Parallel Programming Model for Distributed Architecture MPSoC.

Prof. **Yuriy Sheynin**,  
Director, Doctor of Science  

190 000  St. Petersburg  
Bolshaya Morskaya str., No 67  
Fax: +7 812 3157778  
E-mail: sheynin@online.ru
Distributed Architecture MPSoC

- Heterogeneous distributed scalable architectures with coarse-grained cores are the trend in embedded application specific MPSoC
- Programming of coarse-grained heterogeneous MPSoC remains a challenging task
- It requires new parallel programming paradigms and computational models for application specific MPSoC -- Parallel Programming Model (PPM)
What is a **Distributed Architecture**?

- **Distributed control** – a Computing Module (Node) has its own independent local control ("program counter")

- **Distributed memory** – a local memory in Computing Module (Node) with a separate address space. No global address space

- **Message-passing** interaction between Computing Modules (Nodes)

   It could be called *Network Architecture* also, if it would not pull us too far into similarities with Internet …
Parallel Programming Model incorporates

- **Programmer’s vision of the Computing Platform**: a generalized representation of the computer (Abstract Machine) he is programming

- **Programmer’s vision of a Parallel Computation** (principles of operation and control)
  – parallel computation paradigm and model

- **Programmer’s vision of Parallel Programming itself**
  – Parallel programming methodology and Parallel programming language
Development of Abstract Machine (AM) Parallel Computational Model (CM) and Parallel Programming Language (PPL) are interrelated tasks that require integral approach.
MPSoC as a platform for application-specific parallel computations

MPSoC as a specific parallel computer is an entity for implementation of co-operating processes. Nature and features of systems of processes should define concepts and approach for a computer design and programming.

Core question:
What types of parallel computations are generated by the application-specific MPSoC workload?

- Parallelism level and granularity
- Fixed / Static / Dynamic
MPSoC Parallelism Levels

3 levels of MPSoC Parallelism:

- **Task-level parallelism** (*tlp*)
  To be used in:
  - Parallel Algorithms development;
  - Parallel source code programming;

- **Procedure-level parallelism** (*plp*)
  To be used in:
  - Parallel source code programming;
  - Source code translation and linking;
  - Parallel program optimization;
  - Parallel object code mapping to MPSoC PEs

- **MCA engines’ units parallelism** (*mup*)
  To be used in:
  - Procedure program optimization and local parallelization;
  - Multiple Procedure programs mapping to MPSoC PEs
## Static vs Dynamic parallel computations

<table>
<thead>
<tr>
<th>Static computation</th>
<th>Dynamic computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Low control overheads</td>
<td>- Higher control overheads</td>
</tr>
<tr>
<td>+ Low response time</td>
<td>- Higher response time</td>
</tr>
<tr>
<td>- Excess MPSoC resources (PEs, memory, I/O) expenditure</td>
<td>+ Judicious MPSoC resources expenditure (memory, PEs)</td>
</tr>
<tr>
<td>- Excess power consumption</td>
<td>+ Economical power consumption</td>
</tr>
<tr>
<td>- Scheduling for maximum function processing time → overrated processing time, underrated performance</td>
<td>+ Scheduling for actual function processing time → increased performance</td>
</tr>
<tr>
<td>- Problems of computation adaptation to varying tasks, MPSoC components faults</td>
<td>+ Natural computation adaptation to varying tasks, MPSoC components faults</td>
</tr>
</tbody>
</table>
Dynamic computations are parallel computations, which set of components and links between them depend on data values and change in the cause of computation.

A formal Parallel Computation Model that covers both static and dynamic parallel computations is required.

Asynchronous Growing Processes (AGP-model) - dynamic parallel computation model, that covers static parallel computations as its particular cases.

Rational parallel computations in application-specific MPSoCs - a combination of static and dynamic computations.
Formal models are required for:

- Parallel Programming Language semantics specification
- Parallel algorithms and programs optimization, verification and debugging
- Mapping parallel programs to distributed heterogeneous MPSoC structure
- Tolerant control of distributed parallel computations in MPSoC
- Methodology for balancing parallel software with MPSoC features and characteristics
Asynchronous Growing Processes (AGP-models)

Schema – Partially-interpreted Schema - Program

General AGP-model of parallel computations → Non-interpreted program scheme

Interpretation of control components in a program schema, restrictions on program scheme structure

Special AGP-model of parallel computations → Partially-interpreted program scheme

Interpretation of all the components of a program scheme

Language of parallel Abstract Machine → Fully-interpreted program scheme

= Program
AGP-model features and ideas (informally)

- Parallel program scheme is represented by a directed graph. Vertexes represent *operators* and *data-objects*.

- *All* interactions of processes are *explicitly* represented in the parallel program scheme. Processes interact through data-objects. Thus it can be controlled and verified at the level of parallel program scheme.

- Data, which are accessed by several operators, are explicitly represented in parallel program scheme as *data-objects*. Thus, data shared by several processes are in the frame of the model, as well as buffered data between a pair of operators (like in data-flow graphs)

- Control of computation in the AGP-model is defined in correspondence with MPSoC distributed architecture features. Control is *distributed, parallel* and *asynchronous*. It helps to fill MPSoC resources with computations and to pull high-parallel computations through limited MPSoC resources.
AGP-model features and ideas (informally), continued

- Parallel program scheme is transformed, in general, at every computation step – Dynamic parallel computation. The *graph itself* is changing, not only its marking (as in data-flow computations or Petri nets).

- Alternative computation (*if, case*, etc.) can be implemented as generation of alternative parallel program scheme fragments, instead of routing data to one of data-flow branches, which simultaneously occupy resources. Thus we can save MPSoC resources and power consumption.

- Static parallel computations can be represented as particular cases of dynamic computation. It gives a way to seamless integration of dynamic and static parallel computations in a single formal model.
We believe:

- Programs and algorithms should be developed as parallel ones from the beginning. Inherent parallelism of an application should be defined and extracted at user/application level.
- Parallel programming (with a right language and right tools) is not more complicated than sequential programming.
- Parallel program should be rather made correct automatically (correct by construction, verification), than debugged.
- Sequential programs of processes should be absolutely encapsulated. No inter-process interaction directives inside a sequential program!
MPSoC Parallel programming concepts

- Splitting programming into
  - programming of a parallel program scheme and
  - programming of interpretation of its nodes – operators and data-objects

- Explicit programming of a parallel program scheme

- Two levels of programming languages:
  - new PPL for parallel computation scheme programming
  - conventional programming languages (C, Embedded C) for sequential process programs.

  It corresponds well to the coarse-grain functions in application-specific MPSoCs.

- Algorithmic completeness
  - means for computations control in dependence of data values
  at the level of a parallel program scheme
Visa
-- Parallel Programming Language for parallel program schemes programming

- The **Visa** language semantics is formally specified in terms of the AGP-model
- Control operators are generators of program scheme fragments
- Program control of a computation - through generation of different scheme fragments depending on data values. Dynamic and Static parallel computations unrolling.
- Visual (graphical) PPL for programming of a parallel program scheme. Parallel program scheme visual representation as a hierarchical diagram
- **Visa** - *Interactive* language and visual programming tools
- Scalable language. Standard operators (library functions) and user definable operator types and data types (user-written C code for functions)
A *Visa* program fragment Example

Commented view
Conclusion

The Parallel Programming Model and the Visa PPLanguage give a way

- To work with *static and dynamic* parallel computations in MPSoC, with their manageable integration in particular application software for MPSoC
- To work with distributed memory paradigm (e.g. data-flow computations, message-passing) and with shared data
- To represent different programming styles and paradigms (MIMD, data parallel, data flow) in the single Programming Model, thus – integrate them in one software system
- To mate parallel scheme programming in new PPL with programming of its nodes in conventional programming languages
- To have manageable and adjustable granularity for application-specific MPSoC parallel computations.
- To build correct-by-construction parallel programs or to verify parallel program properties
- To reduce parallel program debugging to debugging of its sequential implementation
Conclusion

- Research is going on to get all these fine features and properties for wider classes of parallel software for MPSoC
Thank you!