Parallel Programming for Many-Core SoC

Alexey Syschikov
SUAI, Institute HPCNT
Saint-Petersburg, Russia
alexey.syschikov@guap.ru

Abstract

The common Many-Core SoC architectures is a quick-growing segment of distributed parallel computing systems. Such systems cannot be effectively programmed "manually", however there are only basic instruments and tools for programming.

In this article we propose the concept of Many-Core SoC parallel programming. The concept presents visual programming approach, coarse-grained parallel program organization with sequential grains and dynamics of parallel computation. There are implemented the set of tools named VIPE: VPL Integrated Programming Environment. VIPE includes strict mathematical formal model basis, visual parallel programming language for coarse-grained programs design, toolset for mapping, translating and pre-compiling of the program scheme and the Many-Core SoC simulation software.

INDEX TERMS: PARALLEL PROGRAMMING, DYNAMIC COMPUTATIONS, INTEGRATED PROGRAMMING ENVIRONMENT.

I. INTRODUCTION

The common Many-Core SoC architecture template consists of processing elements (containing computational core, local memory and communication controller), communicated with some communication environment. The example of such architecture is represented below:

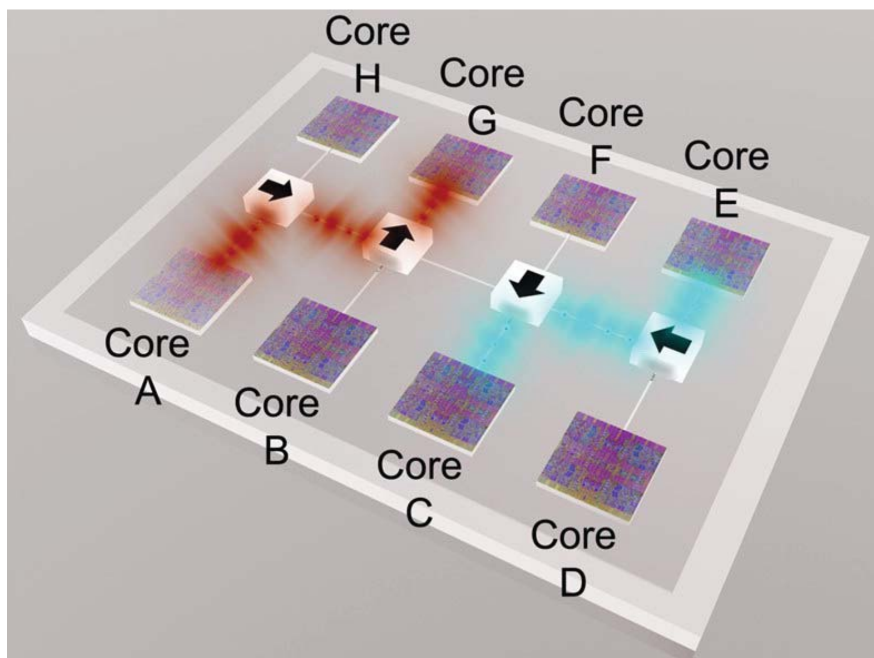

Fig. 1. Many-Core SoC architecture example

It is obvious that such architecture inherit all advances and lacks of distributed multi-processor computing systems. Most important of them are:

+ – Simplicity of scaling;

– – Complexity of programming.

Systems with clustering of processing elements form the separate specific subclass of the common Many-Core SoC architecture.

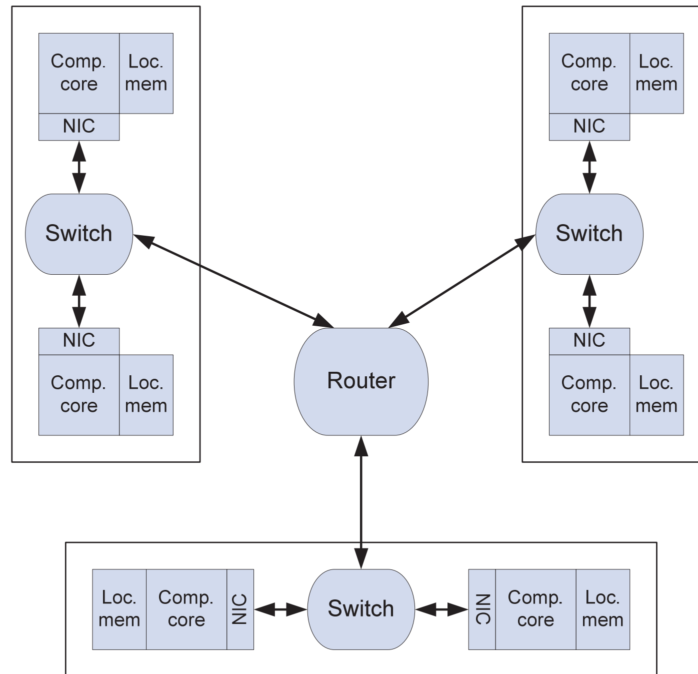

Fig. 2. Clustered Many-Core SoC architecture

For such systems all above characteristics stands correct, however the specified advantages and lacks become stronger: such systems are simpler to scale and more complex to program.

In this article let us stand on the question of Many-Core SoC programming.

II. MAIN PART

The main tasks in programming of Many-Core SoC are:

1. Decomposition of computational algorithm into blocks that can be effectively executed on the cores of distributed system.
   The complexity of this task: processing elements cannot work with shared data (shared memory is not presented in such systems). Thus decomposed blocks should depend just from incoming data and perform only local computations with these data.
2. Allocation (allocation scheme) of the decomposed algorithm on the processing elements of distributed computation system.
   The complexity of this task: duration of algorithm execution and performance of the computing system in a whole considerably depends on the allocation quality. Bad allocation leads to inefficient utilization of computational resources, to

overload of some processing elements or to overload of communication system, which leads for other part of computing system to stay idle.

3. Organization of data interchanges between allocated algorithm blocks and processing elements to which blocks are allocated.
   The complexity of this task: organization of interchanges besides data transfer also requires performing connection set, negotiation of interchange protocols, synchronization of computations and communications and other actions.

The separate task is the selection of computation granularity level, i.e. to decide what size should have blocks of decomposed computational algorithm to effectively allocate blocks on the system modules, to perform computations effectively enough, to evenly load system etc.

Undoubtedly, distributed computing systems can be programmed “manually”. For small enough tasks and large-grain algorithms it’s a feasible task. However for real-size tasks and for algorithms that should be portable between computing systems with different configurations it’s nearly impossible.

The MPI standard can be named as the main instrument for distributed systems programming. It is a standard de-facto for distributed systems programming, it have wide range of functional abilities and it have many implementations for different architectures and operation systems.

However, MPI is still a nearly low-level standard and it doesn’t allow to solve effectively the algorithm decomposition task and the computation blocks allocation task. In addition it has the set of lacks that are mostly appreciable on the low-power computing systems:

1. Programs written using MPI standard works only with MPI loader and runtime that rise system processing elements occupation with overheads.
2. Programming using MPI standard still requires a lot of manual work to organize data interchanges and synchronization.

The separate problem is the programming and debugging of algorithms when there is no ready hardware platform implementation. The presence of ready and tested algorithms to the moment when a hardware platform will become ready is a very significant topic: nearly nobody need hardware platform without software. In addition, from the hardware platform appearance it faster goes out of date, thus it’s not profitable or even impossible to wait for software for the new platform appears. I.e. the development and testing of the software should be done in parallel with the development of the hardware platform.

A. The concept of Many-core SoC programming

We propose the many-core SoC programming concept that has the set of key thesis:

1. Coarse-grained parallel computations with sequential processing inside coarse-grained blocks;
2. Visual approach to parallel programming;
3. Dynamics of parallel computations.

Now we introduce in details what every of thesis involves.

A.1. Coarse-grained parallel computations

The distributed Many-core SoC computing system is traditionally built using the specialized processing elements with not so high productivity.

Thus for such systems it looks reasonable to use coarse-grained parallel computations, where the element of computation has the amount of code equal to functions or procedures of traditional programming languages. Such computations in nature correspond to the organization of computations in parallel computing systems with distributed architecture.

With this the processing elements to which the decomposed coarse-grained algorithm blocks will be allocated rarely have wide abilities for parallel execution or have a specific

command set for internal parallelization. Thus it's inconvenient and ineffective to program and parallelize computations inside the decomposed block (fine-grained programming, i.e. programming on the operator level) using the same instruments as for decomposition of the algorithm. So for such programming task we propose to use traditional programming language or specialized assemblers with further linking with coarse-grained scheme objects.

A.2. Visual approach to parallel programming [1]

A decomposed algorithm with obviously specified split blocks and data and control dependencies between blocks corresponds to a network of objects or graph. Graphical (visual) representation is a most natural representation of a graph or network.

A.2.1 Advantages of the visual approach

Let's see, what abilities and advantages are proposed by a visual approach against a traditional programming.

1. Integration of design and programming.
   What is the common way to write a program? Software designer takes thw source algorithm description (text description, abstract schemes, SDL diagrams etc.) and draw either scheme of the future parallel program or the state mschine of the future system. After that programmers take this scheme and write it on the programming language. I.e. in fact the program scheme is described twice: the first time in the form that is natural for designer and the second time in the form natural for compiler.
   Graphical (visual) languages allow avoiding double work: designer can draw program scheme immediately in terms of visual programming language.
   In addition this approach allows avoiding problems with further synchronization of changes in design scheme and program.
2. Obviousness and interactivity.
   Visual approach provides obvious representation of the language objects interconnections, their dependencies, interactions, control etc. In addition visual approach makes possible to use interactive development environment for dynamic representation of program parts, hierarchical constructions, grouping and consolidation of objects, visual debugging etc.
3. Obvious parallelism.
   Program on any visual language in nature is a scheme. Writing program as a scheme allows describing obvious parallelism naturally, moreover it stimulate to describe parallelism.
   This gives wide abilities to make automatic parallelization of parallel program objects that are independent by data or control and organize pipeline parallelism for dependent objects.

For implementation of this concept the visual parallel language VPL was developed.

A.2.2 Visual parallel programming language VPL [2]

The experience in visual language development shows: graphical approach have a set of unquestionable advantages, however it cannot fully substitute traditional programming languages. A solution was found in usage of combination of graphical (VPL language) and traditional approaches: visual programming of high-level program scheme and programming on traditional languages for interpretation of scheme elements.

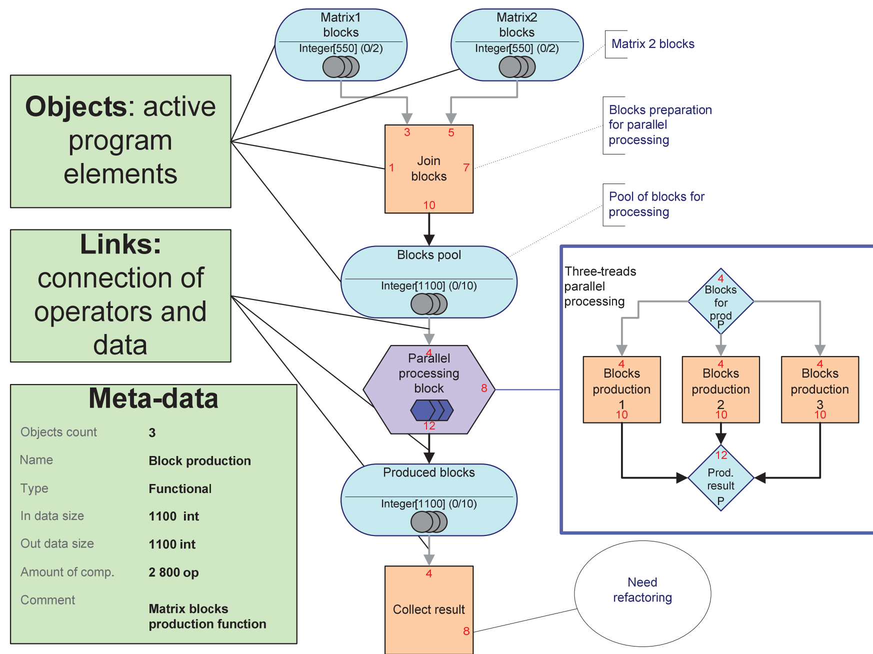

Fig. 3. Program in visual parallel programming language VPL

Parallel program is represented as an hierarchical network of operators and data-objects. Operators are an abstraction of active program components – functions, procedures, tasks (depends on granularity of parallel computation). Data-objects are an abstraction of any data storage types that exists outside operators.

All interactions between each other or with data-objects are obviously represented on the parallel program scheme level.

Operators interact through data-objects. All data that can be accessed by several operators should b obviously represented on the parallel program scheme as data-objects.

VPL language is algorithmic complete, it provide abilities to organize computations control depending on data values on the level of parallel program scheme. Programmer neither need nor can to control parallel computations inside the sequential programs of separate functions. This prevent interactions in parallel program from being not observable and parallel program scheme from being chaotically and non-verifiable.

Terminal operators of program scheme are the only instrument to make data processing. They are described on traditional textual sequential programming languages. This allows describing local data processing in processing elements of parallel computing system on habitual programming languages and in habitual terms.

A.3. Dynamics of parallel computations

Computational tasks of real life rarely can be laid onto simple algorithms with linear structure. In common algorithms of such tasks contain lots of conditional branching, alternative paths, cycles etc.

Undoubtedly the proposed concept and VPL language allows comfortably describing such computations, however for effective computations organization it’s not enough, especially in cases when we have limited resources.

Of course it’s possible to build static graph with many alternative paths. However it will contain lots of nodes especially when there will be hierarchical branching. Allocation of such graph cannot be build effectively enough: it is impossible to define on an allocation building stage which branches will be executed in which moment because they will depend on the

values of processing data and on other conditions. In addition the presence of redundant amount of nodes in graph will require a lot of resources on system processing elements for allocation, which will be used inefficiently: there is a very low possibility that all allocated branches will be executed.

To solve this problem we propose the model of dynamic parallel computations: it means that program graph during program allocation to the program still remains hierarchically organized. Only when condition for some control operator will be calculated its branch for these and only for this condition processing will be dynamically unrolled. Similarly to this the parallel cycles will work: there will be dynamically unrolled the exact amount of iterations that is needed to process the current data.

It should be noted that our concept allows describing dynamic computations but it is not mandatory. Moreover the most effective is to use the rational combination of static and dynamic unrolling methods. For example, for some tasks and computing systems it's more effective to use fully static approach.

The selection between static and dynamic implementation of operators currently should be done manually be an algorithm designer and programmer. In future the appearance of tool that will automatically optimize the usage of dynamics for an exact configuration of platform and task is possible.

A.4. Examples of developed programs

Within the scope of the presented concept there were developed the set of programs from the area of communication algorithms, image processing etc.

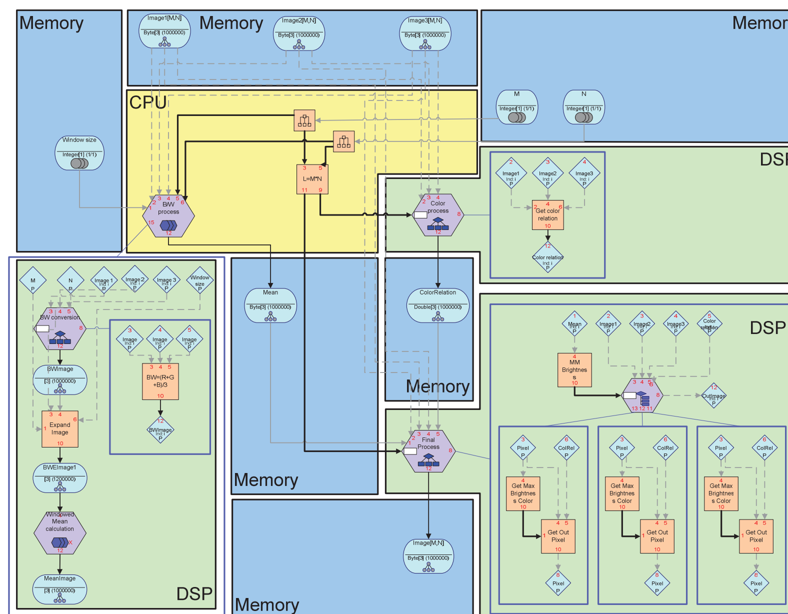

Fig. 4. Example of parallel program in VPL language

On the presented example there can be seen some advantages that are provided by the proposed concept, by visual approach and by programming language.

1. Obviousness of program scheme that represent an algorithm scheme;
2. Obviously seen data and control dependencies between algorithm objects;
3. Hierarchical program composition;
4. Abilities of clustering and specialization of separate blocks, components etc.

B. VIPE: VPL Integrated Programming Environment

For distributed Many-core SoC computing systems programming on the VPL language the software complex was developed. It allows performing all necessary stages for development, debugging and testing of distributed computing system software.

The general scheme of the VIPE (VPL Integrated Programming Environment) is the following:

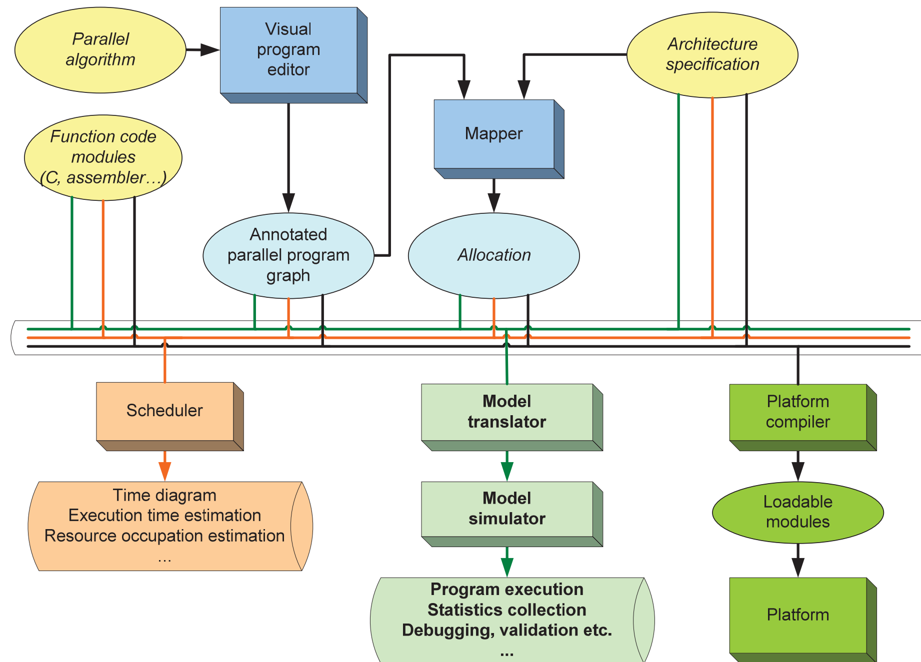

Fig. 5. The general scheme of the VIPE

- The software development complex consists of the next base components:
- *Visual designer*. The environment for developing parallel program on the VPL programming language;
- *Translator*. The tool for transformation of program scheme in internal system format into the unified annotated graph format for other complex components and for external tools;
- *Mapper (allocation builder)*. Tool for automated allocation of program scheme elements to processing elements of Many-core SoC;
- *Pre-compiler and linker*. Code preparation for program graph, linkage functional code and program scheme, code generation for execution on heterogeneous Many-core SoC, linkage of system loadable modules.
- *Software simulator*. Simulation of parallel program execution on different platform models and configurations, algorithm program debugging and testing without requirement to have ready hardware platform.

The presented software complex VIPE – VPL Integrated Programming Environment allows to:

1. Develop parallel algorithm for the specified task, perform hierarchical decomposition to the required granularity level, design parallel algorithm that will be effective for execution on parallel Many-core SoC with distributed structure.

2. Estimate execution time and computation complexity of the algorithm on the early stages. This will allow making a frame estimation of requirements to computation resources, memory, and communication system.
3. Make a full algorithm program; perform debugging and testing of developed software. Execute software on different models and configurations of computing platform; obtain more detailed characteristics of software and requirements to hardware.
4. Perform compiling and linking for an exact platform using specialized platform compiler. Obtain loadable modules that can be executed exactly on the specified computing platform.
5. Use benefits of pipeline parallelism, natural parallelism of applied algorithms, natural dynamics of executing task to extend characteristics of parallel program, increase performance and throughput, decrease execution time and latency.

It should be separately noted that all concept, methodology and toolset is based on the strict formal mathematical model of parallel computations. It guaranty that obtained execution results will be equal either for virtual algorithm execution on simulator or for execution on hardware platform and independently from the exact platform configuration to which algorithm will be allocated.

C. Formal model of parallel computations

It doesn't look reasonable to go into mathematical details of formal computational model of parallel computations that lies in the basis of the proposed concept [3,4,5], programming language and integrated programming environment. However we need to say some words about main advantages that presence of formal model provides. Such advantages are:

1. Verification of parallel program "by design".
   Computational model allows formulating and proofing theorems, corollaries and characteristics of program schemes. Presence of VPL object formal descriptions allows applying proofed in the formal model characteristics to programs that are developed using the VPL language. Such characteristics can be:
   a. Formal correctness of program design
   b. Finding deadlocks/livelocks on translation stage
   c. Finding circularities on translation stage
   d. Ability to obtain conclusions about overall program execution basing only on some time of program execution.
2. Equivalent transformations.
   Equivalent transformations are transformations from one program scheme to another program scheme that are formally equal to the source ones using rules that are defined in formal model. The mechanism of equivalent transformations can be used for:
   a. Optimization of program scheme for specific tasks and platforms using aggregation, grouping etc.
   b. Functional validation of parallel program using debugging of equivalent sequential version with guaranty of equivalent execution of the source parallel implementation and other program characteristics.

III. CONCLUSION

Summarizing the above it can be said that the proposed concept, language and integrated environment for Many-Core SoC programming offers:

- Portability across a wide set of reconfigurable Many-Core SoC architectures
- Efficiency in usage of Many-Core SoC resources by one or more applications
- Formal verification of correctness of a parallel program scheme.
- Graphical design as a native way for expressing parallelism inherent to Many-Core SoC and applications

REFERENCES

[1] A.Y. Syschikov, "Application of visual approach for software development", Proceedings of ninth scientific session of SUAI, vol. I.: technical sciences, SUAI, Saint-Petersburg, 2009, c.130-133.

[2] Y.E. Sheynin, A.Y. Syschikov, "Task-level Parallel Programming Language for Space and Aeronautical Applications", European conference for aerospace sciences (EUCASS) (EUCASS), Moscow, 2005.

[3] Y.E. Sheynin. "Asynchronous growing processes – formal model of parallel computations in distributed computing systems". Proceedings of international conference "Distributed data processing" Novosibirsk, 1998, 111–115 c.

[4] Y.E. Sheynin, "Formal model of dynamic parallel computations in parallel computing systems of experimental data processing", Scientific Instrumentation, 1999, vol. 9, #2. c.22–29.

[5] V.I. Ivanov, Y.E. Sheynin, A.Y. Syschikov, "Programming model for coarse-grained distributed heterogeneous architecture", XI International Symposium on Problems of Redundancy in Information and Control Systems: Proceedings, SUAI, 2007, c.246-250.