DYNAMIC SCHEDULING ALGORITHM FOR MULTIPROCESSORS SYSTEM

[1]MISS. HEENA JANI

M.Tech Student, PanditNathulalji Vyas Technical Campus,Wadhwan City-363030

jani.heena96@gmail.com

ABSTRACT: *This research Paper to be carried out is on the design methodology required to make intelligent decisions about the RUN-TIME multiprocessor –systems-on chip (MP-SoC) management and mapping of dynamic embedded software. However, obtaining experimental evidence of the soundness of the methodology by applying it on actual applications is a major requirement. This evolution, embedded processors become ubiquitous (present every where simultaneously) and a new role for embedded software in contemporary and future Multiple-Processor systems -on-Chip (MP-SoC) is reserved. Next to these programmable components, they contain a large number of memories organized in many different ways. Hence there is a need for proper management of all the data and computation in these complex systems.*

Keywords- MPSoC, DSP,computers, design, structures

I. INTRODAUCTION

The merging of computers, consumer and communication disciplines gives rise to very fast growing markets for personal communication, multi-media and broadband networks, in the information technology (IT) area. Rapid evolution in sub-micron process technology allows ever more complex systems to be mapped on platforms that become integrated on one single chip. Technology advances are however not followed by an increase in design productivity, causing technology to leapfrog the design of IT systems. A consistent system design technology that can cope with such complexity and with the ever- shortening time-to-market requirements is of crucial importance. It should allow mapping these applications cost-efficiently to the target architecture while meeting all real-time, power, and other constraints. Today, a new heterogeneous architectural design paradigm is emerging usually called a 'platform', including one or more programmable components, either general-purpose or DSP processors, cores or ASIPs (application-specific instruction-set processor), augmented with some specialized data paths or co-processors (accelerators)[3][4]. Through this evolution, embedded processors become ubiquitous and a new role for embedded software in contemporary and future Multiple-Processor Systems-on-Chip (MPSoC) is reserved[5].

II. RESEARCH

This research Paper includes both methodology and prototype tool design aspects. The short time to market available to realize these designs Indeed requires a very high productivity from the part of the designer. The key solution for this problem is writing down the specification of the system at a higher level of abstraction and providing a methodology to refine this specification into an implementation or components that manage the most time consuming aspects of the implementation through an API. The following essential research problems need to be addressed to solve this problem: Massive parallelism is needed in order to reach to low power goals. This includes sub-word parallelism, instruction parallelism and coarse multi-processor parallelism. Especially the problem of using sub-word parallelism is not solved in both academia and industry. It must be investigated how much of each type of parallelism needs to be exploited for the Ambient Intelligence application domain. Also flexible architectures that can exchange a certain type of parallelism for another can enable a better exploitation. But then the question is raised how to explore the different alternatives without increasing the design time much? Careful decisions need to be taken concerning the dynamic data management[6]. It has to be decided what data structures to use to implement the abstract data types, how much memory to allocate for each data structure and how, and how to keep track of free and used memory for each data structure. Also transformations on these dynamic data structures can significantly reduce implementation cost. Higher-level models are needed to explore and transform different data representation alternatives.

Initial research has shown the feasibility[9]. The next step is to fit this in the global design flow. All the above can be used to steer system-wide trade-offs between power consumption, memory footprint and quality or throughput/latency (within the constraints to be met). Resources are assigned to tasks such that they can meet their (real-time) constraints while minimizing cost (such as energy). These tradeoffs have to be partly decided at run-time based as much as possible on design time preparations and analysis. The current approach is mainly

based on a spatial assignment of tasks to resources in the assumption that there are more resources than tasks[6][7]. The run-time management and the design-time preparations will change drastically when allowing multiple tasks sharing the same resources over time. The challenge is how to use the design-time information appropriately to limit the run-time overhead. Optimized Design flows, Middleware and RTOS components will play a major role in addressing this challenge.

III. RESEARCH METHODOLOGY

There is a need to create a run-time management glue layer that is perfectly suited for the MPSoC environment. This component illustrates how an MPSoC-specific run-time management layer is able to alleviate the needs of the MPSoC environment. The (design time) exploration and optimization component starts out by analyzing and profiling the target application together with a set of potential platform properties (e.g. different memory sizes for different memory hierarchy layers, different communication bandwidth, number of PEs, etc.). The result of the design time exploration should be an improved, scalable application (e.g. containing multiple code versions) bundled with a large amount of design time exploration information. The design time information is coded by means of a multi-dimensional Pareto curve. The application itself will be expanded with run-time resource management functionality such as e.g. block transfers to manage the (scratchpad) memory hierarchy[1][2].

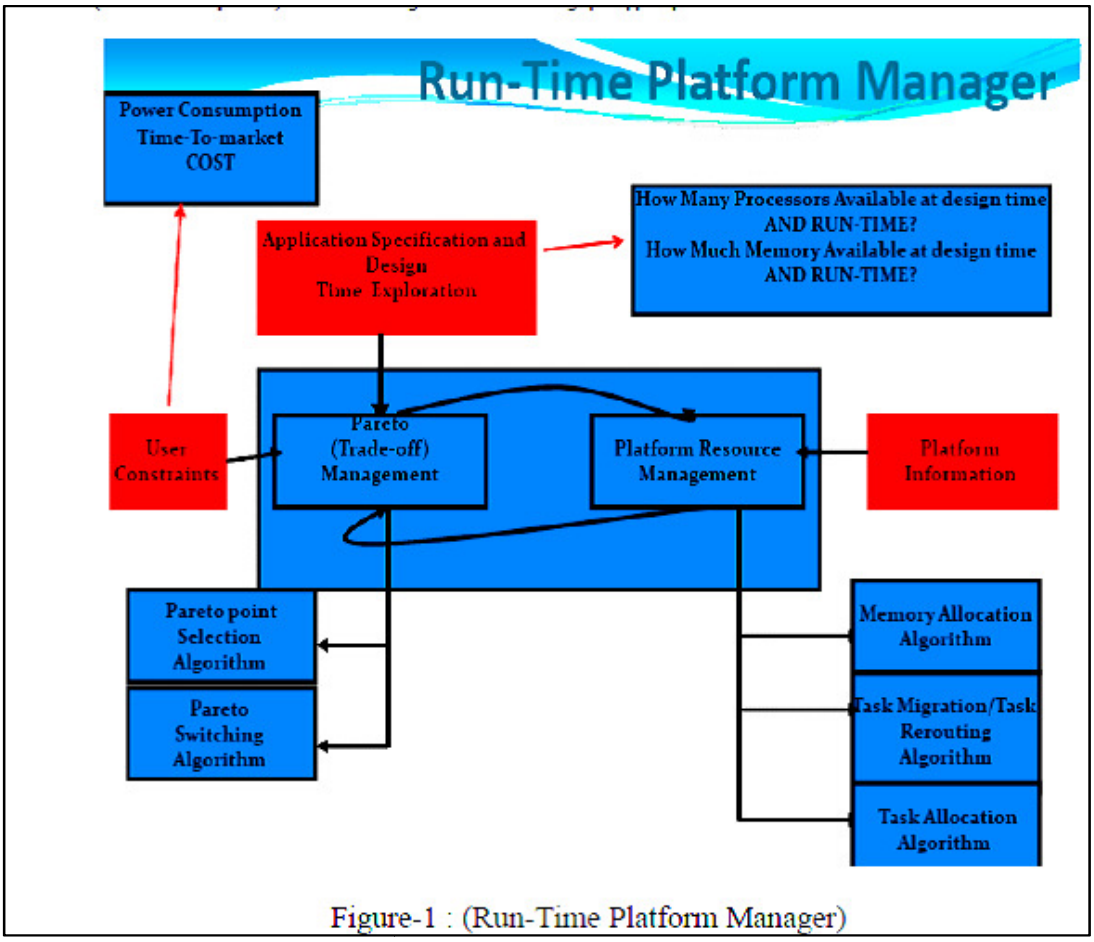

Figure-1 : (Run-Time Platform Manager)

The run-time platform manager (above figure) takes the improved application and the Pareto curve as input. Consequently, the run-time manager selects an active Pareto point based on the quality requirements (real-time deadlines, energy constraints, etc.) and the available platform resources. The exploration information allows the run-time manager to predictably tune the application performance to the available resources and the user requirements/constraints.

Once the resources have been assigned, the application itself will be partly responsible for managing these resources. For example, the application is responsible for managing the assigned L2 memory block or to make sure data is copied in a timely fashion from its assigned L2 memory block to its assigned L1 memory block[5]. Although the run-time manager is responsible for handling the application's request for a memory block, the

application itself is responsible for managing the allocation of memory resources within that block. This cooperation obviously requires special constructs within the MPSoC runtime manager[1][4][10].

In addition, it should be possible to perform a Pareto point switch at run-time in order to react to a changing environment (e.g. when a new application is started, when user requirements change, etc.). Here the run-time manager will be responsible for selecting the destination Pareto point and for guiding the Pareto switching. However, deciding when the Pareto switch should occur needs to be decided in cooperation with the application. Concretely, the goal of the MPSoC run-time management research activity should be to create RTOS extensions and algorithms that efficiently exploit the presence of design time application exploration information in order to manage the resources in an MPSoC environment. Hence the research deals with the thepareto's principal that is; the unequal distribution of wealth in his country, observing that twenty percent of the people owned eighty percent of the wealth. There is a management theory floating around at the moment that proposes to interpret Pareto's Principle in such a way as to produce what is called Superstar Management. The theory's supporters claim that since 20 percent of people produce 80 percent of results We should focus limited time on managing only that 20 percent, the superstars. The theory is flawed, as we **are discussing here** because it overlooks the fact that 80 percent of time should be spent doing what is really important. Helping the good become better is a better use of time than helping the great become terrific. Apply the Pareto Principle to all we do, but use it wisely[9].

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar. The goals of the run-time management research in the MPSoC activities deals with:

A. Pareto Point Selection:

The main question here deals with how to select a Pareto point in the Pareto hyper surface that minimizes the cost (according to some cost function) for a certain required performance and that considers the available platform resources. As the above figure illustrates, run-time management contains two major components: requirements management, dealing with (i) Pareto point selection and Pareto point switching according to the user requirements and (ii) with the actual assignment of the resources. Hence, this raises the question of how the Pareto point selection and the resource assignment algorithm will be linked. Since selecting a Pareto point also means having an amount of available resources, a specific Pareto point can only be chosen if those resources are (i) available and (ii) can be allocated[10].

B. Run-time Pareto Point Switching:

The main question in this case is how to efficiently switch between complex Pareto points? There are a set of issues that have to be considered when performing a Pareto switch. Choosing a new Pareto point (also denoted as destination Pareto point) may be dependent on the current point. Making an assessment of the amount of work involved when switching from one Pareto point to another can partly be done at design time by creating a Pareto point switch cost matrix. However, there will be some run-time issues that need to be taken into account: it might be that due to certain run-time conditions (e.g. platform resource allocation status), some destination Pareto points are favored[2][4][8].

C. Resource assignment:

In this research Paper, we have to find some good resource allocation algorithms to solve the actual resource assignment problem after a certain Pareto point has been chosen. This assignment problem includes assigning tasks to processing elements, assigning the requested memory blocks to specific memory layers/blocks and assigning the communication resources (see above figure). The algorithms performing this allocation need to be tightly coupled. In addition, they have to yield good solutions in a minimum of time, since assignment of resources happens at run-time.

IV. CONCLUSION

Proposed RPM (Run-Time Platform Manager) Architecture System will reduce the time of the MPSoC by implementing Run-time Application, save the memory block area at run time for multiprocessors, Reduce the Cost, order to reach to low power goals. The application itself will be expanded with run-time resource management functionality such as e.g. block transfers to manage the (scratchpad) memory hierarchy. Proposed System will manage and follow the new heterogeneous platforms like MPSoC, l provide the Prototype at RUN-TIME MPSoC., Manage the real-time deadlines, energy constraints, low power consumption, Speed, area, Application Performance, Time to market and manage resources Which all are challenges in Design time MPSoC.

V. REFERENCES

1. K. Asanovic, R. Bodik, B. Catanzaro, J. J. Gebis,P. Husbands, K. Keutzer, D. Patterson, W. Plishker,J. Shalf, S.
2. W. Williams, and K. A. Yelick. Thelandscape of parallel computing research: A view fromberkeley. Technical Report UCB/EECS-2006-183,EECS Department, University of California, Berkeley,Dec 2006.
3. P. Bellens, J. M. Perez, R. M. Badia, and J. Labarta.CellSS: a programming model for the cell bearchitecture. In Proceedings of the ACM/IEEE Supercomputing 2006 Conference, November 2006
4. M. Bimberg, M. Tavares, E. Matus, and G. Fettweis.A high-throughput programmable decoder for LDPCconvolutional codes. In Proceedings of the 18th IEEEInternational Conference on Application-speci¯cSystems, Architectures and Processors (ASAP'07),Montreal, Canada, July 2007.
5. K. Flautner. The wall ahead is made of rubber. In 4thHiPEAC Industrial Workshop on Compilers and Architectures, Cambridge, UK, November 2007
6. E. A. Lee and D. G. Messerschmitt. Synchronous data°ow. Proceedings of the IEEE, 75(9):1235{1245, 1987.
7. T. Limberg, M. Winter, M. Bimberg, R. Klemm, E. Matus, M. Tavares, G. Fettweis, H. Ahlendorf, and P. Robelly. A fully programmabel 40 gopssdr single chip baseband for lte/wimax terminals. InProceedings of the 34th European Solid-State Circuits Conference,ESSCIRC, Edinburgh, Scotland, September 2008.
8. D. Markovic, B. Nikolic, and R. Brodersen. Power andarea minimization for multidimensional signalprocessi
9. Dr.D.S.Vyas "Architectrue of Run-Time Platform Manager For Dynamic Data Management in MPSoC(MultiProcessor-System-On-Chip) Volume 2, No. 6, Nov-Dec 2011, ISSN No. 0976-5697