Mapping and Load-Balancing Iterative Computations on Heterogeneous Clusters

Arnaud Legrand, Hélène Renard, Yves Robert, and Frédéric Vivien

LIP, UMR CNRS-INRIA-UCBL 5668, École Normale Supérieure de Lyon, France

Abstract. This paper is devoted to mapping iterative algorithms onto heterogeneous clusters. The application data is partitioned over the processors, which are arranged along a virtual ring. At each iteration, independent calculations are carried out in parallel, and some communications take place between consecutive processors in the ring. The question is to determine how to slice the application data into chunks, and to assign these chunks to the processors, so that the total execution time is minimized. One major difficulty is to embed a processor ring into a network that typically is not fully connected, so that some communication links have to be shared by several processor pairs. We establish a complexity result that assesses the difficulty of this problem, and we design a practical heuristic that provides efficient mapping, routing, and data distribution schemes.

1 Introduction

We investigate the mapping of iterative algorithms onto heterogeneous clusters. Such algorithms typically operate on a large collection of application data, which is partitioned over the processors. At each iteration, some independent calculations are carried out in parallel, and then some communications take place. This scheme encompasses a broad spectrum of scientific computations, from mesh based solvers to signal processing, and image processing algorithms. An abstract view of the problem is the following: the iterative algorithm repeatedly operates on a rectangular matrix of data samples. This matrix is split into vertical slices that are allocated to the computing resources. At each step of the algorithm, the slices are updated locally, and then boundary information is exchanged between consecutive slices. This geometrical constraint advocates that processors be organized as a virtual ring. Then each processor only communicates twice, once with its predecessor in the ring, and once with its successor. There is no reason to restrict to a uni-dimensional partitioning of the data, and to map it onto a uni-dimensional ring of processors. But uni-dimensional partitionings are very natural for most applications, and we show that finding the optimal one is already very difficult.

The target architecture is a fully heterogeneous cluster, composed of different-speed processors that communicate through links of different bandwidths. On the architecture side, the problem is twofold: (i) select the processors that participate in the solution and decide for their ordering (which defines the ring); (ii) assign communication routes between each pair of consecutive processors in the ring. One major difficulty of this ring embedding process is that some of the communication routes will (most probably) have to share some physical communication links: indeed, the communication networks

J. Dongarra, D. Laforenza, S. Orlando (Eds.): Euro PVM/MPI 2003, LNCS 2840, pp. 586–594, 2003.

of heterogeneous clusters typically are far from being fully connected. If two or more routes share the same physical link, we have to decide which fraction of the link bandwidth is assigned to each route. Once the ring and the routing have been decided, there remains to determine the best partitioning of the application data. Clearly, the quality of the final solution depends on many application and architecture parameters.

To assess the impact of sharing the link bandwidths, we deal with the simplified version of the problem where we view the target interconnection network as fully connected: between any node pair, the routing is fixed (shortest paths in terms of bandwidth), and the bandwidth is assumed to be that of the slowest link in the routing path. This model is not very realistic, as no link contention is taken into account, but it will lead to a solution ring that can be compared to that obtained with link sharing, providing a way to evaluate the significance of the different hypotheses on the communications.

The rest of the paper is organized as follows. Section 2 is devoted to the precise and formal specification of the previous optimization problem, denoted as SHAREDRING. We also specify the simplified version of the problem denoted as SLICERING. We show that the decision problem associated to SHAREDRING is NP-complete. Section 3 deals with the design of polynomial-time heuristics to solve the SHAREDRING optimization problem. Section 4 is the counterpart for the SLICERING problem. We report some experimental data in Section 5. We state some concluding remarks in Section 6. Due to the lack of space, we refer the reader to [4, 3] for a survey of related papers.

2 Framework

2.1 Modeling the Platform Graph

Computing Costs. The target computing platform is modeled as a directed graph $G = (P, E)$. Each node P_i in the graph, $1 \leq i \leq |P| = p$, models a computing resource, and is weighted by its relative cycle-time w_i: P_i requires w_i time-steps to process a unit-size task. Of course the absolute value of the time-unit is application-dependent, what matters is the relative speed of one processor versus the other.

Communication Costs. Graph edges represent communication links and are labeled with available bandwidths. If there is an oriented link $e \in E$ from P_i to P_j, b_e denotes the link bandwidth. It takes L/b_e time-units to transfer one message of size L from P_i to P_j using link e. When several messages share the link, each of them receives a fraction of the available bandwidth. The fractions of the bandwidth allocated to the messages can be freely determined by the user, except that the sum of all these fractions cannot exceed the total link bandwidth. The eXplicit Control Protocol XCP [2] does enable to implement a bandwidth allocation strategy that complies with our hypotheses.

Routing. We assume we can freely decide how to route messages between processors. Assume we route a message of size L from P_i to P_j, along a path composed of k edges $e_1, e_2, \ldots, e_k$. Along each edge e_m, the message is allocated a fraction f_m of the bandwidth b_{e_m}. The communication speed along the path is bounded by the link allocating the smallest bandwidth fraction: we need L/b time-units to route the message, where $b = \min_{1 \leq m \leq k} f_m$. If several messages simultaneously circulate on the network and happen to share links, the total bandwidth capacity of each link cannot be exceeded.

Application Parameters: Computations. W is the total size of the work to be performed at each step of the algorithm. Processor P_i performs a share $\alpha_i.W$, where $\alpha_i \geq 0$ for $1 \leq i \leq p$ and $\sum_{i=1}^{p} \alpha_i = 1$. We allow $\alpha_j = 0$, meaning that processor P_j do not participate: all resources are not involved if extra communications incurred by adding more processors slow down the whole process, despite the increased cumulated speed.

Application Parameters: Communications in the Ring. We arrange the participating processors along a ring. After updating its data slice, each active processor sends a message of fixed length H to its successor. To illustrate the relationship between W and H, we can view the original data matrix as a rectangle composed of W columns of height H, so that one single column is exchanged between consecutive processors in the ring.

Let $\text{succ}(i)$ and $\text{pred}(i)$ denote the successor and the predecessor of P_i in the virtual ring. There is a communication path $\mathcal{S}_i$ from P_i to $P_{\text{succ}(i)}$ in the network: let $s_{i,m}$ be the fraction of the bandwidth b_{e_m} of the physical link e_m that is allocated to the path $\mathcal{S}_i$. If a link e_r is not used in the path, then $s_{i,r} = 0$. Let $c_{i,\text{succ}(i)} = \frac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$: P_i requires $H.c_{i,\text{succ}(i)}$ time-units to send its message of size H to its successor $P_{\text{succ}(i)}$. Similarly, we define the path $\mathcal{P}_i$ from P_i to $P_{\text{pred}(i)}$, the bandwidth fraction $p_{i,m}$ of e_m allocated to $\mathcal{P}_i$, and $c_{i,\text{pred}(i)} = \frac{1}{\min_{e_m \in \mathcal{P}_i} p_{i,m}}$.

Objective Function. The total cost of one step in the iterative algorithm is the maximum, over all participating processors, of the time spent computing and communicating:

$$T_{\text{step}} = \max_{1 \leq i \leq p} \mathbb{I}\{i\}[\alpha_i.W.w_i + H.(c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)})]$$

where $\mathbb{I}\{i\}[x] = x$ if P_i is involved in the computation, and 0 otherwise. In summary, the goal is to determine the best way to select q processors out of the p available, to assign them computational workloads, to arrange them along a ring and to share the network bandwidth so that the total execution time per step is minimized.

2.2 The SHAREDRING Optimization Problem

Definition 1 (SHAREDRING(p,w_i,E,b_{e_m},W,H)). *Given p processors P_i of cycle-times w_i and $|E|$ communication links e_m of bandwidth b_{e_m}, given the total workload W and the communication volume H at each step, minimize*

$$T_{step} = \min_{1 \leq q \leq p} \min_{\substack{\sigma \in \Theta_{q,p} \\ \sum_{i=1}^{q} \alpha_{\sigma(i)} = 1}} \max_{1 \leq i \leq q} \left(\alpha_{\sigma(i)}.W.w_{\sigma(i)} + H.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)})\right) \quad (1)$$

In Equation 1, $\Theta_{q,p}$ denotes the set of one-to-one functions $\sigma : [1..q] \to [1..p]$ which index the q selected processors that form the ring, for all candidate values of q between 1 and p. For each candidate ring represented by such a σ function, there are constraints hidden by the introduction of the quantities $c_{\sigma(i),\sigma(i-1 \bmod q)}$ and $c_{\sigma(i),\sigma(i+1 \bmod q)}$, which we gather now. There are $2q$ communicating paths, the path $\mathcal{S}_i$ from $P_{\sigma(i)}$ to its successor $P_{\text{succ}(\sigma(i))} = P_{\sigma(i+1 \bmod q)}$ and the path $\mathcal{P}_i$ from $P_{\sigma(i)}$ to its predecessor $P_{\text{pred}(\sigma(i))} = P_{\sigma(i-1 \bmod q)}$, for $1 \leq i \leq q$. For each link e_m in the interconnection

network, let $s_{\sigma(i),m}$ (resp. $p_{\sigma(i),m}$) be the fraction of the bandwidth b_{e_m} that is allocated to the path $\mathcal{S}_{\sigma(i)}$ (resp. $\mathcal{P}_{\sigma(i)}$). We have the equations:

$$\begin{cases} 1 \leq i \leq q, \;\; 1 \leq m \leq E, \;\; s_{\sigma(i),m} \geq 0, \;\; p_{\sigma(i),m} \geq 0, \;\; \sum_{i=1}^{q} (s_{\sigma(i),m} + p_{\sigma(i),m}) \leq b_{e_m} \\ 1 \leq i \leq q, \;\; c_{\sigma(i),\text{succ}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{S}_{\sigma(i)}} s_{\sigma(i),m}}, \;\; c_{\sigma(i),\text{pred}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{P}_{\sigma(i)}} p_{\sigma(i),m}} \end{cases}$$

Since each communicating path $\mathcal{S}_{\sigma(i)}$ or $\mathcal{P}_{\sigma(i)}$ will typically involve a few edges, most of the quantities $s_{\sigma(i),m}$ and $p_{\sigma(i),m}$ will be zero. In fact, we have written $e_m \in \mathcal{S}_{\sigma(i)}$ if the edge e_m is actually used in the path $\mathcal{S}_{\sigma(i)}$, i.e. if $s_{i,m}$ is not zero.

From Equation 1, we see that the optimal solution involves all processors as soon as the ratio $\frac{W}{H}$ is large enough: then the impact of the communications becomes small in front of the cost of the computations, and the computations should be distributed to all resources. Even in that case, we have to decide how to arrange the processors along a ring, to construct the communicating paths, to assign bandwidths ratios and to allocate data chunks. Extracting the "best" ring seems to be a difficult combinatorial problem.

2.3 The SLICERING Optimization Problem

We denote by SLICERING the simplified version of the problem without link sharing. The SLICERING problem is exactly given by Equation 1 with an important simplification concerning routing paths and communication costs: the routing path between any node pair is fixed, as well as its bandwidth. This amounts to assuming a fully connected interconnection network where the bandwidth between P_i and P_j has a constant value. Given a "real" network, we define $c_{i,j}$ as the inverse of the smallest link bandwidth of a path of maximal bandwidth that goes from P_i to P_j. This construction does not take contentions into account: if the same link is shared by several paths, the available bandwidth for each path is over-estimated.

We summarize the construction of the simplified problem as follows: take the actual network as input, and compute shortest paths (in terms of bandwidths) between all processor pairs. This leads to a (fake) fully connected network. Now in Equation 1, the cost of all communication paths is given. But there remains to determine the optimal ring, and to assign computing workloads to the processors that belong to the ring.

2.4 Complexity

The following result states the intrinsic difficulty of the SHAREDRING problem (the same result holds for the simplified SLICERING problem; see [3, 4] for the proofs):

Theorem 1. *The decision problem associated to the* SHAREDRING *optimization problem is NP-complete.*

3 Heuristic for the SHAREDRING Problem

We describe, in three steps, a polynomial-time heuristic to solve SHAREDRING: (i) the greedy algorithm used to construct a solution ring; (ii) the strategy used to assign bandwidth fractions during the construction; and (iii) a final refinement.

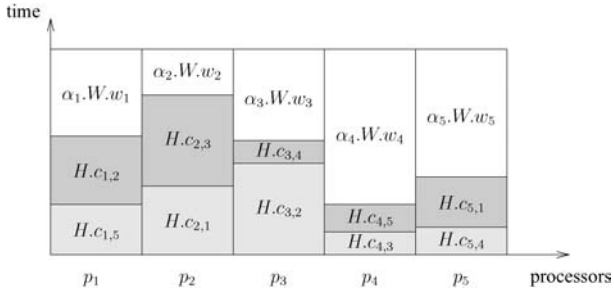

Fig. 1. Summary of computation and communication times with $q = 5$ processors.

3.1 Ring Construction

We consider a solution ring involving q processors, numbered from P_1 to P_q. Ideally, all these processors should require the same amount of time to compute and communicate: otherwise, we would slightly decrease the computing load of the last processor and assign extra work to another one (we are implicitly using the "divisible load" framework [3]). Hence (see Figure 1) we have for all i (indices being taken modulo q):

$$T_{\text{step}} = \alpha_i.W.w_i + H.(c_{i,i-1} + c_{i,i+1}). \tag{2}$$

Since $\sum_{i=1}^{q} \alpha_i = 1$, $\sum_{i=1}^{q} \frac{T_{\text{step}} - H.(c_{i,i-1}+c_{i,i+1})}{W.w_i} = 1$. With $w_{\text{cumul}} = \frac{1}{\sum_{i=1}^{q} \frac{1}{w_i}}$:

$$T_{\text{step}} = W.w_{\text{cumul}} \left(1 + \frac{H}{W} \sum_{i=1}^{q} \frac{c_{i,i-1} + c_{i,i+1}}{w_i} \right) \tag{3}$$

We use Equation 3 as a basis for a greedy algorithm which grows a solution ring iteratively, starting with the best pair of processors. Then, it iteratively includes a new node in the current solution ring. Assume we already have a ring of r processors. We search where to insert each remaining processor P_i in the current ring: for each pair of successive processors (P_j, P_k) in the ring, we compute the cost of inserting P_i between P_j and P_k. We retain the processor and pair that minimize the insertion cost. To compute the cost of inserting P_i between P_j and P_k, we resort to another heuristic to construct communicating paths and allocate bandwidth fractions (see Section 3.2) in order to compute the new costs $c_{k,j}$ (path from P_k to its successor P_j), $c_{j,k}$, $c_{k,i}$, and $c_{i,k}$. Once we have these costs, we compute the new value of T_{step} as follows:

- We update w_{cumul} by adding the new processor P_k into the formula.
- In $\sum_{s=1}^{r} \frac{c_{\sigma(s),\sigma(s-1)} + c_{\sigma(s),\sigma(s+1)}}{w_{\sigma(s)}}$, we suppress the two terms corresponding to the two paths between P_i to P_j and we insert the new terms $\frac{c_{k,j}+c_{k,i}}{w_k}$, $\frac{c_{j,k}}{w_j}$ and $\frac{c_{i,k}}{w_i}$.

This step of the heuristic has a complexity proportional to $(p - r).r$ times the cost to compute four communicating paths. Finally, we grow the ring until we have p processors. We return the minimal value obtained for T_{step}. The total complexity is

$\sum_{r=1}^{p}(p-r)rC = O(p^3)C$, where C is the cost of computing four paths in the network. Note that it is important to try all values of r, because T_{step} may not vary monotonically with r.

3.2 Bandwidth Allocation

We now assume we have a r-processor ring, a pair (P_i, P_j) of successive processors in the ring, and a processor P_k to be inserted between P_i and P_j. Together with the ring, we have built $2r$ communicating paths to which a fraction of the initial bandwidth has been allocated. To build the four paths involving P_k, we use the graph $G = (V, E, b)$ where $b(e_m)$ is what has been left by the $2r$ paths of the bandwidth of edge e_m. First we re-inject the bandwidths fractions used by the communication paths between P_i and P_j. Then to determine the four paths, from P_k to P_i and P_j and vice-versa:

- We independently compute four paths of maximal bandwidth, using a standard shortest path algorithm in G.
- If some paths happen to share some links, we use a brute force analytical method to compute the bandwidth fractions minimizing Equation 3 to be allocated.

Then we can compute the new value of T_{step} as explained above, and derive the values of the workloads α_i. The cost C of computing four paths in the network is $O(p + E)$.

3.3 Refinements

Schematically, the heuristic greedily grows a ring by peeling off the bandwidths to insert new processors. To diminish the cost of the heuristic, we never re-calculate the bandwidth fractions that have been previously assigned. When the heuristic ends, we have a q-processor ring, q workloads, $2q$ communicating paths, bandwidth fractions and communication costs for these paths, and a feasible value of T_{step}. As the heuristic could appear over-simplistic, we have implemented two variants aimed at refining its solution. The idea is to keep everything but the bandwidth fractions and workloads. Once we have selected the processor and the pair minimizing the insertion cost in the current ring, we perform the insertion and recompute all bandwidth fractions and workloads. We can re-evaluate bandwidth fractions using a global approach (see [3] for details):

Method 1: Max-min fairness. We compute the bandwidths fractions using the traditional bandwidth-sharing algorithm [1] which maximizes the minimum bandwidth allocated to a path. Then we compute the α_i so as to equate all execution times (computations followed by communications), thereby minimizing T_{step}.

Method 2: quadratic resolution using the KINSOL software. Once we have a ring and all the communicating paths, the program to minimize T_{step} is quadratic in the unknowns α_i, $s_{i,j}$ and $p_{i,j}$. We use the KINSOL library [5] to solve it.

4 Heuristic for the SLICERING Problem

The greedy heuristic for the SLICERING problem is similar to the previous one. It starts by selecting the fastest processor and iteratively includes a new node in the current

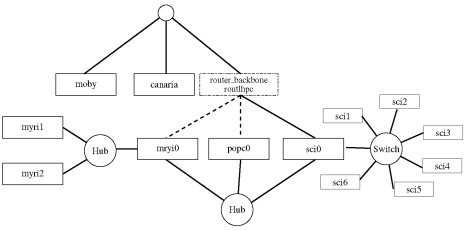

Fig. 2. Topology of the Lyon platform.

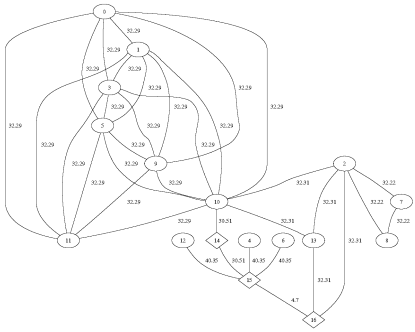

Fig. 3. Abstraction of the Lyon platform.

Table 1. Processor cycle-times (in seconds per megaflop) for the Lyon and Strasbourg platforms.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
0.0206	0.0206	0.0206	0.0206	0.0291	0.0206	0.0087	0.0206	0.0206	0.0206	0.0206	0.0206	0.0291	0.0451	0	0	0

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}
0.0087	0.0072	0.0087	0.0131	0.016	0.0058	0.0087	0.0262	0.0102	0.0131	0.0072	0.0058	0.0072	0	0	0	0	0	0

solution ring. Assume we have a ring of r processors. For each remaining processor P_i, for each pair of successive processors (P_j, P_k) in the ring, we compute the cost of inserting P_i between P_j and P_k. We retain the processor and the pair minimizing the insertion cost. This step of the heuristic has a complexity proportional to $(p-r).r$. We grow the ring until we have p processors, and we return the minimal value obtained for T_{step}. The total complexity is $\sum_{r=1}^{p}(p-r)r = O(p^3)$. It is important to try all values of r as T_{step} may not vary monotically with r. See [4] for further details.

5 Experimental Results

5.1 Platform Description

We experimented with two platforms, one located in ENS Lyon and the other one in the University of Strasbourg. Figures 2 and 3 show the Lyon platform which is composed of 14 computing resources and 3 routers. In Figure 3, circled nodes 0 to 13 are the processors, and diamond nodes 14 to 16 are the routers. Edges are labeled with link bandwidths. Similarly,the Strasbourg platform is composed of 13 computing resources and 6 routers. Processor cycle-times for both platforms are gathered in Table 1.

5.2 Results

For both topologies, we evaluate the impact of link sharing as follows. In the first heuristic, we build the solution ring without taking link sharing into account. Using the abstract graph in Figure 3, we run an all-pair shortest distance algorithm to determine the bandwidth between any pair of nodes, thereby simulating a fully connected interconnection network. Then we return the solution ring computed by the greedy heuristic

Table 2. T_{step}/W for each heuristic for the Lyon and Strasbourg platforms respectively.

Ratio H/W	H1 : slice-ring	H2 : shared-ring	Improvement
0.1	3.17	3.15	0.63%
1	3.46	3.22	6.94%
10	10.39	3.9	62.46%

Ratio H/W	H1 : slice-ring	H2 : shared-ring	Improvement
0.1	7.32	7.26	0.82%
1	9.65	7.53	21.97%
10	19.24	10.26	46.67%

for the SLICERING problem, as described in Section 4. The value of T_{step} achieved by the heuristic may well not be feasible, as the actual network is not fully connected. Therefore, we keep the ring and the communicating paths between adjacent processors in the ring, and we compute feasible bandwidth fractions using the quadratic programming software. The second heuristic is the greedy heuristic designed in Section 3 for the SHAREDRING problem, using the quadratic programming refinement. The major difference between the two heuristics is that the latter takes link contention into account when building up the solution ring. To compare the value of $\frac{T_{step}}{W}$ returned by both algorithms, we use various communication-to-computation ratios. Table 2 shows these values for each platform. From these experiments we conclude that:

- When the impact of communication costs is low, the main goal is to balance computations, and both heuristics are equivalent.
- When the communication-to-computation ratio becomes more important, the effect of link contention becomes clear, and the second heuristic's solution is much better.

As a conclusion, we point out that an accurate modeling of the communications has a dramatic impact on the performance of the load-balancing strategies.

6 Conclusion

The major limitation to programming heterogeneous platforms arises from the additional difficulty of balancing the load. Data and computations are not evenly distributed to processors. Minimizing communication overhead becomes a challenging task. In this paper, the major emphasis was towards a realistic modeling of concurrent communications in cluster networks. One major result is the NP-completeness of the SHAREDRING problem. Rather than the proof, the result itself is interesting, because it provides yet another evidence of the intrinsic difficulty of designing heterogeneous algorithms. But this negative result should not be over-emphasized. Indeed, another important contribution of this paper is the design of an efficient heuristic, that provides a pragmatic guidance to the designer of iterative scientific computations. The importance of an accurate modeling of the communications, that takes contentions into full account, has been made clear by the experimental results. Our heuristic makes it possible to efficiently implement iterative computations on commodity clusters made up of several heterogeneous resources, which is a promising alternative to using costly supercomputers.

References

1. D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.
2. D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM SIGCOMM 2002*, pages 89–102. ACM Press, 2002.
3. A. Legrand, H. Renard, Y. Robert, and F. Vivien. Load-balancing iterative computations in heterogeneous clusters with shared communication links. Research Report RR-2003-23, LIP, ENS Lyon, France, Apr. 2003.
4. H. Renard, Y. Robert, and F. Vivien. Static load-balancing techniques for iterative computations on heterogeneous clusters. Research Report RR-2003-12, LIP, ENS Lyon, 2003.
5. A. Taylor and A. Hindmarsh. User documentation for KINSOL. Tech. Rep. UCRL-ID-131185, Lawrence Livermore Nat. Lab., July 1998.