PEPPHER

PEPPHER
Vision & Overview

PEPPHER Workshop @ HIPEAC 2011
Heraklion, Crete, Greece, January 22, 2011

Siegfried Benkner (on behalf of PEPPHER consortium)

University of Vienna

This project is part of the portfolio of the
G.3 - Embedded Systems and Control Unit
Information Society and Media Directorate-General
European Commission

www.peppher.eu

Contract Number: 248481
Starting Date: 2010-01-01
Duration: 36 months

PEPPHER

EU Project PEPPHER

Performance Portability & Programmability for Heterogeneous Manycore Architectures

- EU ICT Call 4, Computing Systems; Start: Jan. 2010, 36 Months;
- Coordinated by University of Vienna

Goal: Enable portable, productive and efficient programming of heterogeneous manycore systems.

Holistic Approach

- Higher-Level Support for Parallel Program Development
- Auto-tuned Algorithms & Data Structures
- Compilation Stragtegies
- Runtime Systems
- Hardware Mechanisms

Focus:
Single node architectures

Crosscutting Domains: Embedded – General Purpose – HPC

Project Consortium

- University of Vienna (Coordinator), Austria
  Siegfried Benkner, Sabri Pllana and Jesper Larsson Träff
- Chalmers University, Sweden
  Philippas Tsigas
- Codeplay Software Ltd., UK
  Andrew Richards
- INRIA, France
  Raymond Namyst
- Intel GmbH, Germany
  Herbert Cornelius
- Linköping University, Sweden
  Christoph Kessler
- Movidius Ltd., Ireland
  David Moloney
- Karlsruhe Institute of Technology, Germany
  Peter Sanders

Heterogeneous MC Architectures

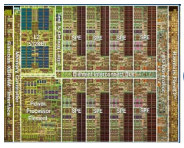
Move towards heterogeneous manycore architectures

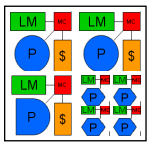
- Different types of cores on a chip
- Same cores but different clock frequencies
- Partitioning groups of homogeneous cores, …

→ Parallelization & Specialization

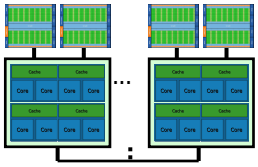
Examples

- Cell Processor: PPU + 8 SPUs
- SARC Research Processor
- CPU + GPU/Accelerators
- Tianhe-1A, Roadrunner, TSUBAME
- Nvidia Tegra, AMD Fusion, SandyBridge


Cell BE



SARC


GPU Cluster

Programming Heterogeneous Multicore

PEPPHER

Much harder than for homogeneous systems

- Need of allocating and managing resources
- Explicit memory management (DMA transfers, double buffering, …)
- Partitioning of code for different cores
- Different memory models, ISAs, compilers, programming models

Performance Portability is a major challenge

- Increasing architectural diversity
- Compilers can't keep pace with shorter and shorter innovation cycles

Current Solutions?

- IBM CellSDK, NVIDIA CUDA, ATI Stream SDK, OpenCL, …
- Intel TBB/ArBB, HMPP, PGI Accelerator, StarSs, OpenMP 4.0?…
- Codeplay Offload++

Programming Heterogeneous Multicore

PEPPHER

Will we need new programming models?

- No „one-size-fits-all" model
- Need to integrate different models

Programmability/Productivity

- Raise level of abstraction
- Hide/Automate low-level optimization tasks

Portability of major importance

- Increasing complexity of architectures
- Increasing architectural diversity

Performance Portability

- Consider different aspects not just FLOPs
- Energy/Power as important as performance

PEPPHER

Compositional Approach

Adaptation
Auto-Tuning
Algorithmic Choice

Abstract Hardware Models

Abstract Performance Models

Scope and Focus of PEPPHER

Goal

Methodolgy for development of performance portable code from (existing) components by stepwise annotation.

Component Model & Prototype Framework

- Performance-aware components with rich meta-data
- Meta-data and platform description language
- Annotation framework
- Advanced Runtime System for heterogeneous architectures

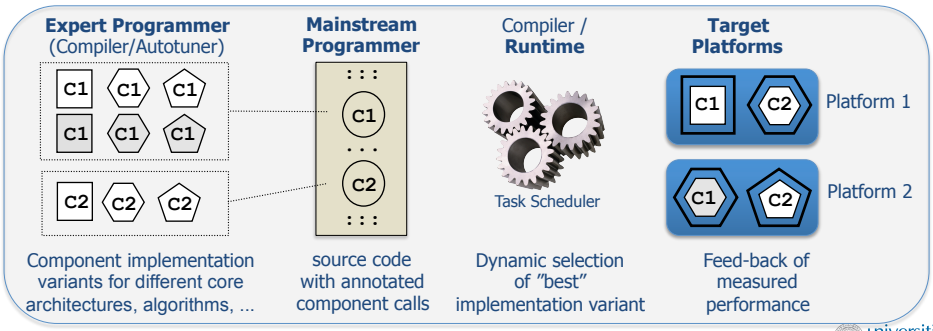
Non-Goals

- Parallelization per se
- New programming model or language
- New compiler
- New hardware

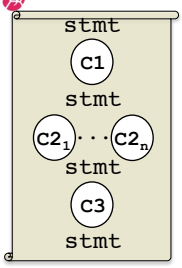
PEPPHER Vision

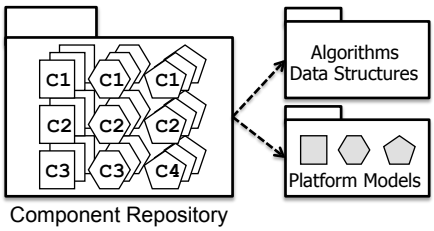
Compositional Approach to Parallel Software

- Components that hide implementation specific details (Mainstream Programmer)
- Component implementations variants (Expert Programmer, Autotuning)
- Performance-aware components based on **rich meta-data**
- Smart runtime system to select best component implementation variant



PEPPHER Application Development





Component Repository

Mainstream programmer

- Builds application using components
- Annotates component calls
  (e.g. expresses performance constraints)
- High-Level coordination mechanisms

Expert programmer

- Implements component variants for specific architectures (class)
- Algorithm & Data Structure Library
- Provides performance models, meta-data

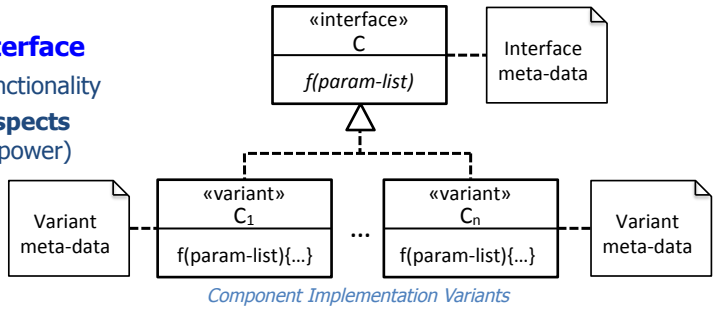
(Implicit) Tasking Model

- PEPPHER components executed by tasks
- Multi-Level Parallelism (components may be parallel inside)
- Runtime Component Variant Selection

PEPPHER Components

Component Interface

- Declaration of functionality
- **Performance aspects** (execution-time, power)



Component Implementation Variants

Implementation Variants

- Different architectures/platforms
- Different algorithms
- Different input characteristics
- Different performance goals
- Written by expert programmer or generated (auto-tuning)

Features

- Different programming languages (C/C++, OpenCL, Cuda, OpenMP)
- Task & Data parallelism
- Performance-Awareness

Constraints

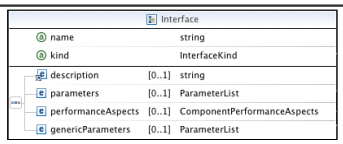
- Stateless; Non-preemptive
- Composition on CPU only

Component Meta-Data

PEPPHER

Interface Meta-Data (XML)

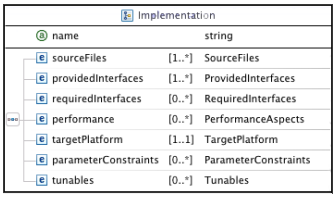
- Parameter intent (read/write)
- Supported performance apsects (execution-time, power)



XML Schema for Interface Meta-Data

Implementation Variant Meta-Data (XML)

- Required components (if any)
- Supported target platforms (PDL)
- Performance Model
- Input data constraints (if any)
- Tunable parameters (if any)



XML Schema for Variant Meta-Data

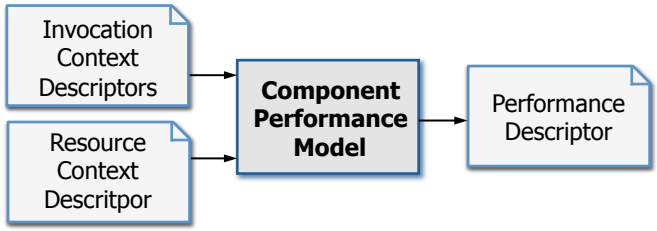
Key issues

- Make platform specific optimizations/dependencies explicit.
- Make components performance and resource aware.
- Support runtime variant selection.
- Support code transformation and autotuning.

Performance-Aware Components

PEPPHER

Each component is associated with an **abstract performance model.**



☐ **Invocation Context:** captures performance-relevant information of input data (problem size, data layout, etc.)

☐ **Resource Context:** specifies main HW/SW characteristics (cores, memory, …)

☐ **Performance Descriptor:** usually includes (relative) runtime, power estimates

Generic performance prediction function:

```
PerfDsc getPrediction(InvocationContextDsc icd, ResourceContextDsc rcd)
```

Architecture/Platform Models

XML Platform Description Language (PDL)
to capture relevant aspects of heterogeneous architectures at ***different abstraction levels***.

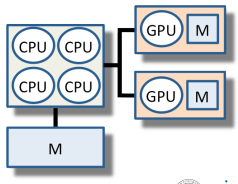
PDL comprises

- Processing Units (master, worker, hybrid)
- Memory Regions
- Interconnects

Platform descriptors are used by

- Expert programmers
- Compiler / Autotuner
- Runtime system

```
<Master id="0"quantity="4">
 <PUDescriptor>
  <Property fixed="true"
   <name>ARCHITECTURE</name>
   <value>x86</value>
  </Property>
  ...
 </PUDescriptor>
 <Worker quantity="2" id="1">
  <PUDescriptor>
   <Property fixed="true">
    <name>ARCHITECTURE</name>
    <value>gpu</value>
   </Property>
  </PUDescriptor>
 </Worker>
 <Interconnect type="rDMA"
  from="parent" to="1" scheme="unicast"/>
</Master>
```

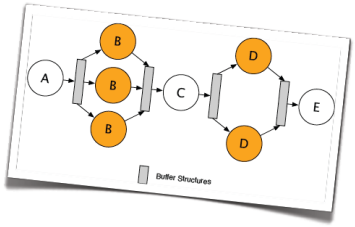


Tunable Patterns

Support for parallel design patterns

- Master/worker, pipelining, ...

Annotations for specifying tuning points

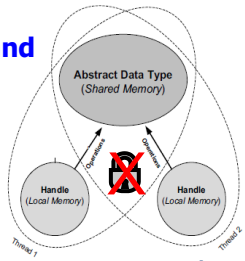


Example: Pipeline pattern

```
@pipeline[exit_criterion] with buffer[fifo] {
   @stage
   stage_A(...)
   @stage replication <count>   // replicates the stage <count> times
   stage_B(...)
   @stage buffer [priority]     // change buffer to priority, to keep ordering
   stage_C(...)
   ...
}
```

Adaptive Data-Structures & Algorithms

- Toolbox of **non-blocking shared data structures and parallel algorithms** written by expert programmers.
  e.g: stack, queue, list; sort, suffix, skyline, …



- **Parameterize** algorithms and data structures **with platform models** (PDL) (e.g. degree of parallelism, memory usage, …)

- Investigate static and dynamic **adaptation and auto-tuning methods.**
  Specialization/optimization for specific platforms/inputs/optimization goals.

Based on existing work:

- Parallel C++ STL for GNU compiler (Karlsruhe)
- Lock-Free Shared Abstract Data Types (Chalmers)

PEPPHER Runtime System

Heterogenous Runtime System (based on INRIA's StarPU)

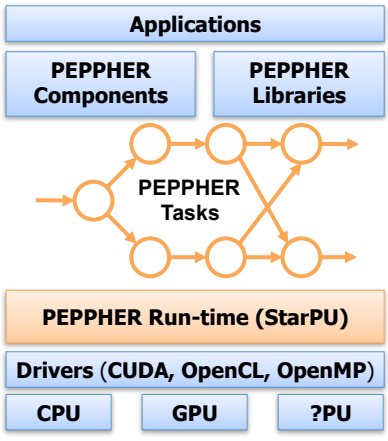
- Selection of component variants based on available hardware resources
- Data-aware & performance-aware task scheduling onto heterogeneous PUs

Tasks
- Explicit dependencies with other tasks
- Multiple implementations (GPU, CPU)

Automatic data transfer
- Virtual Shared Memory (VSM) layer
- Minimize data transfers btw. PUs

Flexible scheduling strategies
- Performance-aware
- Scheduling algorithm = plug-in

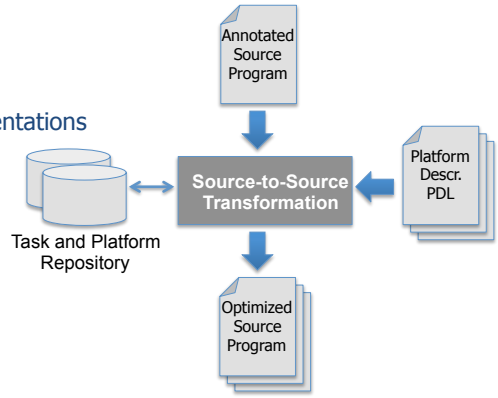
Performance Feed-back



Transformation / Compilation

PEPPHER Transformation System

- Generation of glue-code for runtime component selection.
- Pre-selection of component implementations based on platform models (PDL).
- Generation of parallel tasks from data distribution annotations.
- Optimization of data management.



Compilation of Offload++ to OpenCL (Codeplay)

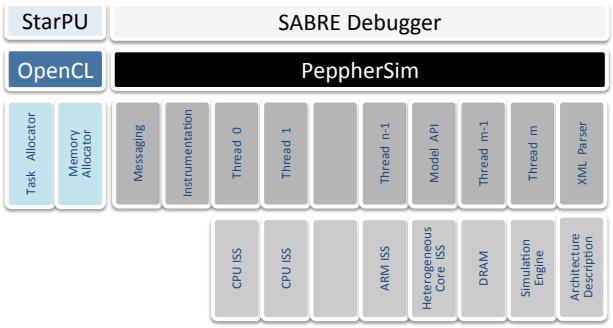
- Offload component variants

Hardware-related Research Issues

Investigate HW-support mechanisms for

- performance tuning,
- data-structures and synchronization,
- memory management and utilization,
- runtime scheduling.

Simulation and experimental evaluation on prototype board (Movidius).



PEPPHER Benchmark Set

Performance Benchmark SET	x86 Multi-Core	GPU	Cell	Owner
Enterprise/General Purpose				
Suffix Array Construction	x	x		KIT
AISandbox	x			Codeplay
High Performance Computing				
GROMACS (Life-Science)	x	x		Intel
Embedded/Gaming/Multi-Media				
Computational Photography	x	x	x	Movidius
x264 (codec)	x	x		UNIVIE
Kernels				
PLASMA/MAGMA (LU/Cholesky/QR) Matrix Decompositions	x	x		INRIA
RODINIA (CFD Solver)	x	x		INRIA
FFTW - Spectral Analysis	x	x		INRIA
STL Libraries	x	partly		KIT

Related Research Efforts

- HiPEAC NOE (Europe)
- ParLab (Berkeley)
- Pervasive Parallelism Laboratory (Stanford)
- CellSs (Barcelona)
- Sequoia (Stanford)
- PetaBricks (MIT)
- Elastic Computing (Univ. Florida)
- ... and many more ...

Conclusion

Paradigm shift to (heterogeneous) manycore architectures leads to huge research challenges.

- Raise level of abstraction in parallel programming
- Performance Portability of major concern
- Memory management (locality) most critical issue in future architecture
- Auto-tuning & adaptivity to fight architectural complexity

PEPPHER proposes:

- **Component-based approach** combining different programming models/APIs
- **Performance-aware components** with explicit platform dependencies **(PDL)**
- **Smart runtime** system
- Adaptive/auto-tuned algorithms and data structures
- Investigations of HW mechanism to support performance

Acknowledgments

- ☐ European Commission
- ☐ HiPEAC Consortium
- ☐ PEPPHER Consortium

Some of the consortium members (from left): D. Moloney, E. Marth, S. Pllana, V. Osipov, M. Wimmer, B. Bachmayer, P. Tsigas, J.L. Träff, C. Kessler, J. Singler, S. Benkner, D. Cederman, U. Dastgeer, H. Cornelius, S. Thibault, A. Richards, M. Sandrieser, U. Dolinsky, R. Namyst, C. Augonnet, H.C. Hoppe