Fast generation of large task network mappings

Karl-Eduard Berger, Francois Galea, Bertrand Le Cun, Renaud Sirdey

Fast Generation Of Large Task Network Mappings

Karl-Eduard Berger*†, François Galea*, Bertrand Le Cun†, Renaud Sirdey*
*CEA, LIST
Embedded real time systems laboratory
Centre de Saclay, PC 172, 91191 Gif-sur-Yvette Cedex, France
†University of Versailles Saint-Quentin-en-Yvelines
PR*i*SM laboratory
55 avenue des Etats-Unis, 78035 Versailles Cedex, France
Email: karl-eduard.berger@cea.fr, francois.galea@cea.fr, bertrand.lecun@prims.uvsq.fr, renaud.sirdey@cea.fr

***Abstract*—In the context of networks of massively parallel execution models, optimizing the locality if inter-process communication is a major performance issue. We propose two heuristics to solve a dataflow process network mapping problem, where a network of communicating tasks is placed into a set of processors with limited resource capacities, while minimizing the overall communication bandwidth between processors. Those approaches are designed to tackle instances of over one hundred thousand tasks in acceptable time.**

***Index Terms*—heuristics, process network mapping, manycore execution optimization**

I. Introduction

With the end of the frequency version of Moore's law, a new generation of massively multi-core microprocessors is emerging. This has triggered a regain of interest for the so-called dataflow programming models in which one expresses computation-intensive applications as networks of concurrent processes (also called agents or actors) interacting through (and only through) unidirectional FIFO channels. See e.g. [7] for a recent instantiation of this model.

On top of more traditional compilation aspects, compiling a dataflow program in order to achieve a high level of dependability and performance on such complex processor architectures involves solving a number of difficult, large-size discrete optimization problems amongst which graph partitioning, quadratic assignment and (constrained) multi-flow problems are worth mentioning [14].

In this paper, we focus on the problem of mapping a dataflow process network (DPN) on a clusterized parallel microprocessor architecture composed of a number of nodes, each of these node being a small SMP, interconnected by an asynchronous packet network.

A DPN is modeled by a graph where the vertices are tasks to place, and the edges represents communication channels between tasks. Vertices are weighted with one or more quantities which correspond to processor resources consumption and the edges are weighted with an inter-task communication outflow. The aim of our problem is to maximize inter-task communications inside SMPs while minimizing inter-node communication under capacity constraints to be respected in terms of task resource occupation on the SMPs.

We present in this paper two methods able to tackle large instances of that problem in a reasonable amount of time. The rest of the paper is organized as follows. Sect. II formally states the DPN mapping problem and locates our work in the literature. Sect. III and Sect. IV portray the two methods we developped. Sect. V presents the results of our approach and Sect. VI allows us to conclude and provide some ideas for the evolution of our methods.

II. The DPN mapping problem

A. Problem statement

Let T denote the set of tasks in the DPN and N the set of nodes. Let R denote the set of resources offered by the nodes (e.g., memory capacity, processing capability). Also, let w_{tr} denotes the consumption of tasks t in resource r, $q_{tt'}$ denote the bandwidth between tasks $t \neq t'$ and $d_{nn'}$ denote the routing distance between nodes $n \neq n'$. Also, for simplicity sake and with a slight loss of generality, we assume that all nodes are identical and we denote by C_r the capacity of any of the nodes for resource r.

Given the variables

$$x_{tn} = \begin{cases} 1 & \text{iff task } t \text{ is assigned to node } n, \\ 0 & \text{otherwise,} \end{cases}$$

our DPN placing problem can then be expressed as the following mathematical program:

$$\begin{cases} \text{Minimize } \displaystyle\sum_{t\in T}\sum_{t'\neq t}\sum_{n\in N}\sum_{n\neq n'} x_{tn}x_{t'n'}q_{tt'}d_{nn'}, \\ \text{s. t.} \\ \displaystyle\sum_{n\in N} x_{tn} = 1 \ \forall t \in T, & (1) \\ \displaystyle\sum_{t\in T} w_{tr}x_{tn} \leq C_r \ \forall n \in N, r \in R, & (2) \\ x_{tn} \in \{0,1\} \ \forall t \in T, n \in N. \end{cases}$$

Constraints of type (1) simply express that each task must be assigned to one and only one node and constraints of type (2) requires that the node capacity is not exceeded.

This generalized quadratic assignment problem is straightforwardly NP -hard in the strong sense notably by restriction to the Node Capacitated Graph Partitioning Problem [5] (arbitrary network topology and bandwidths as well as equidistant nodes), to the Quadratic Assignment Problem (in the case

where the capacity constraints allow to assign one and only one task per node and where the internode distance is arbitrary) as well as to the bin-packing problem. This list emphasizes the numerous sources of difficulties when tackling this problem. In our work, we will only work with one resource.

In terms of instance size, in our application context, we have to be able to map networks of over a hundred thousand of tasks on architectures having several hundreds of nodes. Such an order of magnitude rules out exact resolutions methods: the best known methods for the node capacitated graph partitioning problems are limited to graphs with a few hundreds of vertices, and the best known algorithms for the QAP are limited to instances of size around 30 [10]. Therefore, in our specific context, heuristic approaches are required to provide results for our problem on large graphs in reasonable time.

B. Related works

Partitioning with graphs which contain several hundreds of thousands of vertices is a well studied problem. The Path Optimization parallel algorithm [2] is a variation of the hill climbing algorithm and is able to find a good partitioning of several tens of thousands of tasks in a reasonable amount of time. However, it does not place the partitionned tasks.

Parallel solvers are able to solve large instances which contain millions of vertices [8], [11]. Those approaches deal with load balancing constraints and don't consider resource capacity constraints.

Previous works established two solvers for the DPN mapping problem in the context of building a cyclo-static dataflow compilation toolchain for parallel microprocessor architectures targeting the embedded market [1].

A first method is based on progressive construction for the multi-resource Node Capacited Graph Paritioning Problem ([14], [12]). The method consists of two phases: a partitioning phase and a placement phase. The partitioning phase is a GRASP approach [4]. It is an affinity-based randomized iterative process which creates a satisfying partitioning of the tasks graph. As this process may make unfortunate choices during its execution, it is run several times using different randomization parameters in a multi-start process. Only the best solution is kept. The second part of the algorithm consists in a simulated-annealing-based quadratic assignment problem (QAP) heuristic which assigns one partition of the task graph to each of the SMPs. This first algorithm is fast and used at the early development cycle. It also uses a fusion principle. A task to node affinity function and an node to node affinity function are used in the process. If the affinity of the best node to node affinity is higher than the best vertex to node affinity and if there is enough space in one of the nodes, then a fusion occurs. Otherwise, the task is placed in the candidate node.

A second approach [6] is based on a parallel simulated annealing. This is a single-phase heuristic which directly assigns tasks to the SMPs. It provides better results but it takes considerably more time than the previous method even though parallelism allows to drastically reduce the execution time.

The two methods are quite efficient while dealing with a reasonable number of tasks. However, those methods do not scale with instances which contains more than a hundred thousand of tasks. This is why we propose two methods which are able to deal with those instances.

Those methods are based on graph spanning. This is due to the fact that tasks graphs are not random graphs but show a pecular structure that we want to exploit.

III. Greedy task wise placement method

A. Affinities

In order to identify a good task assignment, we used a metric [3] which is defined as an affinity between two subsets. A subset consists in a task, a node or a group of tasks.

Let T_1, T_2 two subsets of the set of tasks T of the DPN. The affinities $\alpha_{T_1T_2}$ of T_1 to T_2 is:

$$\alpha_{T_1T_2} = \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} q_{t_1 t_2}.$$

B. Distance affinities

When iteratively choosing on which node a task must be placed, an intuitive way is to place it as close as possible from its neighbour tasks which are already placed. For this, we introduce the notion of distance affinity. For any DPN mapping solution, the distance affinity β_{tn} between a task t and a node n is the following:

$$\beta_{tn} = \sum_{n'} \sum_{t'} x_{t'n'} x_{tn} q_{tt'} \times \frac{1}{2 \times d_{nn'} + 1}$$

Locally maximizing this function when chosing a node to assign a task to, intuitively tends to minimize the DPN mapping objective function.

C. Description of the algorithm

In this method, all tasks are assigned one after another using the distance affinities computation properties. It is a one phase placement.

Initially, the task, which has got the sum of the weights of adjacence edges maximum, is placed in the first avalaible node. All its unassigned neighbors are pushed into a FIFO queue and their corresponding distance affinities towards the selected node are computed.

Next, The task with the highest task to node affinity are selected and removed from the queue. If two or more tasks have the same affinity, the priority corresponds to the FIFO order. The selected task is then placed in the node with whom it has the greatest affinity. As long as there are no saturated node, this strategy is applied.

Then, sooner or later, some nodes becomes saturated. All tasks whose greatest affinity corresponds to a saturated node are removed from the queue. All unsaturated nodes which are in the neighborhood of the saturated node are selected. A pre-generated ordering of all tasks, generated by breadth first traversal, is used to choose as many unassigned tasks as we selected nodes. Basically each of the first unassigned tasks in the ordering are assigned to a different selected node.

The unassigned neighbors of those tasks are placed in the queue and their respective affinities are updated.

This process repeats as long as there are unassigned tasks. The method is shown in Algorithm 1.

Algorithm 1 Greedy Task Wise Placement

```
/*
 Q : a FIFO queue which contains the neighborhood of V
 Q_BFT : A queue which contains the result of
         the BFT algorithm
 T2NA : A |V|x|N| matrix which
                          contains all tasks to
                          nodes affinities.
 v : candidate vertex
 n : candidate node*/

 v = vertexWithHighestSumOfWeightsOfAdjacencyEdges()
 Q = enQueue(neighbors of v)
 assignToNode(v,0, assignment)
 updateAffinities(T2NA, neighbor of v)
 Q_BFT = BFT(v,|V|)
 while(unassigned tasks left)
   (v,n) = deQueueTaskwithHighestAffinity(Q)
   if(n is not saturated)
     enQueue(neighbors of v,Q)
     assignToNode(v,n)
     updateAffinities(T2NA, neighbor of v)
   else
     removeAllTasksFromQ
         WithHighestAffinityToSaturatedNode(T2NA,n)
     while(i < numberOfAvailableNode(neigbors of n))
       n = neighbor(V,i)
       v = dequeue(Q_BFT)
       Q = enQueue(neighbors of v)
       assignToNode(v,n)
       updateAffinities(T2NA,neighbors of v)
       i++;
     end while
   end if
 end while
```

IV. Subgraphs placement method

This second approach is a two phase method. Instead of assigning tasks one by one like the previous method, we generate a subgraph of connected tasks which are then placed on a node. The connected subgraph obtained is assigned to a node depending on the affinity between the subgraph and the nodes.

A. Description of the algorithm

The task with the lower number of neighbors is selected. This task is used as the starting task. We defined a size for the subgraph we want to generate.

Subgraphs are generated by performing a breadth first traversal of the unassigned tasks, starting from a promising task, until a certain total size (int terms of resouce occupation) is reached. Empiric experimentation showed that choosing a size of maximal remaining capacities for all clusters multiplied by a factor of $\frac{1}{2}$ gave the best results.

The breadth first strategy for graph traversal selects the closest tasks from the initial task.

Once the subgraph has been built, the affinities between the subgraph and all nodes are computed using the relation cited in III-A. The subgraph is assigned to the node which has the strongest affinity and enough space.

Name	# tasks	# nodes	node capacity
grid12x12	144	4	40
grid23x23	529	16	40
grid46x46	2,116	64	40
grid100x100	10,000	256	40

TABLE I
GRID SHAPED TASK TOPOLOGIES

This process repeats until all tasks are assigned. Algorithm 2 presents the algorithm in pseudo code.

Algorithm 2 Subgraph Placement method

```
/*
 remCap(n) : remaining capacity for node n
 G : The task graph
 V : A subgraph of tasks
 C : maximal remaining capacity
 StNA : A |V|x|N| matrix which contains
                    all subgraph to nodes affinities.
 v : candidate vertex
 n : candidate node
 A : affinity*/

 v = TaskWithTheLowestDegree()
 assignToNode(v,0)
 while(Unassigned tasks left)
   C = max remCap for all nodes.
   V = BFT(G,C)
   (A,n) = computeSubgraphToNodeAffinity(StNA, V)
   assignToNode(V,n)
 end while
```

V. Experimental results

A. Execution platform

The target system is a P.C based on the Intel Xeon E5/Core i7 processor running at 2.0 GHz. As our algorithms are purely sequential, only one CPU core is used.

B. Instances

Two kinds of task graph topologies were used. First, a set of grid shaped task topologies, which correspond to dataflow computational networks like matrix products. (Table I). The other kind of task graph is generated out of logic gate networks resulting in the design of microprocessors. These configurations of task networks typically can be found in real life complex dataflow applications (Table II).

The node layout is a square torus, hence the number of nodes in all instances in a square value.

For each pair (t, t') of tasks of the grid, the bandwidth $q_{tt'}$ is set to 1 if tasks t and t' are adjacent in the task grid, and 0 otherwise. For graphs generated out of logic gate networks, the edge weights are the number of arcs between the corresponding elements in the original multigraph.

For each pair of nodes (n, n'), the distance $d_{nn'}$ is the Manhattan distance between nodes n and n'.

In those experimentations, all instances are limited to one resource and the resource occupation of every tasks in arbitrarily set to 1.

Name	# tasks	# nodes	node capacity
b12	1,065	36	40
b17	24,171	256	100
b18	114,561	400	300
b19	231,266	576	410

TABLE II
LOGIC GATE NETWORK TOPOLOGIES

Name	# tasks	P&P		GTWP	
		Sol. Val.	run time	Sol. Val.	run time
grid12x12	144	37	0.02 s	41	2.3 ms
grid23x23	529	220	0.05 s	338	5.2 ms
grid46x46	2,116	2,500	2 s	2,306	0.17 s
grid100x100	10,000	45,613	240s	16,000	3 s

TABLE III
P&P. AND GTWP APPROACH WITH GRID SHAPED TASK TOPOLOGIES

Name	P&P		GTWP	
	Sol. Val.	run time	Sol. Val.	run time
b12	1,200	4.85 s	1,598	9.6 ms
b17	135,000	3,100 s	109,879	88 s
b18	832,538	40 h 18 min	395,624	2163 s
b19	-	-	-	-

TABLE IV
P&P AND GTWP APPROACH WITH LOGIC GATE NETWORK TOPOLOGIES

Name	P&P		Subgraph	
	Sol. Val.	run time	Sol. Val.	run time
grid12x12	37	0 .02 μs	75	382 μs
grid23x23	320	0.05 s	338	2.46 ms
grid46x46	2,500	2 s	2,565	13 ms
grid100x100	45,613	240 s	18,790	0.33 s

TABLE V
P&P AND SUBGRAPH APPROACH WITH GRID SHAPED TASK TOPOLOGIES

Name	P&P		Subgraph	
	Sol. Val.	run time	Sol. Val.	run time
b12	1,188	4.85s	2,205	48.46 ms
b17	135,000	3,100s	155,396	3.89 s
b18	832,538	40 h 18 min	1,936,952	40.55 s
b19	-	-	5,613,634	413.56 s

TABLE VI
P&P AND SUBGRAPH APPROACH WITH LOGIC GATE NETWORK TOPOLOGIES

Name	GTWP		Subgraph	
	Sol. Val.	run time	Sol. Val.	run time
grid12x12	41	2.3 ms	75	382 μs
grid23x23	338	5.2 ms	338	2.46 ms
grid46x46	2,306	0.17 s	2,565	13 ms
grid100x100	16,000	3 s	18,790	0.33 s
b12	1,598	9.6 ms	2,205	48.46 ms
b17	109,879	88 s	155,396	3.89s
b18	395,624	2,163 s	1,936,952	40.55 s
b19	-	-	5,613,634	413.56 s

TABLE VII
GTWP AND SUBGRAPH APPROACH WITH GRID SHAPED TASK TOPOLOGIES AND LOGIC GATE NETWORK TOPOLOGIES

C. Computationnal results

We compare our methods with that of [14]. We denote this method as Partition and Place (P&P).

All tables display the solution objective value and the execution time of the methods we cited. The tables display either the application of the algorithms on grids shaped task topologies (Table III, IV) or on logic gate network topologies (Table V, VI). Table VII shows the result on either the grid shaped task topologies or the logic gate network topologies.

On Table III we can observe that for small instances, the P&P algorithm provides better results than the GTWP algorithm whereas that the last one is faster. However when the number of tasks is higher than 2000, the GTWP begins to provide better results and far better run times.

On Table IV, the GTWP algorithm seems to have the same behaviour than cited above. The higher the number of tasks, the more the solution quality of the GTWP method increases compared to the solution quality of the P&P algorithm, with a relative speedup of 67 for the b18 instance. One fact that calls our attention is the fact that the GTWP algorithm works fine either on logic gate network topologies or grids shaped task topologies. However, the algorithm can not give any results in a reasonable amout of time for instances of 200000 tasks.

Table V and Table VI shows the result of the Subgraph algorithm. The runtimes are several orders of magnitude faster than the P&P approach, while providing solutions whose quality tends to get comparatively similar or better on the largest instances.

The difference between our methods are illustrated in Table VII which contains the grid shaped task topologies and the logic gate network topologies. The GTWP method provides better results than the subgraph method, while the subgraph method runs faster than the GTWP method and scales easily on very large instances.

The increase in terms of compared solution quality between our methods and the P&P algorithm finds its explanation in two different aspects. First, as the partitioning phase of P&P does not take node distance into account, tasks are gathered together with no knowledge of the destination processor topology. Thus, choices made during this phase may undermine the overall solution quality. In the opposite, the distance affinity notion we use in the GTWP approach allows us to take profit of the topology and avoid many bad choices. Second, even not taking profit from the node distances, the subgraph placement method has the advantage that it tries to avoid placing singletons or very small subgraphs, while the last 10% (or perhaps more) of the tasks to be assigned in P&P may probably not be efficiently assigned, leading to a drop in quality.

VI. CONCLUSION

The goal of this study was to evaluate new heuristic methods to tackle large instances of the DPN mapping problem which emerge from the cyclo-static dataflow parallel programming paradigm. Being able to provide good placements is crucial

for execution performance of large-sized dataflow programs on massively parallel architectures which are currently emerging.

We presented two heuristic methods based on progressive construction, and compared the execution results with those obtained from a solver coming from a previous work and currently used in a cyclo-static dataflow programming toolchain. Both methods run much faster than the one to which it is compared to, by several orders of magnitude. Both methods show good scalability, as they provide relatively better solutions on large instances.

We are conscious that the solutions we obtained are of relatively low quality. A study seems necessary to develop an optimization method, possibly taking profit from locality for parallel acceleration, using the methods we proposed for the generation of initial solutions.

REFERENCES

[1] Aubry et al. (2013): "Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor", Procedia Computer Science 18 pp. 1624-1633.

[2] J. W. Berry, M. K. Goldberg (1999): "Path Optimization for Graph Partitioning Problems", Discrete Applied Mathematics, pp 27 - 50.

[3] V.David, C.Fraboul, J.-Y. Rousselot and P.Siron (1991): "Etude et ralisation d'une architecture modulaire et reconfigurable: projet MODULOR", Rapport technique 1/3364/DERI, Onera

[4] T.A. Feo and M.G.C. Resende (1995): "Greedy Randomized Adaptive Search Procedures", Journal of Global Optimization, pp. 10913.

[5] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel and L. A. Wolsey (1998): "The Node Capacitated Graph Partitioning Problem: A Computational Study", Mathematical Programming 81, pp. 229-256.

[6] F.Galea, R.Sirdey (2012): "A Parallel Simulated Annealing Approach for the Mapping of Large Process Networks", IPDPS, pp.1787-1792.

[7] T. Goubier, R. Sirdey, S. Louise and V. David (2011): "ΣC: A Programming Model and Langage for Embedded Many-Cores", LNCS 7016, pp. 385-394.

[8] G.Karypis and V.Kumar (1998): "Multilevel Algorithm for Multi-Constraint Graph Partitioning", Technical Report, pp. 98-019.

[9] D.E Knuth (1997): "The Art of Computer Programming 3rd Edition", Boston: Addison-Wesley, ISBN 0-201-89683-4

[10] E. M. Loiola, N. M. N. de Abreu, P. O. Boaventura-Netto, P. Hahn and T. Querido (2007): "A Survey for the Quadratic Assignment Problem", European Journal of Operationnal Research 176, pp. 657-690.

[11] F.Pellegrini (2010): "Contribution to Parallel Multilevel Graph Partioning", HDR Thesis, Université de Bordeaux 1.

[12] O.Stan, R.Sirdey, J.Carlier, D.Nace (2012): "A Aeuristic Algorithm for Stochastic Partitioning of Large Process Network", Proceedings of the 16th IEEE International Conference on System Theory, Control and Computing.

[13] R.Sirdey, J. Carlier and D. Nace (2010): "A GRASP for a Resource-Constrained Sheduling Problem", International Journal of Innovative Computing And Applications, pp. 143-149.

[14] R. Sirdey (2011): "Contributions à l'optimisation combinatoire pour l'embarqué : des autocommutateurs cellulaires aux microprocesseurs massivement parallèles", HDR Thesis, Université de Technologie de Compiègne.