New program model for program partitioning on NUMA multiprocessor systems

G.-J.Lai
C.Chen

Indexing terms: Program model, NUMA multiprocessor, Partitioning, Scheduling

Abstract: A new program model is presented to accurately represent parallel programs for partitioning and scheduling problems. This model extends the graphic representation of the macrodataflow by considering the complex communication options supported by NUMA systems. The proposed model shows not only task precedence relations but also data sharing status. Moreover, a new partitioning method based on the proposed model is also developed. Experimental results show that many conventional partitioning algorithms operate more efficiently using the proposed model, and that the proposed algorithm surpasses existing algorithms.

1 Introduction

Partitioning a program into parallel tasks and scheduling them are essential steps in parallel compilation systems [1–3]. Parallel compilers manipulate partitioning and scheduling problems according to the information supplied by program models. Accurate representation of program models is therefore one of the most important issues in parallel compilation systems. Wolski and Feo [2] stated that macrodataflow model [1] is not adequate for NUMA systems. They provided thorough details and motivations for a new program model (memory node model) for NUMA systems. However, their work still does not precisely characterise programs for NUMA systems. Their work hypothesised that communication edges associated with the same memory location may have no different access latency. This assumption is true only for multiprocessor systems with physical memory modules that are uniformly shared by all processors. The global address space is scattered over physically separated memory modules for NUMA multiprocessors with distributed shared memory systems (for example, the BBN TC-2000 Butterfly). Access latencies for the same memory location addressed by different processors are not identical in such NUMA systems; only one of these processors is local with respect to this memory address, all the others are remote processors. Moreover, Wolski and Feo hypothesised that the earliest starting time of a task is the time after all of its predecessor tasks finishing data transmission. So, even after the necessary data having already arrived, this task still can not be issued if any one of its predecessors has not finish the data transmission to other successors. The memory node model therefore cannot precisely characterise programs for NUMA multiprocessor systems.

In this paper, a new program model is offered, and a new partitioning heuristic based on this model is established for evaluating performance improvement. The proposed model is called the shared communication resource (SCR) model. It enlarges the macrodataflow program description to allow employment of partitioning heuristics for NUMA systems. It not only indicates precedence relations but also the status of data sharing within programs as well as resource contention associated with system models. A new partitioning algorithm, referred to as the shortest extended critical path first (SECPF), is also presented here to demonstrate excellent qualities of the SCR model. It considers the complex communication options, data sharing and resource contention in the SCR model. Experimental results show that adoption of the SCR model can achieve more realistic outcomes for the partitioning problem, and that parallel partitioned tasks execute well on NUMA systems.

2 Shared communication resource model

The main idea behind the SCR model is that tasks generally communicate through some 'shared' communication resources, such as registers, caches (when the tasks are in the same processor) and shared memories (when the tasks are in different processors). Therefore, a communication operation is divided into two phases in the SCR model. In the first, the producer task sends data to a 'shared' communication resource, and in the second, the consumer task receives this data from the 'shared' communication resource. The 'shared' communication resource is referred to as the shared communication resource (SCR) node in the SCR model.

Now, we describe the SCR model formally as follows. A program is represented as a directed acyclic graph (DAG) based on the SCR model. The DAG is defined by a tuple $G = (N_t, N_s, E, C, T)$, where N_t is the set of task nodes, N_s is the set of SCR nodes, C is the set of communication volumes, T is the set of com-

IEE Proceedings online no. 19960637

Paper first received 27th June 1995 and in revised form 29th April 1996

The authors are with the National Chiao Tung University, Institute of Computer Science and Information Engineering, 1001 Ta Hsueh Road, Hsinchu, Taiwan, Republic of China

putation costs, E is the set of communication edges which define a partial order or precedence constraints on $N_t \cup N_s$. There is no communication edge between n_i and n_j when $n_i, n_j \in N_t$ or $n_i, n_j \in N_s$. The value of $c_{ij} \in C$ is the communication volume occurring along the edge $e_{ij} = (n_i, n_j) \in E$, either $n_i \in N_t$, $n_j \in N_s$ or $n_i \in N_s$, $n_j \in N_t$. The value $\tau_i \in T$ is the computation time for node $n_i \in N_t$, and $\tau_i = 0$ for all $n_i \in N_s$. When there is data dependence between tasks n_i, and n_j, n_i and $n_j \in N_t$, there exists a node $n_s \in N_s$ such that two edges (n_i, n_s) and $(n_s, n_j) \in E$. An example of the DAG is shown in Fig. 1*c* with three tasks n_1, n_2 and n_3.

For the SCR model, we have the following assumptions:

(i) A task is an indivisible unit of computation, which maybe an assigninent statement, a subroutine or even an entire program.

(ii) Tasks are convex, which means that once task execution begins, it must continue to completion without interruption [1].

(iii) Only one task at a time can access data from one SCR node in this model. Two independent tasks must therefore access data from the same SCR node in sequence, making resource contention issue.

(iv) Task execution is triggered by satisfying precedence constraints and removing resource contentions.

(v) Precedence constraints occur when the execution of one task must be postponed until the arrival of all necessary data.

(vi) The SCR model includes two kinds of resource contentions: one in which execution of a task must be deferred, until the completion of all the tasks scheduled before it in the same processor; and another that entails receiving data from immediate predecessors sequentially, this is a task cannot receive data from all its predecessors simultaneously.

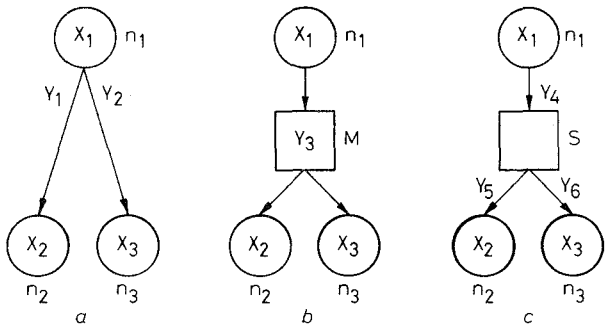

Fig. 1 *Examples of three models*
$n_1, \ldots, n_3$ task nodes
$X_1, \ldots, X_3$ computation times
$Y_1, \ldots, Y_6$ communication volumes
M memory node
S SCR node

An example of the same program segment represented according to three models is shown in Fig. 1. Fig. 1*a* shows the DAG represented by the macrodataflow. The DAG represented by the memory node model is shown in Fig. 1*b*. In memory node model, each communication is divided into three phases, write, communicate and read. Two memory nodes are inserted along each communication edge to represent the write–communicate–read sequence. These two memory nodes are merged into one node, if the communication is through shared memories. In Fig. 1*b*, data transferred from n_1 to n_2 and n_3 are assumed to pass through the same output port and the system is supposed to be a shared memory multiprocessor system. The representation of the SCR model is shown in Fig. 1*c*. Communication between two tasks is divided into two phases, the write phase and the read phase. Only one SCR node is inserted along the communication edge to indicate the 'shared' communication resource. The values within the task nodes indicate their computation times, and those at the communication edges are the communication volumes.

Since the memory node model supposes that the access latencies associated with the same memory node are identical for the same access direction (i.e. either write or read operations), it cannot represent complex communication options such as the case in which two consumer tasks access the same memory node from different processors. Even if the access latencies associated with the same memory node could be different in memory node model, the number of memory node types increases profusely as the number of access latency types increases. Consequently, the time complexity of any algorithms which applies such a model could not be polynomial. However, SCR model can depict such kinds of complex communications; if the nodes n_2 and n_3 shown in Fig. 1*c* are allocated to different processors, the communication latencies along the edges (S, n_2) and (S, n_3) could be different values.

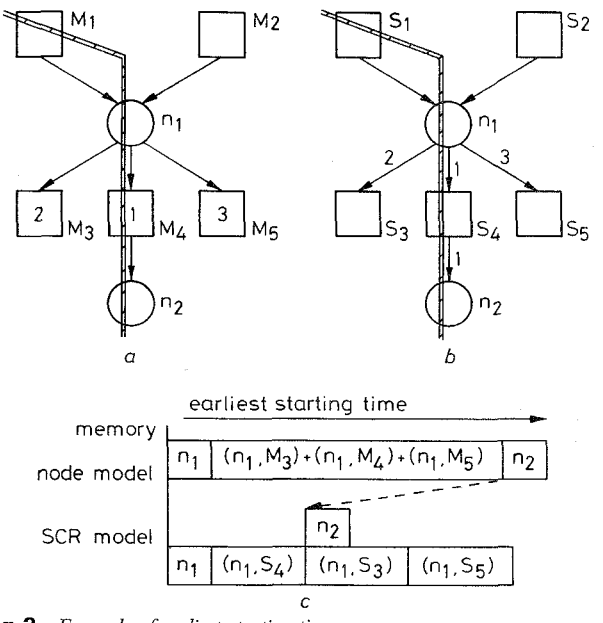

Fig. 2 *Example of earliest starting time*
a Memory node
b SCR model
$M_1, \ldots, M_5$ memory nodes
n_1, n_2 task nodes
$S_1, \ldots, S_5$ SCR nodes
——— critical path

The memory node model hypothesised that the earliest starting time of a task is the time after all of its predecessor tasks finishing data transmission. Therefore, n_2 cannot start execution, until n_1 completes all its write operations [2] as shown in Fig. 2*c*. If the communication latency of each edge is supposed to be 3 cycles/byte, then, as shown in Fig. 2*c*, n_2 must wait $21(2 \times 3 + 1 \times 3 + 1 \times 3 + 3 \times 3)$ cycles to start its execution in the memory node model. If the start time to execute n_2 could be advanced, then the critical path of the program could be shortened. In the SCR model, a task can start its execution as soon as precedence constraints

and resource contentions affecting it have been resolved. The earliest start execution time for n_2 is therefore $6(1 \times 3 + 1 \times 3)$ cycles after n_1 is completed in SCR model, as depicted in Fig. 2*b*. The SCR model thus saves 15 cycles along the critical path when compared with memory node model, and then improves the resource utilisation.

To summarise, the SCR model has several benefits. First, the representation is simple, consistent and system-independent. Second, it can express complex communication options. Third, it can easily characterise programs for NUMA systems such as shared local memories (for example, the BBN Butterfly), the hierarchical cluster model (for example, the Cedar system at the University of Illinois), and even Cache-Coherent NUMA systems (for example, the DASH at Stanford and the Alewife at MIT). Fourth, the data sharing could be shown in the SCR model. For example, a producer task sends data to an SCR node only once and consumer tasks then access this data from this SCR node sequentially, when these consumers would like to receive the same data from the same producer. Finally, SCR model explicitly reveals the memory access contention relationships within programs.

3 SECPF partitioning algorithm

A new partitioning heuristic, SECPF, based on the SCR model is presented in this section. We only consider the nonbacktracking algorithm here to avoid high complexity. This means once the partitions have been generated by some steps in the algorithm, they cannot be unpartitioned afterwards. The number of partitioning steps thus remains polynomial-bounded with respect to the size of the DAG.

Minimising the parallel execution time for a multiprocessor system with an unbounded number of processors is the goal of this partitioning algorithm. Here we suppose that the multiprocessor system is homogeneous, and communications between any two tasks are half-duplex. Each processor element (PE) has a processor and a coprocessor to deal, respectively, with computation and communication, which allows computation and communication to be overlapped if they are independent of each other. Formally, let $P = \{p_i | \ i = 1, ..., |P|\}$, $|P| \leq \infty$, be the set of identical processors in the multiprocessor system. Let $P(n_i)$ be the PE allocated by n_i. Also, parameter $\eta(p_i, p_j)$ denotes the communication latency required to transfer a message unit from processor p_i to processor p_j, where $p_i, p_j \in P$. To simplify the verification of our algorithm, only two kinds of communication latencies are considered here to compare existing algorithms with the proposed algorithm. The set of communication latencies $M = \{L_1, L_r\}$, where $L_1 = \eta(p_i, p_i)$ $p_i \in P$, is the intraprocessor communication latency occurring when two tasks communicate with each other in the same PE, and $L_r = \eta(p_i, p_j)$ $p_i, p_j \in P$ and $i \neq j$, is the interprocessor communication latency occurring when two tasks communicate with each other through an interconnection network. System models with more latency types will be contemplated for future investigations of the advantages of the SCR model.

The initial communication latencies of DAG edges are considered L_r. If a partitioned DAG is modified (a) by assigning L_1 to the edge between any pair of nodes n_a and n_b of a partition, where n_b is executed immediately after n_a, and there is a data dependence edge between n_a and n_b, or (b) by adding a zero-weighted edge between any pair of nodes n_a and n_b of a partition, where n_b is executed immediately after n_a, and there is no data dependence between n_a and n_b, then a 'scheduled DAG' is obtained.

The 'earliest starting time' (*est*) of node n, *est*(n) [9], in a scheduled DAG is defined as the earliest time after satisfying precedence constraints and removing resource contention, that node n can begin execution.

The 'longest completion time' (*lct*) of a node is the longest execution time from this node to the sink node. Mathematically, $\forall\ n_i \in N_t \cup N_s$,

$$\mathrm{lct}(n_i) = \max_{n_j \in \mathrm{succ}(n_i)} (\mathrm{lct}(n_j) + \tau_j) + \sum_{n_j \in \mathrm{succ}(n_i)} (c_{ij} \times L_r)$$

where succ(n_i) is the set of immediate successors of n_i.

The 'extended critical path' (*ecp*) of a scheduled DAG is the longest path from the source node to the sink node. The *ecp* length is

$$ecp(G) = \max(est(n) + \tau_i + lct(n_i)), \text{for all } n_i \in N_t \cup N_s$$

The SECPF algorithm process is to reduce the *ecp*(G) by assigning L_1 to edges along the *ecp*; then nodes with the most uncompleted jobs are issued as soon as possible. If this *ecp* is unique, it will reduce the parallel completion time. SECPF initially assigns a default latency to all communication edges, then calculates the *lct* for each node bottom up. Nodes with no predecessors are selected first. Candidates with maximum *est* and *lct* sums are allocated to existing partitions or to new partitions after the *est* of each candidate in the current step has been determined; chosen candidate nodes are then examined. The algorithm repeats this procedure, until all nodes have been examined. The SECPF pseudo program is given as follows:

Algorithm SECPF_Partitioning

Input: $G = (N_t, N_s, E, C, T)$ and $M = \{L_1, L_r\}$.

Output: The partitioned program with the minimal parallel execution time.

Begin

Assign each communication edge to be L_r.

Calculate *lct*(n), $\forall\ n \in N_t \cup N_s$.

Select nodes without predecessors as candidates.

Calculate *ecp*(G).

While there exists some unexamined node

{ Calculate the *est*s for all candidate nodes.

Choose the candidate node with max(*ecp*(G)).

Calculate the possible *ecp*s for this chosen candidate.

If there are possible *ecp*s smaller than the *ecp* in the previous step

{Allocate the chosen candidate to the partition with the minimal possible *ecp*.}

else

{Allocate the chosen candidate to a new partition.}

Set this node to be examined, and update the *ecp* in the current step.

Re-select the nodes without unexamined predecessors to be candidates.

}

End

SECPF algorithm has been implemented on the SCR model. It is similar to the DSC [3], however, considers the communication contentions. The SECPF algorithm deals with task graphs by considering precedence constraints and resource contention in the SCR model. The time complexity for calculating *lcts* is $\mathbf{O}(|N_t| + |N_s| + |E|)$. The time complexity for calculating *ecp* is $\mathbf{O}(|N_t| + |N_s| + |E|)$ at each step, making the time complexity of SECPF $\mathbf{O}((|N_t| + |N_s|)(|N_t| + |N_s| + |E|))$. Consequently, the complexity of the SECPF is reasonable, because in [1, 3–5] corresponding complexities are $\mathbf{O}(|E|(|N_t| + |E|))$, $\mathbf{O}(|N_t|(|N_t| + |E|))$, $\mathbf{O}((|N_t| + |E|)\log|N_t|)$ and $\mathbf{O}(|N_t|^2\log|N_t|)$.

4 Experimental results

We demonstrated the feasibility of the SECPF for the SCR model by evaluating 540 randomly generated IF1 [6] program graphs. IF1 is the proposed intermediate form of SISAL [7]. The IF1 program graph generated by the SISAL compiler initially is transformed into the intermediate form of SCR model by a preprocessor.

The size of the graphs varied from a minimum of 100 nodes with 102 edges to a maximum of 150 nodes with 300 edges. The granularity, that is, the ratio of average remote communication to average computation, varied from 0.5, 1 to 2. L_1 varied from 2, 4 to 8 cycles/byte. The ratio of L_r to L_1 varied from 2, 4, 6, 8 to 10. The communication volume associated with each communication edge varied randomly from 1 to 96 bytes. The computation time required for each computation node varied randomly from 1 to 96 cycles. The data sharing, which means the probability of consumers accessing the same data from the same predecessor, varied from 0%, 20%, 40%, 60%, 80% to 100%. The performance improvement is defined as

$$\frac{ecp(G) \text{ before partitioning} - ecp(G) \text{ after partitioning}}{ecp(G) \text{ before partitioning}} \times 100\%$$

Since DSC [3, 8] generally surpasses existing partitioning algorithms, such as Sarkar's [1], KB/L [4] and MCP [5], as shown in [3]; only DSC was implemented as a basis for comparison of performance improvement. DSC and SECPF were implemented for four program models to investigate the benefits of the SCR model. The first program model was the traditional macrodataflow; the second one was macrodataflow with consideration of resource contention as in the SCR model; the third was the memory node model; the last was the SCR model.

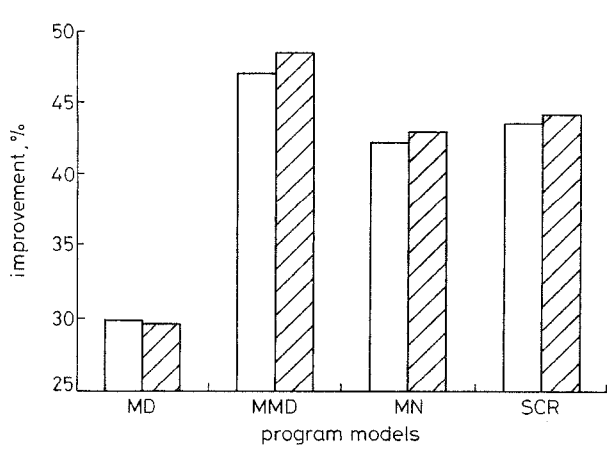

Fig. 3 *Average performance improvement*
Unshaded boxes: DSC
Shaded boxes: SECPF

Fig. 3 shows the average improvement for four models. 'MD' means the original macrodataflow, 'MMD' macrodataflow with considering resource contention, 'MN' memory node model and 'SCR' the SCR model. DSC performed better than the others in macrodataflow, but SECPF surpassed the others after adopting macrodataflow with resource contention consideration. Furthermore, the SECPF algorithm, as shown in Fig. 3, outdoes the others based on the SCR model. Adoption of the SCR model can achieve a more realistic outcome than using existing program models for the partitioning problem in NUMA multiprocessor systems. Fig. 3 also shows that performance improvement of both algorithms in MMD model is greater than that in SCR model. The main reason of the superiority could be that MMD model did not express data sharing such that the initial $ecp(G)$ before partitioning in MMD model is greater than that in SCR model. As for the performance improvement of algorithms in SCR compared with MN model, the reason of the superiority of SCR model appears that a task can start its execution as soon as precedence constraints and resource contentions affecting it have been resolved.

Only the performance of partitioning algorithms based on SCR model is discussed below, since outcomes using the SCR model are more realistic than the others. Fig. 4 shows the impact of granularity on the partitioning algorithms. In Fig. 4, the symbol 'X–Y' means that the value of the granularity is 'Y' for 'X' algorithm. The performance improvement increases as granularity increases. The same experimental consequence is also represented in [2]. The impact of data sharing is also shown in Fig. 4. The performance of DSC is better than that of SECPF when granularity is 0.5 and data sharing is less than 40%. The performance of DSC would be worse than that of SECPF if the data sharing is greater than 20%, when granularity is 1. If the granularity is greater than 2, then the performance improvement of SECPF will always be better than DSCs regardless of data sharing. Notice that the influence of the communication contention overhead on the performance increases as the granularity or data sharing increases. Because SECPF takes communication contention into account, SECPF is superior to DSC as the granularity or data sharing increases.

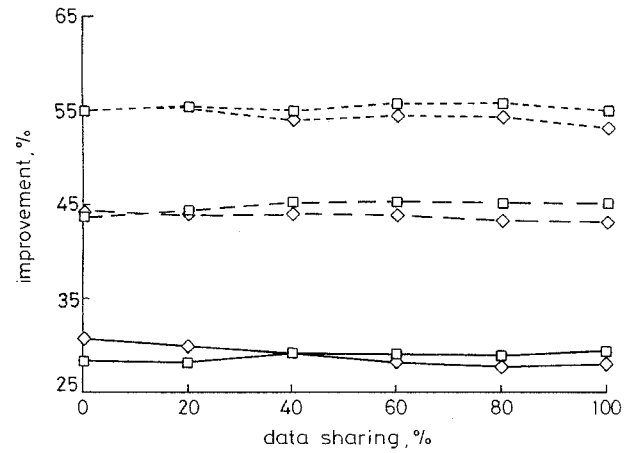

Fig. 4 *Improvement in varying data sharing and granularity*
—◇— DSC - 0.5
- -◇- - DSC - 1
···◇··· DSC - 1
—□— SECPF - 0.5
- -□- - SECPF - 1
···□··· SECPF - 2

5 Concluding remarks

An efficient program model for NUMA systems has been presented in this paper. The representation of the SCR model is simple, consistent and system-independent. It can reveal precedence constraints and resource contention. Users also can design more efficient partitioning algorithms using the SCR model. Moreover, the SCR model can express complex communication options, capture data sharing, and explicitly reveal the memory access contention relationships. A new partitioning algorithm also was developed here to investigate the workability of the SCR model. The SECPF algorithm based on the SCR model is superior to DSC, as demonstrated by our experimental results.

6 Acknowledgments

This work was supported by the R.O.C. National Science Council under the contract NSC85-2221-E009-038. The authors thank the referees for their useful comments.

7 References

1 SARKAR, V.: 'Partitioning and scheduling parallel programs for multiprocessors' (MIT Press, Cambridge, MA, 1989)
2 WOLSKI, R.M., and FEO, J.T.: 'Program partitioning for NUMA multiprocessor computer systems', *J. Parallel Distrib. Comput.*, 1993, **19**, pp. 203–218
3 GERASOULIS, A., and YANG, T.: 'A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors', *J. Parallel Distrib. Comput.*, 1992, **16**, pp. 276–291
4 KIM, S.J., and BROWNE, J.C.: 'A general approach to mapping of parallel computation upon multiprocessor architectures'. Proc. ICPP '88, 1988, pp. 1–8
5 WU, M.Y., and GAJSKI, D.: 'A programming aid for hypercube architectures', *J. Supercomput.*, 1988, **2**, pp. 349–372
6 SKEDZIELEWSKI, S., and GLAUERT, J.: 'IF1 – an intermediate form for applicative languages, version 1.0'. Technical report, M-170, Lawrence Livermore National Laboratory, CA, July 1985
7 MCGRAW, J., SKEDZIELEWSKI, S., ALLAN, S., GRIT, D., OLDEHOEFT, R., GLAUERT, J. DOBES, I., and HOHENSEE, P.: 'SISAL: stream and iteration in a single-assignment language, version 1.2'. Technical report, M-146, Lawrence Livermore National Laboratory, CA, 1 March 1985
8 GERASOULIS, A., and YANG, T.: 'On the granularity and clustering of directed acyclic task graphs', *IEEE Trans. Parallel Distrib. Syst.*, 1993, **4**, pp. 686–701
9 LAI, G.J., and CHEN, C.: 'A new scheduling strategy for NUMA multiprocessor systems'. Int. Conf. on *Parallel & Distributed Systems* (ICPADS'96), Tokyo, 1996, June, pp. 222–229