Programming Models for Multi-core Systems

Mujahed Eleyat

October 1, 2014

Learning Goals

- Define of a parallel programming model
- Classify parallel programming models
- Explain data parallelism and task parallelism
- Explain several approaches for parallel programming
- Present examples of parallel programming models

The Sequential Execution Model

- Programming was easy
- Applications gain more performance as the frequency of uniprocessors are increased
- The shift to multicore systems put an end to the “free lunch”
  - Automatic “good” utilization of multicore processing power is very difficult

NTNU
Norwegian University of Science and Technology

Challenges of Multicore Programming

- Parallel algorithms are hard to develop, implement, and debug
- Efficiency is tied to knowledge architectural details
- Wide range of architectural differences
- Memory Wall and Power Wall
- Fast evolution
- Many tools by many vendors

A Parallel Programming Model

- An abstraction of a computer system
- Specifies the programmer's view on parallel computer
- Influenced by the architectural design and the language, compiler, or the runtime libraries
  - different parallel programming models for the same architecture
- Implementation of a Parallel Programming Model
  - Libraries
  - Compiler directives
  - Parallel languages

Characteristics of an Ideal Multicore Programming Model

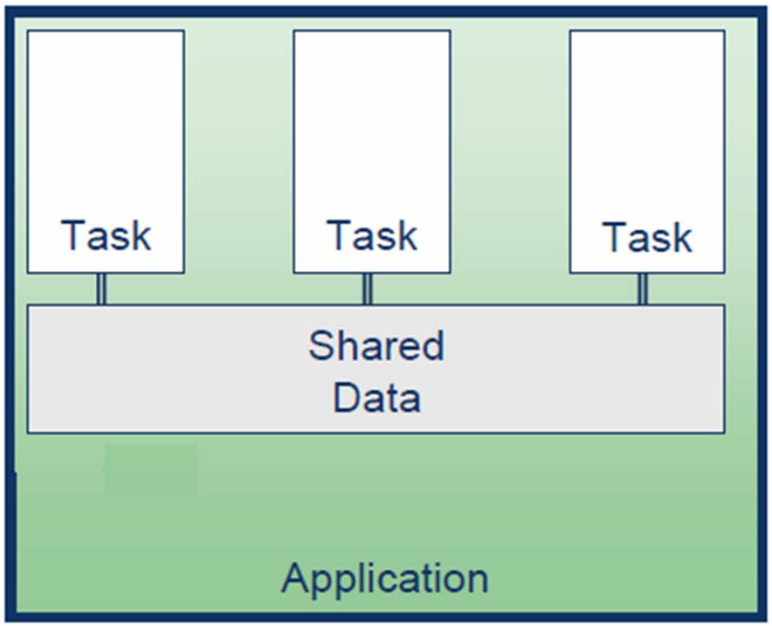
1. Productivity
   - Ease of programming
2. Performance
3. Portability
4. Forward Scalability
   a) Number of compute resource
   b) Diversity of computer resource types

Programming Models as Abstractions of Multicore Systems

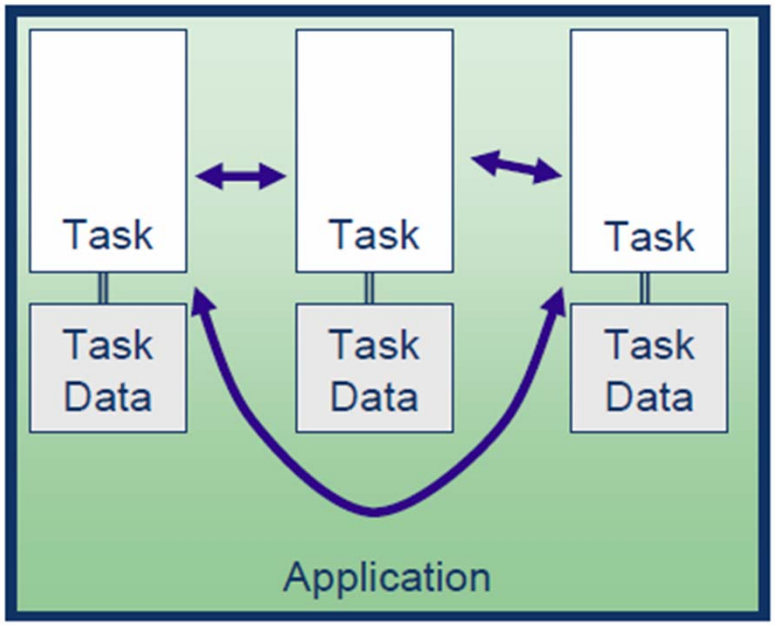
1. Classical parallel programming models
   - Shared Memory
     - Pthreads and OpenMP
   - Message passing
     - MPI
2. Heterogeneous Parallel Programming Models
3. Partitioned Global Address Space (PGAS)
4. Hybrid, shared-distributed memory + GPUs

Shared Memory and Message Passing

Shared Memory



Message Passing



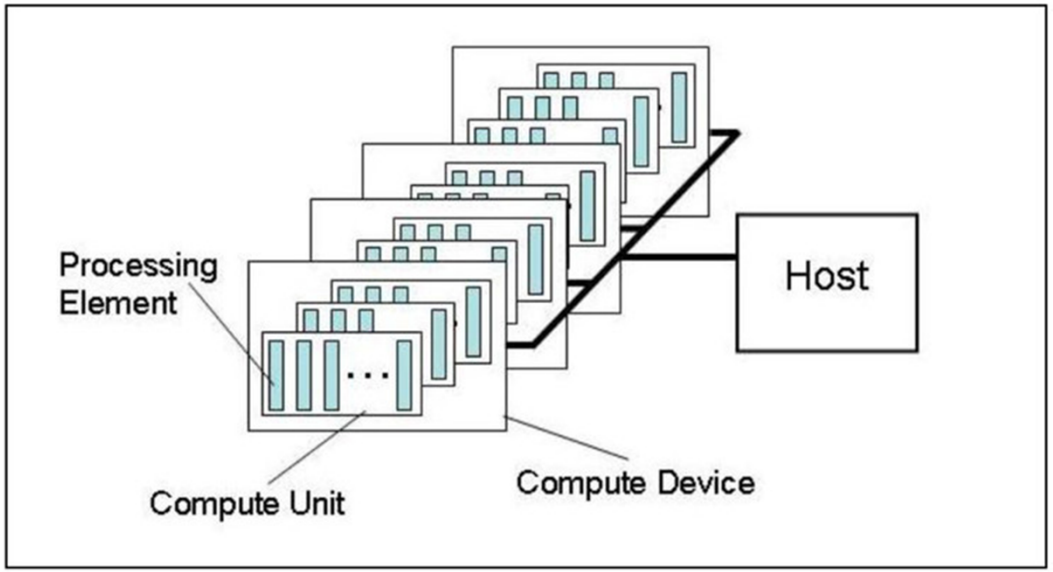
NTNU
Norwegian University of Science and Technology

OpenCL

- An open standard heterogeneous programming model
- Main goals are portability and code reusability across heterogeneous platforms
- Host(CPU) and devices (GPUs, CPUs, …)
- Robust API for doing data parallel and task parallel computation
- Not easy to use

OpenCL Platform Model

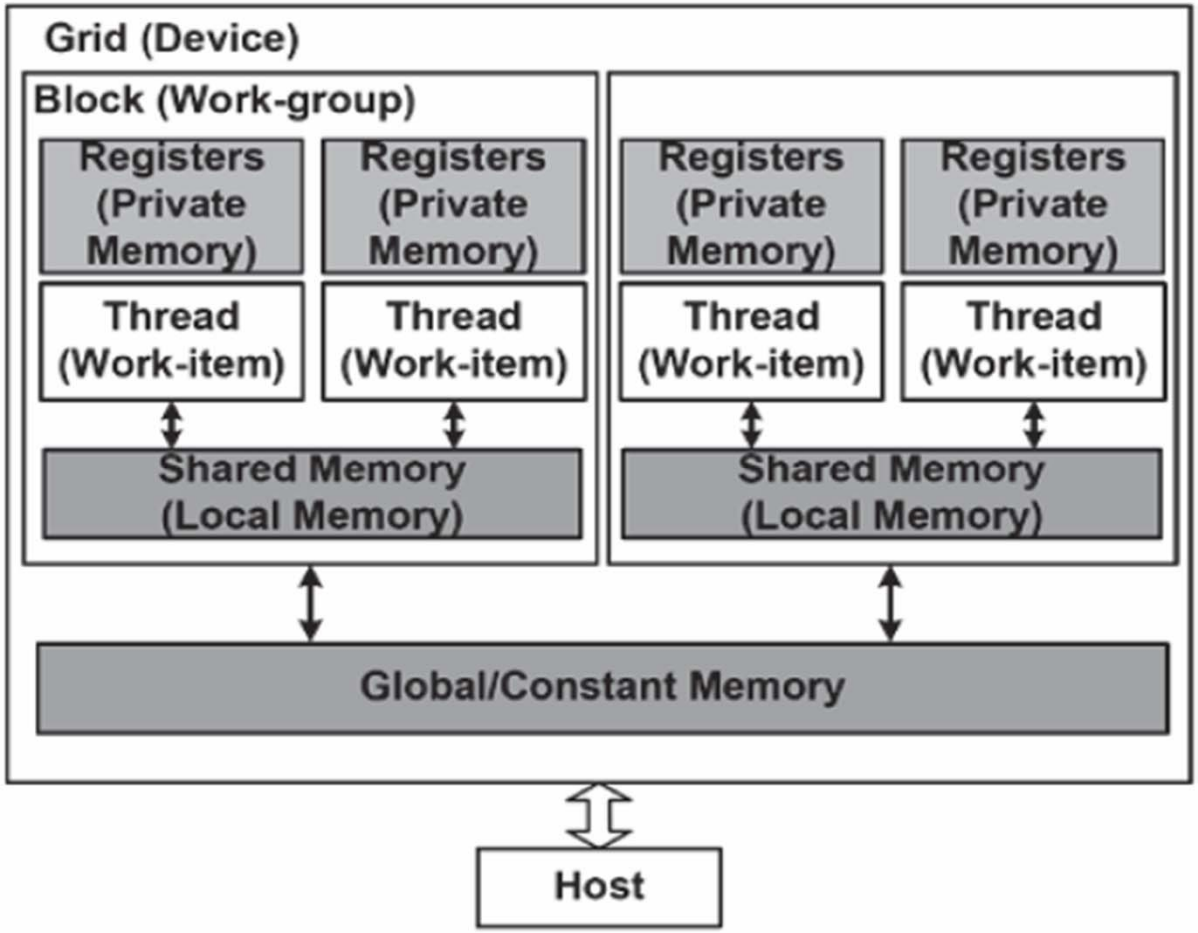
- A host coordinates execution, transferring data to and from an array of Compute Devices.
- Each Compute Device is composed of an array of Compute Units.
- Each Compute Unit is composed of an array of Processing Elements.



OpenCL Execution Model

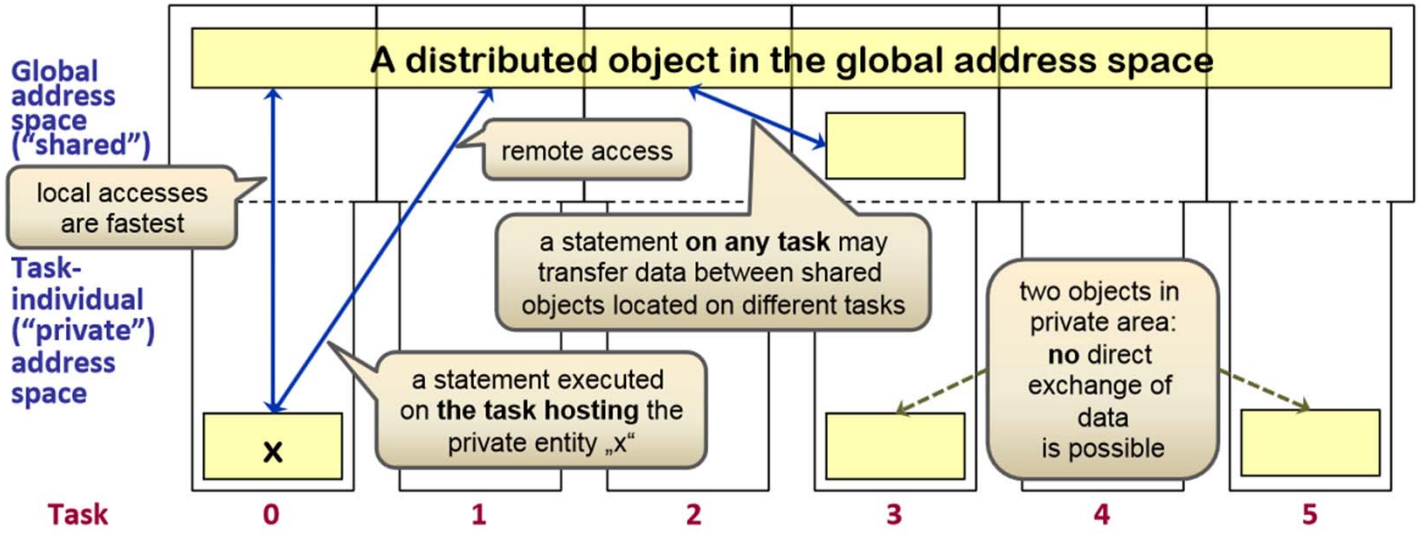
- Simultaneous execution of multiple instances of a kernel on one or more OpenCL devices
- Kernels/(work items) are organized into work groups for communication and synchronization
- The host application enqueues commands to a command queue
  - Kernel execution commands launch work items
- kernels are compiled for the targeted device in runtime
  - The host application exploit all compute devices on the system

OpenCL Memory Model



Partitioned Global Address Space (PGAS)

- Global memory address space that is logically partitioned and a portion of it is local to each process or thread
  - exploiting locality of reference.
- Basis for Unified Parallel C, Coarray Fortran, Fortress, Chapel, X10



Hybrid Parallel Programming Models

- Mixing different programming models as a match to the hybrid hardware architectures
  - Pthreads and MPI (SMP Cluster)
  - MPI and OpenMP (SMP Cluster)
  - CUDA and Pthreads (Multiple GPUs)
  - CUDA and OpenMP
  - CUDA and MPI (GPU cluster)

Programming Models Based on Problem Decomposition

- Data parallelism
  - Splitting data into pieces and letting the processing elements do, in parallel, same take on different pieces
  - Original OpenMP.

- Task Parallelism (MIMD)
  - Decomposing a program into different tasks and execute them on different processing units in parallel.

Data parallelism - Advantages

- Number of parallel processes can be matched to the number of cores
- Can address memory bandwidth bottleneck.
  - Padding and Alignment.
  - Cashing
- Safe
  - Synchronization can be handled by the platform runtime rather than the programmer.
  - no deadlocks or race conditions

Data parallelism - Disadvantages

- Some styles of data-parallel processing perform too little work for each memory access and also are limited in expressiveness
- Challenges in the implementation of control flow in some data parallel programming models

Forms of data parallelism

- SIMD
- Vector Processors
- Stream Processing
  - applies a computationally intense kernel to the input and output streams rather than a single operation
  - Stream processing can achieve high arithmetic intensity

Task Parallelism

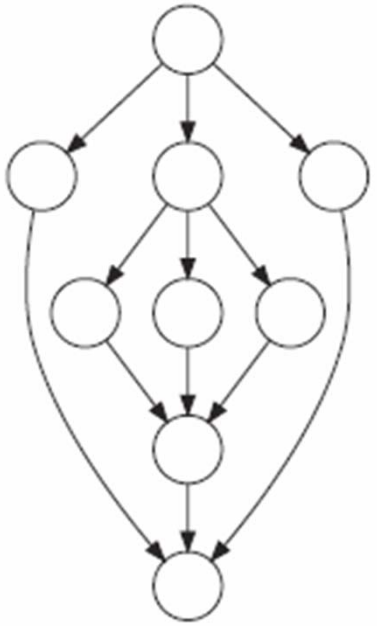
- Different tasks in parallel
- Usually need to communicate
  - Shared memory
  - Message passing
- Usually need to synchronize
  - Shared memory: locks, barriers, … etc.
  - Message passing: synchronous and asynchronous
  - Problems: race conditions, deadlocks, scalability problem
- Need to be scheduled
  - Load balancing

Task-Based Programming Models

- A task is a unit of parallel work
- Used in
  - Adopted by OpenMP 3.0
  - Cilk
  - Intel'sTBB
  - StarSS, OpenSS, … etc
- Task dependencies
  - Control-flow model
  - Dataflow model

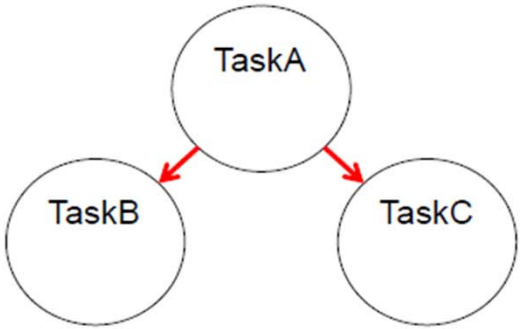
Task Dependencies Based on Control-Flow

- A task create other task(s) and wait for them to finish
- Suited for recursion and divide-conquer algorithm
- Examples: OpenMP 3.0, Cilk, Intel'sTBB



Task Dependencies Based on Dataflow

- Task dependencies is built based on data dependencies
- Task arguments are declared as input, output, or inout
- Task dependency graph
  - directed acyclic graph (DAG)
- More parallelism and less synchronization
- supported SMPSs, OMPSs, OpenMP 4.0




```
#pragma css task output(a)
void TaskA( float a[M][M]);

#pragma css task input(a)
void TaskB( float a[M][M]);

#pragma css task input(a)
void TaskC( float a[M][M]);
```

NTNU
Norwegian University of Science and Technology

Task Scheduling

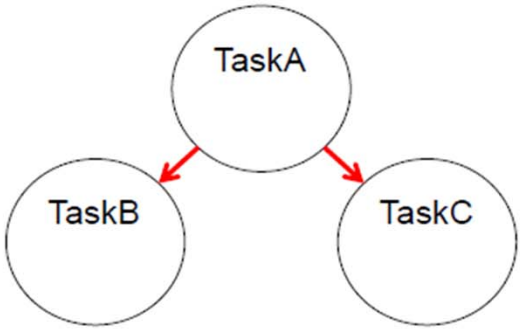
- Goal: minimizes the overall execution time
- Dependencies are represented using a task graph
- Static and dynamic scheduling
  - Assignment to processors at compile time
  - Assignment to processors at runtime
- Load balancing
  - Task pool

OpenMP

- Managed by OpenMP Architecture Review Board (or OpenMP ARB)
- API supports multiplatform using C, C++, and Fortran
- Compiler directives and library routines
- Original OpenMP
  - Data parallelism
- OpenMP 3
  - Support task based approach
  - Dataflow dependency: in|out|inout dependency
- OpenMP 4
  - Support for heterogeneous architectures
  - target construct creates tasks to be executed by accelerators

StarSs and OMPSs

- Source to Source Compiler
- The programmer a serial application with **annotations**
- Dataflow task dependencies
- The runtime system build a data dependency graph




```
#pragma css task output(a)
void TaskA( float a[M][M]);

#pragma css task input(a)
void TaskB( float a[M][M]);

#pragma css task input(a)
void TaskC( float a[M][M]);
```

The StarPU project

- LaBRI-INRIA at university of Bordeaux
- Task-based library written in C
- Portability is a goal
  - Dynamic scheduling
  - Several implementations for each task for different architecture
- Schedule of computations on heterogeneous systems
  - Transfer data between CPU and accelerators
  - Determine which tasks to run on CPU and which to run on accelerators based on efficiency

Cilk

- A parallel programming language
- Task based programming model
- Main construct
  - Spawn (fork)
  - Sync (join)
- Control-flow dependencies
- Work-stealing scheduler

Intel’s Thread Building Blocks(TBB)

- C++ library
- Task-based
- Constructs for parallel loops, reductions, scans, pipeline parallelism
- Control-flow dependency

Further Parallel Programming Approaches

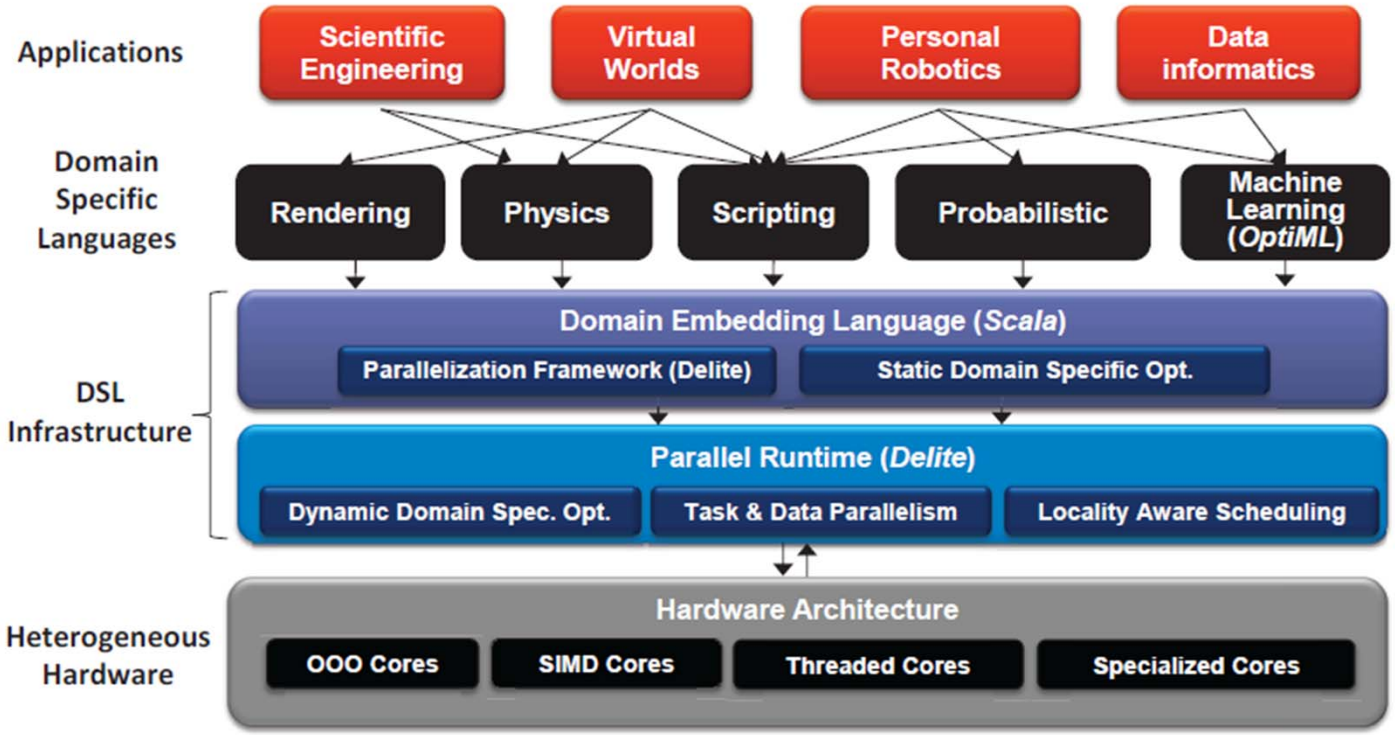
1. Transactional memory
   - Allows a group of load and store instructions to execute in an atomic way
   - A scalable alternative to memory locks
2. Resource aware task scheduling
   - consider resources like memory and bandwidth
3. Domain specific languages
   - Use a *program style that has implicit parallelism*
4. *Pattern Programming Model*
   - defined building blocks out of a Pattern Catalogue

Domain-Specific Approach

- A domain-specific language is a computer programming language of **restricted expressiveness** focused on a **particular domain**.
- Examples: TeX, SQL
- Advantages of using the DSL approach
  - support productivity, portability, and performance goals
  - The ability to use domain knowledge to apply static and dynamic optimizations to a program written using a DSL.

Domain-Specific Approach To Heterogeneous Parallelism

Pervasive Parallelism Laboratory at Stanford University



NTNU
Norwegian University of Science and Technology

Summary

- The goal of a programming model is to provide a mechanism with which the programmer can specify parallel programs.
- There are many different programming models but also open standard models
- There is a high interest in task-based programming
- Domain-specific approach represents a chance implicit parallelism.

References

- Chafi, H., A. K. Sujeeth, K. J. Brown, H. Lee, A. R. Atreya and K. Olukotun (2011). "A domain-specific approach to heterogeneous parallelism." SIGPLAN Not. **46**(8): 35-46.
- Jahr, R., M. Gerdes and T. Ungerer (2013). A pattern-supported parallelization approach. Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores. Shenzhen, Guangdong, China, ACM: 53-62.
- Javier, D. (2012). "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era." IEEE Transactions on Parallel and Distributed Systems 23(8): 1369-1386.
- Project, S. E. (2010). "Parallel Programming Models for Heterogeneous Multicore Architectures." Micro, IEEE **30**(5): 42-53.
- d. Augonnet, S. Thibault, R. Namyst, Pierre-Andr, #x00e9 and Wacrenier (2011). "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures." Concurr. Comput. : Pract. Exper. **23**(2): 187-198.
- Vandierendonck, H., P. Pratikakis and D. S. Nikolopoulos (2011). Parallel programming of general-purpose programs using task-based programming models. Proceedings of the 3rd USENIX conference on Hot topic in parallelism. Berkeley, CA, USENIX Association**:** 13-13.
- Petrides, P. and P. Trancoso (2013). Addressing the challenges of future large-scale many-core architectures. Proceedings of the ACM International Conference on Computing Frontiers. Ischia, Italy, ACM**:** 1-4.

NTNU
Norwegian University of Science and Technology