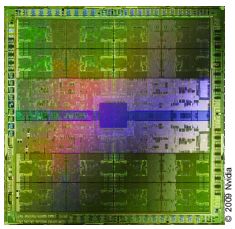
Mapping Simulation Algorithms to NoC MPSoC Computers

Prof. Dr. Hans-Joachim Wunderlich
Dipl.-Inform. Claus Braun

Mapping of Simulation Algorithms

Architectural trends for Many-Core architectures (MPSoC) move away from complex, monolithic cores with complex features of instruction level parallelism to massive thread parallelism. Different levels of on-chip memory and limited off-chip memory bandwidth are now important technology constraints. Another new aspect of these architectures is the on-chip communication which is implemented by Network-on-Chip (NoC) structures.

Example: Nvidia GT300 “Fermi” architecture
- 3.0 billion transistors
- 512 processing cores
- 16 Multiprocessors with 32 cores each
- 384-bit memory interface
- IEEE 754-2008 Floating-Point
- Up 630 GFLOPS for double precision ops.
- Up to 4.2x faster than previous generation in double precision applications



Nvidia GT300 “Fermi”

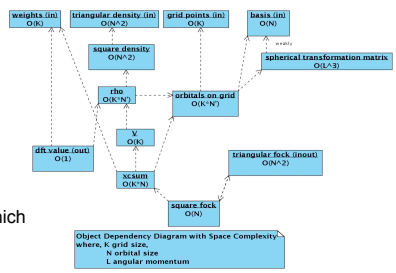
Challenge: Determine the Optimal Mapping

Programmers have to cope with explicit parallelism and the management of different levels within the memory hierarchy. A central role is assigned to communication and synchronization. To achieve optimal mappings of simulation algorithms, several steps of detailed analysis are necessary:

Analysis of algorithms

Efficient algorithms are the key to successful problem solutions. Many-Core architectures differ in many ways from traditional CPU designs which makes a conscientious adaption or even a re-design of existing algorithms necessary.

- Identification of serial and potentially parallel parts
- Identification of time and space complexity
- Identification of possible data dependencies
- Finding of suitable mapping and partitioning schemes which match the structure of the target hardware

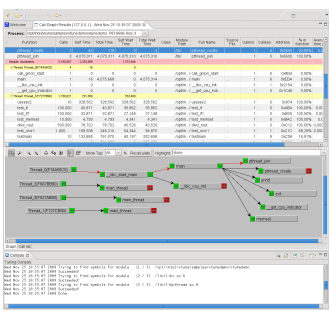


Dependencies within a Density Functional Theory algorithm

Analysis of existing code (Profiling)

In some cases, a re-design of an existing algorithm is not possible due to constraints like time or costs. The mapping of such sequential or coarse-grained parallel code to modern Many-Core architectures can be a challenging and elaborate task. Profiling tools enable the programmer to gain necessary insights into the run-time characteristics of such algorithms.

- Identification of computational demanding parts of the code
- Identification of possible data dependencies and access patterns
- Determination of calling relation-ships between different parts of the code (call graph analysis)
- Identification of communication patterns
- Identification of resource allocation (registers, memory, etc.)

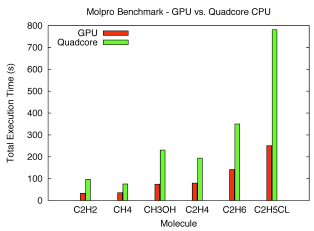

Call-graph analysis using profiling tool Intel VTune

Mapping Algorithms from Quantum Chemistry

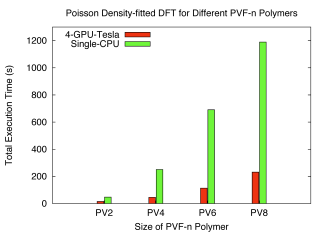
Project-Network Cooperation with Institute for Theoretical Chemistry to accelerate demanding computations from the field of quantum mechanics and molecular dynamics on Many-Core processors.

Finalized work: Enabling Molpro to use GPGPU devices for computations
- Offloading of BLAS 3 operations like DGEMM to GPU
- Molpro integration via BLAS standard interface, support for Fortran and C

Current work:
- Mapping Density Functional Theory (DFT) computations to hybrid Multi-Core and GPGPU systems (Nvidia Tesla 4 GPU Server)



Offloading BLAS 3 routines to GPU

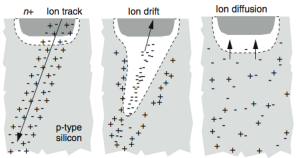


Mapping of DFT calculations to Multi-GPU Tesla System

Challenge: Ensure Reliability

Technology scaling enables semiconductor technology to operate in low nanometer scales. This allows the integration of several hundreds of processing cores and large amounts of embedded memory on a single chip. A downside of this trend is the increasing vulnerability of semiconductor devices.

Reliability harming factors
- Single Event Transients
- Power Droops
- Crosstalk
- Process Variations
- Environmental Variations



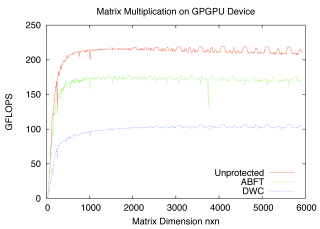
Particle strike causing a single event transient

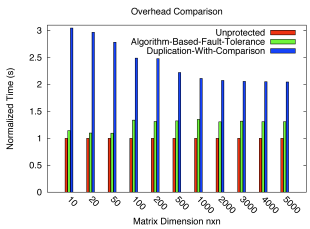
To build reliable and economical computer systems for high-performance and scientific computing, fault tolerance has to be incorporated on the hardware and on the software level.

Algorithm-Based Fault Tolerance

Algorithm-Based Fault Tolerance (ABFT) is a software-based technique to improve reliability on the algorithmic level. Matrix operations are particularly suitable for the application of ABFT. Our ABFT implementation takes the hardware characteristics of GPGPU devices into account and operates in parallel on block level to achieve maximum performance.

- Input matrices A and B are encoded by checksums (row/column checksums)
- Implementation applies ABFT schemes on sub-block level (thread block)
- Efficient mapping to the hardware and execution structure of the GPU
- Low computational overhead compared to other fault tolerance techniques like Triple Modular Redundancy or Duplication with Comparison (DWC)





Fault Tolerant Matrix Multiplication on GPU using Algorithm-Based Fault Tolerance