $C_{m,n}$ between two processors (m, n) in a hypercube are given by their (hamming–) distance; i.e. a message has to be routed over $C_{m,n}$ communication links.

Let d be the dimension of the hypercube, i.e. the machine consists of 2^d processors.

Lemma 3 *The expected communication cost between two randomly selected processors is* $\delta = d/2$.

We are now able to compare mapping strategies:

OPT Optimal mapping.
PART Optimal partition of the process set into 2^d partitions; the mapping is done randomly.
RANDOM Every process is mapped randomly.

PART and RANDOM need an operating system with a routing facility. The routing problem is discussed in the next chapter.

The communication cost for the standard process families are shown in table 2. The costs are normalized, that $K_{p,q} \cdot C_{\pi(p),\pi(q)} = 1$, if there is a link between $\pi(p)$ and $\pi(q)$.

	Ring	2-D Torus	3-D Torus	Binary Tree
	l	$l \times l$	$l \times l \times l$	2^l
OPT	2^d	$2^{(d/2)+1}\, l$	$3 \cdot 2^{(d/3)}\, l^2$	$2^d - 1$
PART	$\delta\, 2^d$	$\delta\, 2^{(d/2)+1}\, l$	$3\,\delta\, 2^{(d/3)}\, l^2$	$\delta\,(2^d - 1)$
RANDOM	$\delta\, l$	$2\,\delta\, l^2$	$3\,\delta\, l^3$	$\delta\,(2^l - 2)$

Table 2: Hypercube communication cost

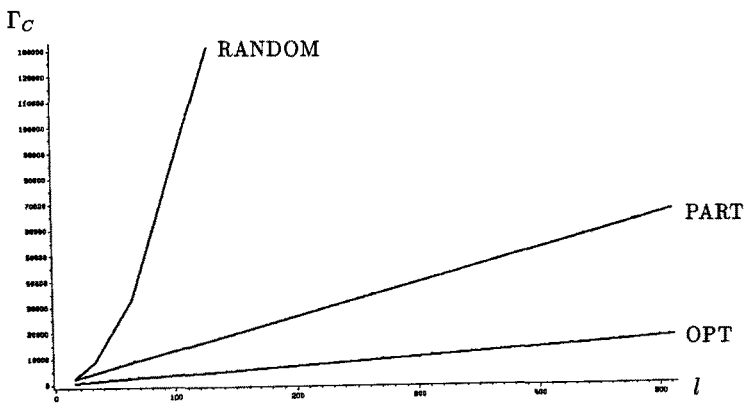

Figure 6: Communication cost for a 2-D torus on the Hypercube

Proof: We take the 2-D torus as an example. The optimal partitioning of the process grid is the regular division into squares. Since 2^d processors are available, each dimension of the process grid has to be divided by $2^{(d/2)}$ cuts, each containing l communication paths. Each partition gets mapped onto one processor. Since the communication cost for two processes on the same processor are assumed to be 0, communication cost arise only between processes on different processors.

OPT: The partitions can be mapped on the processors of the hypercube, that processes in neighboring partitions have a hamming distance of 1 (see [SaSc85]). There exist $2 \cdot 2^{(d/2)} \cdot l$ communication

paths between the partition, giving

$$\Gamma_C = 2^{(d/2)+1} \cdot l$$

PART: Optimal partitioning gives $2 \cdot 2^{(d/2)} \cdot l$ paths between partitions. Because of random assignment these paths have to be multiplied by the average communication cost δ

$$\Gamma_C = \delta \cdot 2^{(d/2)+1} l$$

RANDOM: The 2-D torus of sidelength l contains $2l^2$ links. Each link produces the mean communication cost $d/2$

$$\Gamma_C = d \cdot l^2$$

The proofs for the other process structures are similar. □

Figure 6 shows the functions for a 2-D torus on a Hypercubve of dimension 8.

In all cases the cost for OPT and PART differ by the factor of $\delta = d/2$. This gives 300% for a hypercube of dimension 8 with 256 nodes.

Thus the hypercube architecture is not very robust for the considered standard parallel processing tasks against different mapping strategies. This has also been conjectured in [HMSCP86].

8 Adaptive Routing

The above analysis was based on process networks with point-to-point connections. The basic performance parameter needed is δ, the average processor to processor communication cost.

Process networks with multicast and broadcast have to be handled differently. Ullman [Ullm84b] proposed a generalization of δ for these networks, the "flux". The flux of a network family is inversely proportional to the minimum time it takes to get the data from any set of processors out of that set. The extension of our analysis for this model is straightforward.

Inhomogeneous process networks have to be mapped with a different strategy, which does not try to optimize globally, but only locally. We illustrate this statement with an example. The inhomogeneous ring of figure 7 is to be mapped onto a 2×2 torus.

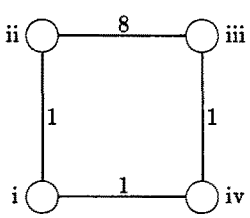

Figure 7: Inhomogeneous ring

We obtain an execution time $T = 8$ for a shift if we assume that CPU and IO overlap. If we can split the communication between process (ii) and (iii) into two independent communication streams, we get $T = 4$ when using two different communication links on the torus.

This example shows that the mapping problem consists of two assignment problems — process to processor and communication path to communication links. Of course this two problems are not independent.

Adaptive process to processor assignment needs process migration. Process migration is a very costly task, if the address space of the process has to be transported to the memory of another processor.

In contrast, adaptive routing is very inexpensive. In adaptive routing the assumption of a static mapping of the communication paths is abandoned. Every message between two processes can be transported by different communication links. Adaptive routing seems to be very promising in processor structures with a rich interconnection structure. A rich interconnection structure is characterized by many different routes between different processors and almost equal communication cost.

Randomized routing is a simple implementation of adaptive routing. It has been proven for hypercubes that random routing gives a nearly optimal communication time for many complex algorithms [Ullm84a]. In the previous chapter we have shown that this statement is not true for simple standard process families.

Adaptive routing for the SUPRENUM architecture is very simple to implement, because only the sending cluster has to decide, whether the row or column bus should be used.

We believe that with a growing number of applications a shift will occur from topological oriented process mapping done a priori to more general adaptive mapping strategies implemented in hardware or operating system. Path mapping (routing) is discussed in detail in [DaSe86].

9 Conclusion

Intricate process mapping strategies are very expensive. Therefore we have investigated the relation between the cost for the mapping and the advantage gained by the mapping.

We have shown that the process mapping can be broken into two subproblems: the process partitioning and the assignment problem. It turned out that for the SUPRENUM architecture a good partitioning of the process graph is sufficient. The partitioning problem is easier to solve than the assignment problem. In contrast the performance of point-to-point distributed architectures, like hypercubes depends more on a good assignment.

The mapping strategies have been analysed separately for different processor graph families (the normalization applied was architecture dependent). In the analysis, the average communication cost δ was used. If the absolute values of δ are given for different architectures it is easily possible to compare different architectures (e.g. a SUPRENUM configuration and a hypercube configuration with 256 processors).

Inhomogeneous process networks are difficult to deal with process mapping strategies. Adaptive routing is more easily to implement and will be the first step towards a more general purpose message-based parallel system.

References

[ArRa80] R. K. Arora, S. P. Rana, *Heuristic Algorithms for Process Assignment in Distributed Computing Systems*, Information Processing Letters 11 (12/80) 199–203

[BGM86] P. M. Behr, W. K. Giloi, H. Mühlenbein, *SUPRENUM: The German Supercomputer Project – Rationale and Concepts*, IEEE Internat. Conference on Parallel Processing (1986)

[Bokh81] S. H. Bokhari, *On the Mapping Problem*, IEEE Transaction on Computers C-30 No. 3 (1981) 207–214

[BrSc86] J. E. Brandenburg, D. E. Scott, *Embedding of Communication Trees and Grids into Hypercubes*, Intel iPSC User Group No. 1 (1986)

[CHLE80] W. W. Chu, L. J. Holloway, M. T. Lan, K. Efe, *Task Allocation in Distributed Data Processing*, Computer (11/80) 57–69

[DaSe86] W. J. Dally, Ch. L. Seitz, *The Torus Routing Chip*, Distributed Computing 1 (1986) 187–196

[EBP86] A. K. Ezzat, R. D. Bergeron, J. L. Pokoski, *Task Allocation Heuristics for Distributed Computing Systems*, IEEE Int. Conf. on Distributed Systems (1986)

[HHR86] C. E. Houstis, E. N. Houstis, J. R. Rice, *Partitioning PDE Computations: Methods and Performance Evaluation*, Report Purdue University (1986)

[HoMü86] H. C. Hoppe, H. Mühlenbein, *Parallel Adaptive Full-Multigrid-Methods on Message-based Multiprocessors*, Parallel Computing 3 (1986) 269–287

[Hock85] R. W. Hockney, *MIMD computing in the USA - 1984*, Parallel Computing 2 (1985) 119–136

[HMSCP86] J. P. Hages, T. Mudge, Q. M. Stow, S. Colley, J. Palmer, *A Microprocessor based Hypercube Supercomputer*, IEEE Micro, Vol. 6, No. 5 (1986)

[MBC83] M. A. Marsan, G. Balbo, G. Conte, *Comparative performance analysis of single bus multiprocessor architectures*, IEEE Trans. Computers, C-31 (1983) 1179–1191

[MGK87] H. Mühlenbein, M. Gorges-Schleuter, O. Krämer, *New Solutions to the Mapping Problem of Parallel Systems — The Evolution Approach*, Parallel Computing, to be published (1987)

[MüKr87] H. Mühlenbein, O. Krämer, *Parallel Solutions of the Graph Partitioning Problem*, to be published

[MKLMS87] H. Mühlenbein, O. Krämer, F. Limburger, M. Mevenkamp, S. Streitz, *Design and Rationale for MUPPET — A Programming Environment for Message-Based Multiprocessors*, Journal of Parallel and Distr. Proc., to be published

[NiSa86] D. Nicol, J. Saltz, *Dynamic Remapping of Parallel Computations with Varying Resource Demands*, ICASE Report 86-45, Nasa Langley (1986)

[SAKA86] C. Saito, H. Amano, T. Kudoh, H. Aiso, *An adaptable cluster structure of (SM)^2 -II*, CONPAR 86, Aachen (9/86) 53–60

[SaSc85] Y. Saad, M. H. Schultz, *Topological Properties of Hypercubes*, Research Report RR–389 Yale University (6/85)

[ScGa85] K. Schwans, C. Gaimon, *Automatic Resource allocation for the CM* Multiprocessor*, 5th Distr. Comp. Conf. (1985)

[ScJo85] K. Schwans, A. K. Jones, *Specifying Resource Allocation for Parallel Programs on the CM* Multiprocessor*, Ohio State University, OSU-CISRC-TR-85-10 (1985)

[Snyd82] L. Snyder, *Introduction to the Configurable, Highly Parallel Computer*, Computer 15 (2/82) 47–56

[Ullm84a] J. D. Ullman, *Computational aspects of VLSI*, Computer Science Press (1984)

[Ullm84b] J. D. Ullman, *Some thoughts about Supercomputer Organization*, Proc. IEEE Compcon, San Francisco (1984) 424–431