A Scheduling Framework for Heterogeneous Multiprocessor Architectures Based on Industrial Processors (DSPs and Microcontrollers)

Adriano Tavares
Department of Industrial Electronics
University of Minho
4800 Guimarães
PORTUGAL
atavares@deiuminho.pt

Carlos Couto
Department of Industrial Electronics
University of Minho
4800 Guimarães
PORTUGAL
ccouto@dei.uminho.pt

***Abstract* – Current VLSI and networking technology, the increase in computational power, and the rapid decrease in computational cost, enable the interconnection of VLSI processors, which can be arranged on a functional decomposition of the computational task to exploit the potential of multiprocessing. The use of multiprocessor systems in such way, provides a novel and cost effective solution in solving many practical problems in signal processing, control systems, instrumentation systems and robotics. In this article we present a framework that addresses the specificities of industrial processors, such as DSPs and microcontrollers and can easily be used to implement a huge range of scheduling algorithms.**

I. INTRODUCTION

Many computational problems in control, signal processing, robotics and instrumentation frequently make a huge requirements of interfaces and processing power that can not be satisfied by a single processor system, making a good case for multiprocessor architectures. However, the potential advantages of multiprocessor systems can be exploited only if the problem to be solved can be partitioned into small tasks and these tasks scheduled onto the processors of a multiprocessor systems in a way that minimizes the completion time. This problem becomes harder since with those systems mentioned above, skills are necessary to deal with additional degree of complexity that emerges from different contributions of the different processors that compose a heterogeneous multiprocessor architecture.

We consider a heterogeneous multiprocessor system to be the one that makes use of several different types of processors, and/or connectivity topologies to optimise performance and/or cost-effectiveness of the system. For instance, different processor types could include DSPs, microcontrollers, PICs, and others special purpose processors. Similarly, different connectivity paradigms could include bus, point-to-point, ring, or a mixture of these. Examples of heterogeneous systems can be found in:

1) Embedded Systems: Application specific systems containing hardware and/or software tailored for a particular task. General-purpose processors, programmable digital signal processors, and ASICs are among the many components of these systems. In this kind of systems, heterogeneous processors are tightly coupled with low IPC (InterProcessor Communication) overhead but with heavy resource constraints.

2) Distributed Systems: Consisting of various workstation, main computers, and even super computers, with significant overhead resulting from the IPC.

3) SHHiPE (Scalable Heterogeneous High Performance Embedded): An embedded system developed by integrating COTS (Commercial Off The Shelf) processors and interconnect components in a 'plug-and-play' fashion. These systems offer advantages of low-cost, scalability, easy programmability, software portability, and ability to incorporate evolving hardware technology [1, 2].

II. MULTIPROCESSOR SCHEDULING PROBLEM

The objective of scheduling is the minimization of the total execution time and its output is a Gantt chart, which contains information about the distribution of the tasks included in a given program graph onto the various target machine's processors. This problem is NP-hard, i.e., the computational requirements for an optimal solution increase exponentially with the number of tasks and number of processors, which makes the finding of the optimal schedule, even for moderately sized problems, not practicable. Therefore, nearly all practical scheduling algorithms incorporate heuristics. Since scheduling is expressed as an optimisation problem, we try out some approaches based on:

1) Enumerate methods: Search every point related to an objective function's domain space, one point at a time. They are very simple to implement but may require significant computation. The domain space of many applications is too large to search using these techniques.

2) Calculus Based Techniques: Use a set of necessary and sufficient conditions to be satisfied by the solutions of an optimisation problem. It can be divided into two groups, direct and indirect methods. Direct methods are computationally feasible, but can be use only on a restricted set of "well-behaved" problems. Indirect methods look for local extrema by solving a nonlinear set of equations, resulting from zeroing the gradient, and so, is computationally heavy.

3) Heuristic Methods: Solve the NP-hard problems within reasonable time, but never guarantee an optimal solution since they only search part of the entire solution space and suffer from the so called horizon effect[1].

4) Guided Random Search Techniques: Including both simulated annealing and genetic algorithms are based on enumerative techniques but use additional information to guide the search.

Scheduling can be dynamic or static. The static scheduling approach can be applied to only a subset of

[1] This property asserts that no matter how far one looks ahead before making a local decision, there may exist something "just over the horizon" which can render the decision detrimental to the objective, i.e., heuristic methods tend to take optimal local decisions that may have a non-optimal global effect.

problems whose run-time behaviour is predictable, i.e., the precedence-constrained task graph must be known beforehand. Therefore, constructs such as branches and loops must be excluded to provide deterministic program behaviour. The disadvantage of dynamic scheduling is its inadequacy in finding global optimums and its additional execution overhead resulting from the fact that the schedule must be determined while the program executes.

All schedulers we have been implement, are statics and due to the nature of the processors in use (microcontrollers and DSPs), they make explicit consideration about interprocessor communications, memory and I/O resources. Therefore, some modelling technique must be used to express IPC, memory, and I/O resources.

A. Memory Constraints

As said before, an edge between two tasks allocated to different processors represents an interprocessor communication, and so, demands memory storage at the destination processor. Using the memory usage information associated with the schedule state $\sum(t)$ at instant t, every solution with memory requirement that violate the memory availability must be discarded. Note that $\sum(t)$ describes the state of the processing, memory, I/O and communication resource at instant t. To schedule a given task N_i on processor P_j at instant t, the memory requirement evaluation is carried out, following the next steps:

1) Deallocation Stage: The schedule state $\sum(t)$ is analysed and input memory requirement is deallocated for all tasks that end execution before or at the instant t,

2) Transfer Stage: For all predecessors of N_i scheduled before on processors P_k, and for all $k \neq j$, the associated output memory requirement to N_i must be transferred to P_j,

3) Output Allocation Stage: For all N_i's successors tasks, memory storage must be allocated on processor P_j,

4) Node Acceptance: The schedule of N_i onto P_j at instant t is accepted only if P_j has enough memory to satisfy the demands at the transfer and output allocation stages.

However, special care must be taken, since some implemented schedulers perform schedule-hole exploitation. Depend on the instant t and the time the last node assigned to the j^{th} processor finishes execution, $TF(N_i,P_j,\sum)$, one of the two cases have to be handled:

1) If $t \geq TF(N_i,P_j,\sum)$: Then the memory usage is evaluated only for N_i,

2) Otherwise: The evaluation must be repeated for all tasks scheduled so far in a way of increase starting time order.

Note that not only the data communication must be take part of the memory usage evaluation, but also the memory for global and static variables, and the program memory. The memory program of each processor is updated accumulatively after each task allocation, and so, before the schedule of a given Task N_i on processor P_j at instant t, the schedule state $\sum(t)$ must be used to verify that P_j has enough program memory for the storage of N_i's code.

B. Interprocessor Communication Modeling

The IPC modelling requires an estimator for the time it takes to communicate a number of data units between two processors, and a model of the interconnection topology. If the communication time is not the same for all interconnections, then an estimator has to be included for each communication channel. The communication model reflects the way the processors communicate, which again depends on the interconnection topology. Different communication times and ways the processor can communicate arise from different topologies and topologies combinations. Topologies such as shared memory (single and multiple bus), point-to-point connections, mesh, and a mixed of shared memory and point-to-point connection are handled by our schedulers. Note that, the availability of some kind of processor I/O is very important, in a way that it makes a processor able to calculate and communicate simultaneously.

In order to provide a unified vision of the interconnection topology to the scheduler, the following considerations are introduced:

1) Shared Memory Modelling: A shared memory is modelled as a "dead processor", i.e., a processor without execution capability, and processor I/O, but with storage capacity

2) Shared Memory Connection: A connection between two processors through a shared memory can be represented as two point-to-point connections,

3) Shared Memory Access: The number of simultaneous accesses to the shared memory depend on the number of ports and the addresses to be accessed.

Using such considerations, the scheduler has a unified vision of the interconnection topologies as based on point-to-point connections. Therefore, the communication cost between two processors P_i and P_j (including the 'dead processor') can be modelled as

$$CommCost_{i,j} = SetupTime + NUnits*WCTime, \quad (1)$$

where NUnits and WCTime, specify the number of data units to be sent and the communication time for one data unit, respectively. In cases where one of the communicating processor is a "dead processor", the SetupTime is the overhead incurred in setting the connection from a processor to the shared memory, i.e., the overhead associated with taking the shared memory from a processor and giving it to another one, and WCTime is equal to the number of wait states plus one.

C. I/O Resource Modelling

The allocation of the I/O channel to the task's variables must be efficiently realized, to avoid a premature emptiness of resources. For instance, a properly resource allocation must guarantee that at the end, only those channels with worst read/write time will be available, while a premature emptiness occurs when there is no free channel that satisfy a given variable profile, due to an early bad allocation. Premature emptiness is closely related to channels that present hybrid profile such as, bidirectional and those that can be used as digital or analog. To guarantee a properly I/O resource allocation, the following scheme is used:

1) Allocation Rule: A priority allocation rule for the processor I/O channels is defined,

a) the allocation must start with analog external variables. A variable that updates an output data channel or is updated by reading an input data channel is defined as external,

b) among all I/O channels available for a specific allocation, chooses the one that presents the best read/write time.

c) after the analog allocation, it will realize the digital one, starting with integer or float variable and then with bit variable. For each variable, first is selected a bidirectional channel if any is free, otherwise, an unidirectional channel. At this stage, the selection of the channel owner of the communication line (for instance, the serial TXD and RXD) must be delayed as much as possible.

2) Premature Emptiness Occurrence: If during the I/O allocation process occurs a premature emptiness, then verify and modify if possible the allocations of the hybrid channels. However, if the premature emptiness outlasts after all possible modification, then it can be concluded that the processor state, $\sum(t)$, has not enough resources for that task.

D. Scheduling Algorithms

Several completely new schedulers were implemented, based on genetic algorithm [3], simulated annealing [4], threshold accepting [5], great deluge [6], and some are improved and modified to incorporate the management of memory, I/O, and communication resources. Now we will present the improvement made to Lee's dynamic-level scheduler [7], as among all implemented schedulers, it is the one with best scheduling result/execution time tradeoff.

Ram Murthy suggested in [8] the exploitation of schedule-holes to improve the Lee's dynamic level scheduling for homogeneous system. We made a soft change in the Lee's dynamic level calculation for heterogeneous system in order to incorporate the Ram Murthy's suggestion and resource management purposed. The new dynamic level is given by

$$newDL_1(N_i,P_j,\sum(t))=SL^*(N_i) - FAH(N_i,P_i,\sum(t)) + \Delta(N_i,P_j) \quad (2)$$

$$oldDL_1(N_i,P_j,\sum(t)) = SL^*(N_i) - \max[DA(N_i,P_i,\sum(t)), TF(N_i,P_j,\sum(t))] + \Delta(N_i,P_j), \quad (3)$$

$$\Delta(N_i,P_j) = E^*(N_i) - E(N_i,P_j), \quad (4)$$

where $DA(N_i,P_i,\sum)$, $FAH(N_i,P_j,\sum)$, $SL^*(N_i)$, $E(N_i,P_j)$, $E^*(N_i)$, stand for: the earliest time all data required by node N_i are available at Processor P_i with memory and I/O requirement satisfied, the first available schedule hole after $DA(N_i,P_j,\sum)$, static level of node N_i, execution time of node N_i over processor P_j, and the adjusted median execution time of node N_i over all processors, respectively. The static level of a node N_i, $SL^*(N_i)$, represents the largest sum of execution times along any directed path from N_i to an end node of the task graph, over all end nodes of the task graph. All others Lee's scheduler parameters, such as descendent consideration and resource scarcity are kept unchanged.

III. EXPERIMENTAL RESULT

Now we will exemplify the use of the described scheduling framework through the "Fig.1" to "Fig.4" that were produced using *μVisualProg*. The upper side of "Fig.1" presents the Gantt chart with the information about tasks distribution onto target machine (given by lower left quadrant of "Fig.2"). The program graph is represented by the upper quadrants of "Fig.2", "Fig.3" and "Fig.4". In the lower side of "Fig.1" is presented the task graph that was automatically generated by the scheduler. "Fig.5" to "Fig.7" show more schedule results obtained using the suggested framework. The Gantt charts are obtained by scheduling the task graph (lower right quadant) onto the machine graph (left side).

IV. CONCLUSIONS

A scheduling framework that explores several heterogeneity's levels was described. This scheduling framework is part of an automatic programming tool for heterogeneous multiprocessor systems named ***μVisualProg*** developed at the Department of Industrial Electronics, University of Minho. Representing the problem as a distributed Grafcet and a multiprocessor system as a graph machine, *μVisualProg* starts to generate a precedence-constrained task graph that represents the parallel program and for each task determines the execution time onto each processor node of the graph machine. After the evaluation of these data, they are used to initialize the scheduler framework with informations about the multiprocessor architecture and task graph, and a scheduler algorithm is selected. A Gantt chart will be generated and used as the input of a code generation process.

Several scheduling algorithms were implemented and tested, and Lee's dynamic-level scheduler revealed as the one with best scheduling result/execution time tradeoff. However, for some very specific problems (with a specific task graph pattern) the schedule produced by this method were not good enough, and so, we recommend the use of other schedulers, such as, those ones based on tabu or simulated annealing. Note that, these last schedulers are very slow, and so, we will focus now on strategies for accelerating their convergence.

V. REFERENCES

[l] Wenheng Liu, "Communication Issues in Heterogeneous Embedded System," in *http://www.usc.edu/dept/ceng/Prasanna/projects/embed.html.*

[2] Prashanth Bhat, "Issues in Using Heterogeneous HPC Systems for Embedded Real Time Signal processing Applications," in *http://www.usc.edu/dept/ceng/Prasanna/projects/embed.html.*

[3] J. Ribeiro Filho, and C. Alippi, "Genetic-Algorithm Programming Environments," *IEEE computer Magazine*, June. 1994, pp. 28-43.

[4] R. W. Eglese, "Simulated Annealing: A Tool for Operational Research," *European Journal of Operational Research*, 46, 1990, pp. 271-281.

[5] G. Dueck e T. Scheuer, "Threshold Accepting: A General Purpose Optimisation Algorithm Appearing Superior to Simulated Annealing," *Journal of Computational Physics*, 90, 1990, pp. 161-175.

[6] Gunter Dueck, "New Optimisation Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel," *Journal of Computational Physics*, 104, 1993, pp. 86-92.

[7] G. Sih, and E. A. Lee, "A Compile-Time Scheduling Heuristic for InterConnection Constrained Heterogeneous Processor Architectures," *IEEE Transaction on Parallel and Distributed Systems*, vol. 4, no. 2, Feb. 1993, pp. 175-187.

[8] S. Selvakumar and C. Siva Ram Murthy, "Scheduling Precedence Constrained Task Graph with Non-Negligible Intertask Communication onto Multiprocessors," *IEEE Transaction on Parallel and Distributed Systems*, vol. 5, no. 3, March. 1994.

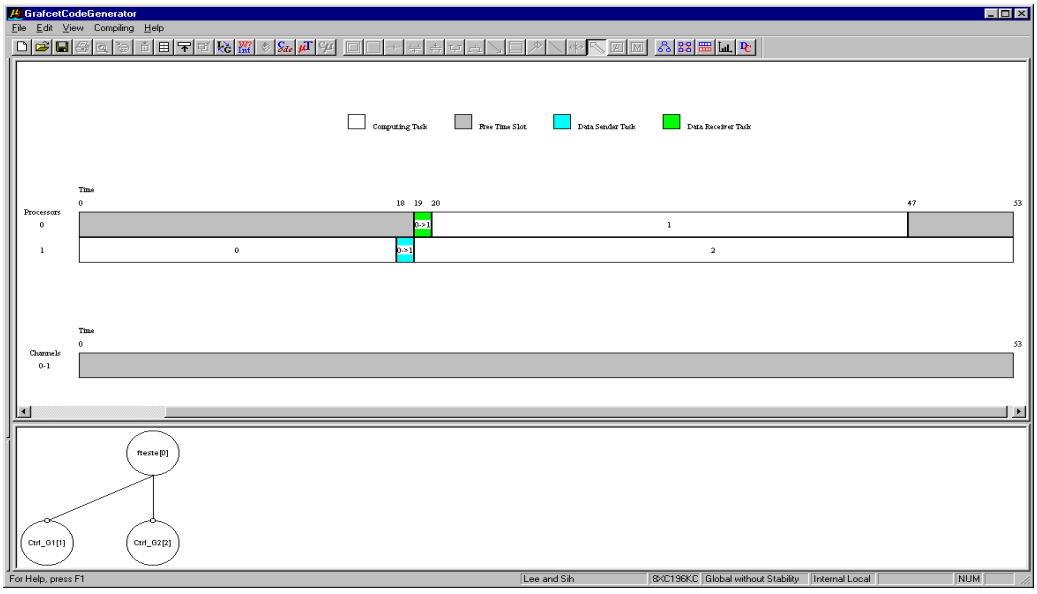

Fig.1 Gant chart and the task graph for the application modelled in Fig.2 to Fig. 4

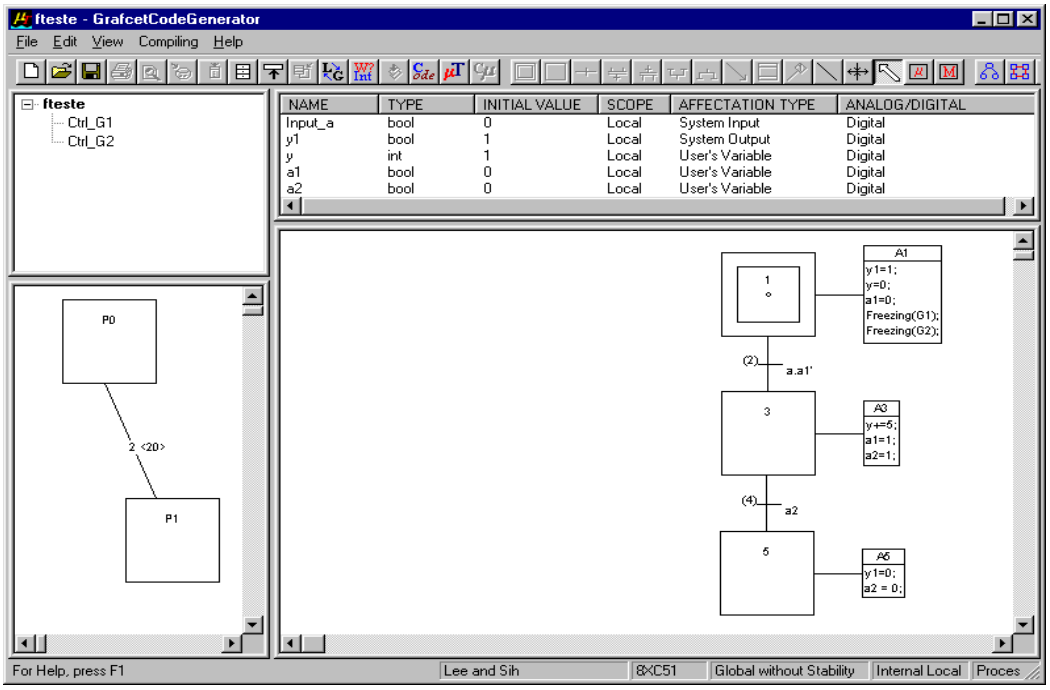

Fig.2 Application modelling: architecture graph and partial grafcet of level 0

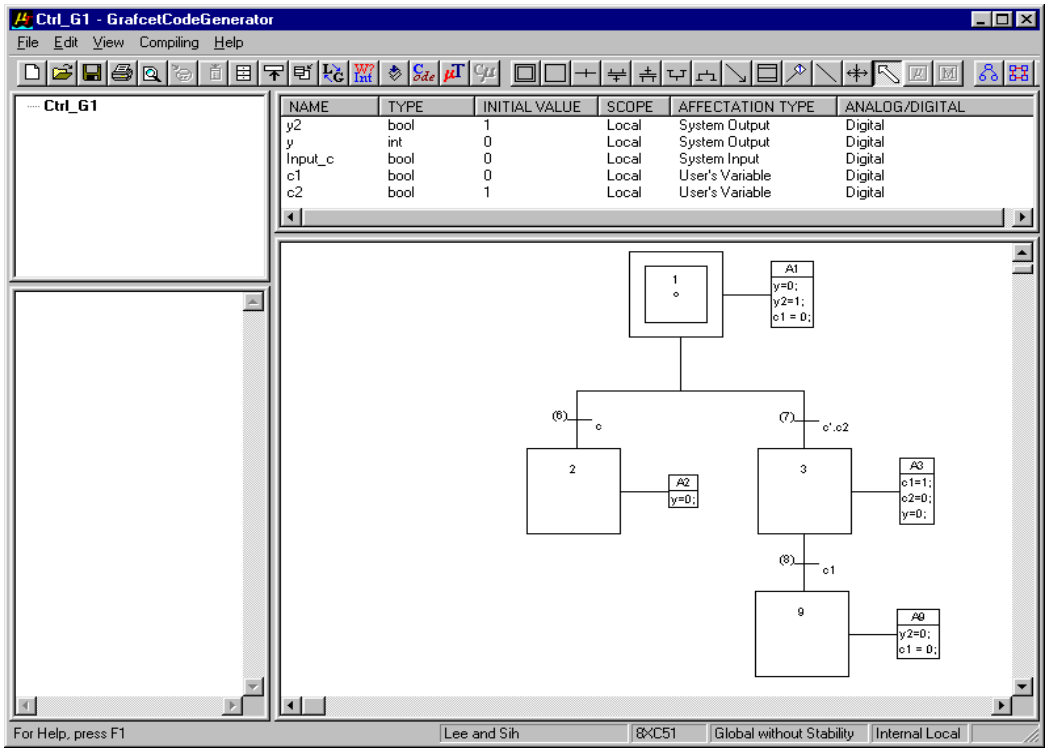

Fig.3 Application modelling: first partial grafcet of level 1

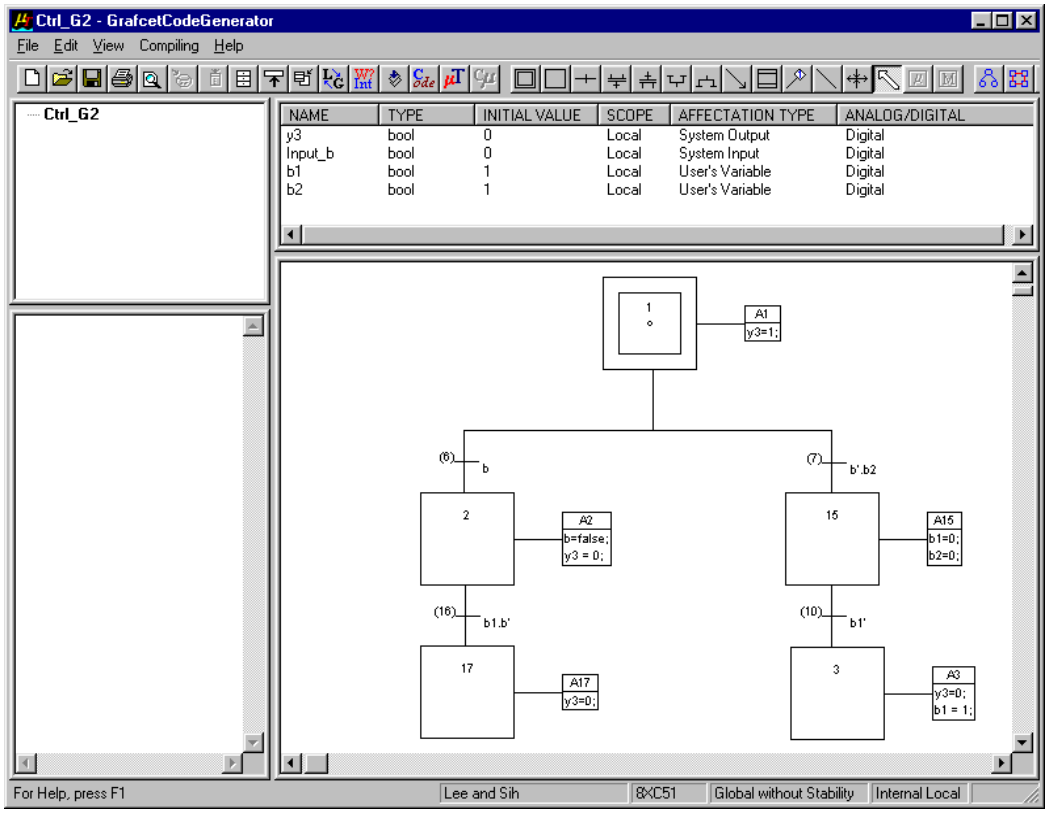

Fig.4 Application modelling: second partial grafcet of level 1

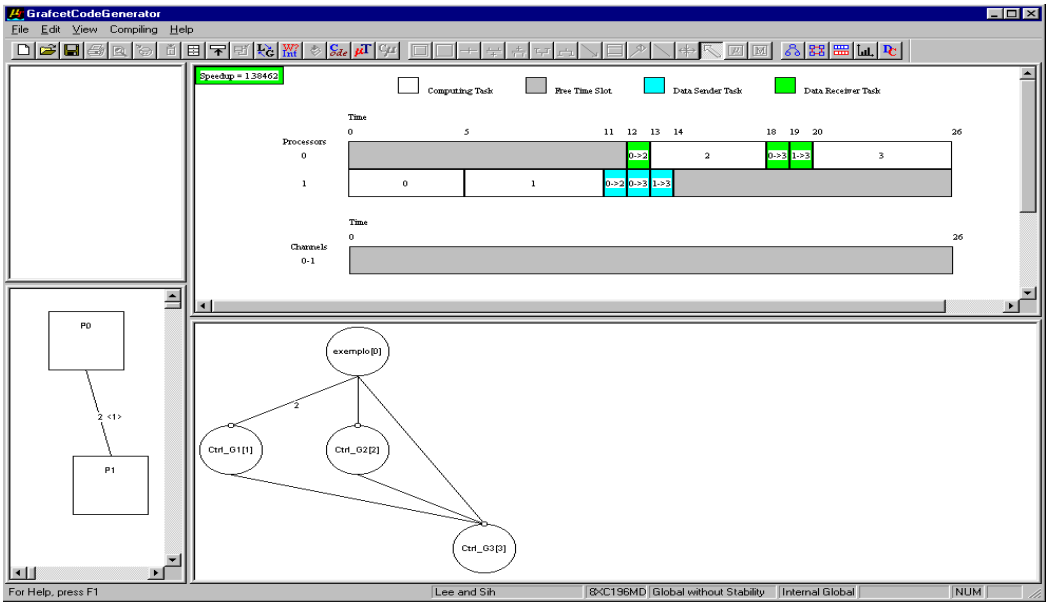

Fig.5 Upper right quadrant shows the Gantt chart obtained by scheduling the task graph onto the machine graph

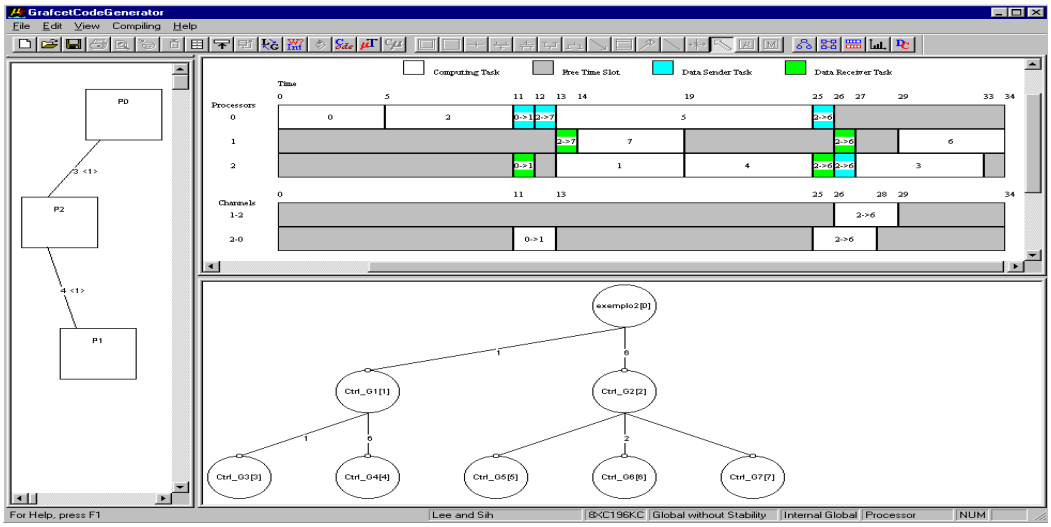

Fig.6 Upper right quadrant shows the Gantt chart obtained by scheduling the task graph onto the machine graph

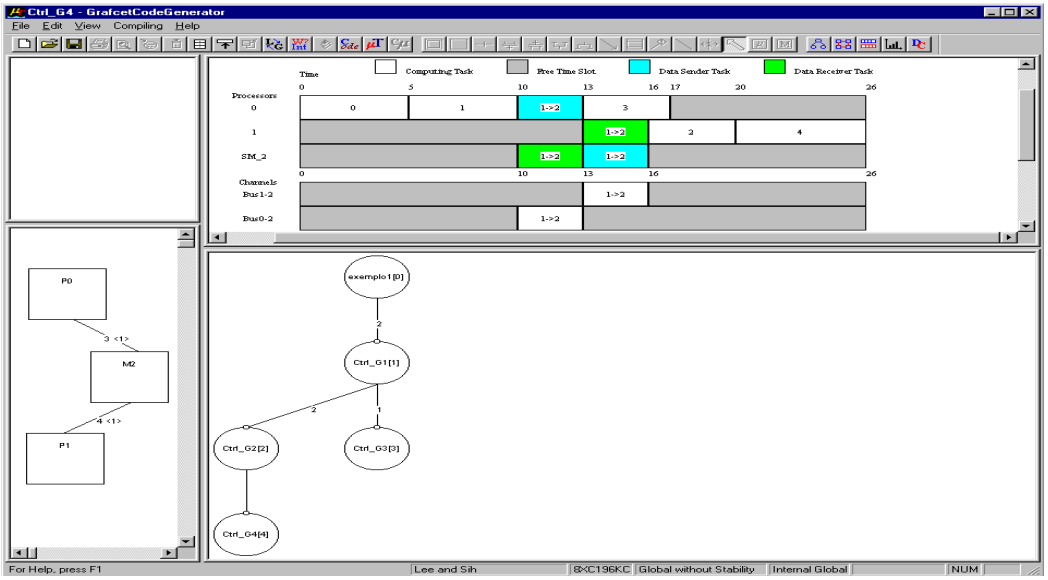

Fig.7 Schedule result onto a machine graph with a dead processor (M2). P0 and P1 communicate via a shared memory