MCPA—MultiCore Portability Abstraction

Martti Forsell
Platform Architectures Team
VTT
Oulu, Finland
Martti.Forsell@VTT.Fi

MP-SOC Forum’11
July 6, 2011
Beaune, France

FOR K:=0 TO [log N]-1 DO
  FOR J:=2^K+1 TO N PARDO Table[J]:=Table[J-2^K]+Table[J];
MCPA — MultiCore Portability Abstraction

Martti Forsell, Chief Research Scientist, VTT (Technical Research Center of Finland)

Abstract: Application portability between different architecture-paradigm/programming tool pairs for MP-SOCs is a big problem nowadays leading often to a complete rewrite of an application when switching from an architecture-paradigm pair to another. This is caused by a wide variety of architectural properties requiring different optimization techniques for different architectures, typically hiding the essence of parallel computing defined by the application.

In this presentation, we introduce the MultiCore Portability Abstraction (MCPA) simplifying portability and implementation of parallel applications. It abstracts away typical architecture dependent effects caused by latency, synchronization, and partitioning and acts as an executable intermediate abstraction/reference implementation as well as a tool for analyzing the intrinsic parallelism of the application and relative goodness of architectures in executing it. We give a short application example with performance measurements.

Interestingly, the MCPA appears to be architecturally directly implementable via our advanced configurable emulated shared memory architecture (CESM), which we are currently prototyping in our recently launched REPLICA project. If successful, this promises to simplify MP-SOC application programming radically.
Problem: MP-SOC application portability

Weak application **portability** between different architecture-paradigm/programming tool pairs for MP-SOCs is a **big problem** nowadays.

This **leads often to a need for complete rewriting** of an application when switching from an architecture-paradigm pair to another.

**GP-GPU/Cuda**

**x86/PThreads**

**The reason:** Different optimization techniques are applied for different architectures, which typically **hides the intrinsic parallelism** of the application from programmers.

Unfortunately the more optimized the architecture is for certain application the bigger the risk is!
MCPA—MultiCore Portability Abstraction
A shared memory-based abstraction to improve portability and simplify parallel implementation
—Natural extension of the model of sequential computation

The first model of computation that comes into the mind of a programmer as he starts to think
how to solve a computational problem in parallel
—abstracts away latency, synchronization cost and data partitioning effects (like its counterpart)
Overview of the MCPA

Paralle programming *techniques*, parallel algorithm *theory*

Architecture-paradigm specific optimization, *guidelines*

**Abstraction**

- Computational problems (very often parallel)
- Sequential legacy code
- Direct architecture-paradigm specific implementation

**MCPA acts as**
- executable parallel reference implementation
- tool for analyzing the intrinsic parallelism of the application and goodness of the architectures
- intermediate model for simplifying implementation and portability

**Architecture paradigm pairs**

- A1,P1
- A1,P3
- A7,P2
- A19,P8
- A17,P11
- A22,P17
MCPA

- Works with different parallel algorithms from sequential (weakest alternative) to fine-grained parallel (most beneficial)
- Helps to analyze how parallel the application is
- Simplifies portability between architecture & paradigm pairs with respect to direct implementation without the abstraction
- Provides simplest programmability
- Helps architecture and paradigm selection
- Provides simple guidelines for optimizing the functionality for architecture-paradigm pairs (assuming they are supported by MCPA)
Natural MCPA-assisted functionality design flow (first outline)

1. **Computational problem (Functionality)**
   - Easy programming
   - Native MCPA version (Natural parallelism)
     - MCPA execution, T=N, invalid/very slow in SMP, NUMA, CC-NUMA, VC, fast/full speed in ESM
     - Add synchronizations etc. to ensure correctness
     - MCPA version modified for the computational model of the target architecture
       - Native execution, T=N, typically badly suboptimal in SMP, NUMA, CC-NUMA, VC, obsolescent for ESM
       - Optimize using the guidelines
         - Architecture optimized version (Architecture dependent design)
           - Native execution, T<<N, best performance in SMP, NUMA, CC-NUMA, work-optimal

2. **Architecture dependency**
   - If this is used as a starting point a rewrite can not be avoided
Examples of guidelines (rough, very early version)

1. Match the #SW threads with #HW threads
2. Synchronize with explicit barriers
3. Minimize the number of synchronizations by reorganizing computation, e.g. with blocking

Intel Core2 Duo SMP & PThreads

MCPA

1. Match the #SW threads with #HW threads
2. Synchronize with explicit barriers
3. Minimize the number of synchronizations by reorganizing computation, e.g. with blocking
4. Maximize locality by locating data needed by a core next to it

4/16/64-NUMA & e-language

4/16/64-ESM & e-language

Guidelines deal with synchronization, mapping, partitioning, blocking, hashing, scheduling, ...
Core2 Duo & PThreads
Optimized (blocking), not unrolled

2x2-core XEON & PThreads
Optimized (blocking), not unrolled

Core2 Duo SMP & PThreads
Optimized (blocking), not unrolled

SMP/NUMA/ESM comparison
Optimized (blocking, localization), not unrolled

Early example: PREFIX sum
Early example: PREFIX sum

![SMP/NUMA/ESM code size diagram]

A horror story—How the first attempt can lead to a complete disaster in performance

We used the standard text-book logarithmic prefix sum algorithm O(log n), made it work on our Core2 Duo SMP & PThreads with explicit barriers for 16 threads.

The resulting program executed **11 000 000 times slower** than the sequential one on Core2 Duo SMP & PThreads although it works as predicted in ESM & e.
Architectural implementability?!

Interestingly, the MCPA appears to be **architecturally directly implementable** via the advanced configurable emulated shared memory architecture (CESM), which we are currently investigating:

The **REPLICA** project of VTT aims developing CESM and methodology that would enable radically **easier programming** and **higher performance** with a help of the PRAM model of computing.

A proof of concept prototype will be built!
REPLICA = Removing Performance and Programmability Limitations of Chip Multiprocessor Architectures

A 3-year Frontier research project funded entirely by VIT

Funding: 500 000 €/year, in total 1 500 000 €

Amount of work: 129 pm, duration 3 year

Target business: Companies that design or manufacture general purpose and application-specific CMPs or develop software/functionality for them

Novel techniques:
- Latency hiding
- Efficient wave synchronization
- Concurrent memory access
- Multioperations
- Virtual ILP exploitation
- Pipeline hazard elimination
- Memory hashing
Conclusions

To address **portability** problems between different MP-SOC architecture-paradigm/tool pairs and to simplify **overall parallel implementation** of the functionality, we have introduced **MultiCore Portability Abstraction** (MCPA) that provides

- an executable **intermediate computational model** that抽象s away latency, synchronization cost and data partitioning effects
- simple **guidelines for optimizing** the application of certain architecture-paradigm/tool pairs
- **means to analyze how parallel** the application is and how good the architecture is for the application

MCPA appears to be directly implementable promising easier programmability in the future. We are **building** an **MCPA architecture prototype** in REPLICA.