**SYMParallel**: Efficient and Scalable OpenMP-based System-Level Design

**SYMParallel** is an innovative design framework for massively parallel applications, blending the OpenMP® programming paradigm with electronic system-level (ESL) design methodologies. A detailed treatment standard for parallel programming, OpenMP® is the most popular approach to shared-memory parallelization of distributed-memory computing systems, ranging from financial applications to scientific computing and graphics processing. [1] OpenMP® naturally meets the characteristics of current multi-processor systems-on-chip (MPSoCs), in that it provides essential variability to express parallelism of system state through applications. The **SYMParallel** design flow enables the automated translation of the OpenMP® programming paradigm into an advanced multi-processor system-on-chip highly customized for the target application, possibly accelerated with hardware cores inferred from the source code through a high-level-synthesis process.

A few key works address the design of hardware accelerators through OpenMP®. They focus either on the integration of hardware accelerators in essentially software-hardware prototyping (ESI) or on pure hardware translation [2,3]. When supporting OpenMP® based hardware design, they impose drastic restrictions to the constructs actually available to parallel programmers, effectively preventing the reuse of legacy OpenMP® programs and tools. Other limitations include the use of centralized mechanisms for controlling interactions among threads, causing scalability issues, the limited support for external memory and efficiency-critical mechanisms such as caching, and several unsupported features of OpenMP®.

The **SYMParallel** design flow enables the generation of heterogeneous systems, including one or more processors and dedicated hardware components, where OpenMP® threads can be mapped to either software or hardware. This provides key benefits making the full OpenMP® difficult to implement in hardware as managed in software, where a full OpenMP® application is addressed by dedicated software/hardware parallelism. Hardware threads are generated by means of high-level-synthesis tools that perfectly fit the structure of an OpenMP® program, where the application logic is still developed by means of the OpenMP® approach. This provides full support for standard-compliant OpenMP® applications, as well as functionally “generic-purpose” code, such as memory hierarchies and management.

The segment below shows the typical architecture of a system generated by **SYMParallel**. Each subsystem represents an OpenMP® thread, or a group of threads executed in parallel. The OpenMP® directives either indicate shared memory blocks or contain directives specific for the target hardware. These directives are interpreted by the compiler to generate the source code in the form of reports and graphical diagrams helping the designer visualize and understand the actual design requirements. This allows design choices to be made as early as the specification stage, without any simulation.

**Blending parallel programming with ESL design**

The **SYMParallel** design environment exposes a Graphical User Interface (GUI) seamlessly integrated into the Eclipse IDE as an external plug-in. Programmers are provided with a quick and user-friendly interface, translating directly from functional simulation to system implementation and execution. The system-level design methodology is implemented by a high-level synthesis toolchain that takes as input the OpenMP® code and generates the source code for the resulting MPSoC. The designer can optionally turn on/off specific OpenMP® constructs.

**Applications:**

- **Graphical Processing with **SYMParallel**: An important application domain for parallel computing due to the massive thread-level parallelism inherently present in most algorithms. A common example of graphical processing applications, the JPEG compression, is employed in the **SYMParallel** environment to compress bitmaps images. According to the JPEG standard, the first, relatively expensive step is the cosine Transform. Quantization, and their inverse transformation into the domain of a more intuitive and user-friendly interface. This way, the designer can easily compile, debug, and launch the generated code.

- **Computational Finance with **SYMParallel**: A financial application, i.e. we could use a fully-fledged OpenMP® code on a different platform, allowing for a more cost-effective solution. The result is that the Monte Carlo Option Price in a numerical method often used in computational finance to calculate the value of an option with multiple sources of uncertainty and risk is implemented as a black-box problem, running on a distributed system. This system is built using **SYMParallel**, using the Multi-Threaded Processor (MTP) as the target platform. The system-level design methodology allows us to define the behavior of the system on the fly, by specifying the number of cores, the number of threads, and the amount of memory to be used. This allows us to define the behavior of the system on the fly, by specifying the number of cores, the number of threads, and the amount of memory to be used. This allows us to define the behavior of the system on the fly, by specifying the number of cores, the number of threads, and the amount of memory to be used.

- **Watermarking and Encryption with **SYMParallel**: Security is a key concern in the development of multimedia applications. The **SYMParallel** environment allows designers to embed watermarking information directly into the source code, ensuring its security and integrity. This is achieved through a patented technique that uses a combination of hardware and software acceleration to generate unique watermarks for each content item.

**References**


