Insieme

Insieme - an Optimization System for OpenMP, MPI and OpenCL Programs

Institute of Computer Science
University of Innsbruck

Thomas Fahringer, Ivan Grasso, Klaus Kofler, Herbert Jordan, Hans Moritsch, Simone Pellegrini, Radu Prodan, Heiko Studt, Peter Thoman, John Thomson

TU Munich 2011-03-30
High Performance Parallel & Distributed Computing

Our Research

**Cloud Computing**
- Hardware and Software Virtualisation
- Performance Modelling and Analysis
- Quality of Service
- Multi-criteria Scheduling
- Service Level Agreements

**Grid Computing**
- Programming Paradigms and Methods
- Meta Scheduling
- Resource Brokerage
- Performance Measurement, Analysis, and Prediction
- Ontologies for the Grid

**Parallel Processing**
- Programming Paradigms
- Performance Instrumentation and Measurement
- Performance Analysis and Interpretation
- Performance Prediction
- Compiler Analysis and Optimisation
- Experiment Management
Managing power dissipation is limiting clock speed increases

Source: Michael Perrone, IBM
Hardware Architecture Evolution

Constrained by:
• Power
• Complexity

Enabled by:
• Abundant data parallelism
• Power-efficient GPUs

Limited by:
• Programming models

Hardware Architecture Evolution

Constrained by:
• Power
• Complexity

Enabled by:
• Abundant data parallelism
• Power-efficient GPUs

Limited by:
• Programming models

Future Hardware Diversity

Intel:
- “Single chip cloud computer”
- 24 dual-core tiles
- Mesh interconnect

Nvidia:
- “Fermi” GPGPU
- 512 CUDA cores
- Configurable L1 cache / scratchpad

AMD:
- “Fusion” combines GPU and CPU
- 4 CPU cores
- 480 Stream processors
Parallel Processing: Past and Future

- Parallel Processing has long been an essential component of scientific computing that drives natural and technical sciences.
- Parallel Processing appears to be merging with
  - embedded systems
  - multi-media and entertainment
  - reliable systems
  - and more to come ...
- Different application domains require different parameters to be optimized:
  - performance
  - cost
  - energy
  - reliability, etc.
- This makes HPC a multi-parameter optimization problem
The Multicore Software Problem

- There is more than 1 million software engineers and programmers working in the EU
- A negligible fraction know how to program parallel computers.
- Enormous legacy investment in serial programming technology and training.

“[Multicore] could become the biggest software remediation task of this decade.”

-- Gartner Group, January 31, 2007
Current/Future Many-core Architectures

Heterogeneous cores running at different speed

Homogeneous bus attached

Heterogeneous bus attached

IO bus attached

Network attached

Source: Michael Perrone, IBM
Why is it so hard to optimize codes for parallel systems?

• Question:
  – If the strategy for I/O scheduling, process scheduling, cache replacement policy would be changed, how would you re-write your code?

• Complexity, undecidability and difficulty to predict program and system behavior:
  – Dynamic reallocation of cores, memory, clock frequency; external load, sharing of resources, etc.
  – Processor and system architectures are so complex that it is impossible for a human being to find best code transformation sequences
  – Operating system, external load, queuing systems, caches often have non-deterministic behavior
Example: ADI Solver (Alternating Direction Implicit)

Sequential Algorithm

Phase 1

Phase 2

Parallelization Strategies

OpenMP

MPI

MPIMessage
What is the optimal number of cores to use?

- Performance impact: CPU architecture, cache size and memory hierarchy
- Ideal number of threads requires knowledge about the program and architecture.
ADI/MPI Comparison

- Data is block-wise distributed onto set of MPI processes
- \((N, M) \rightarrow N\) row and \(M\) column block distribution

**Total of 32 cores**

SMP node with 8 AMD quad-core (Barcelona) CPUs
- \((1,32)\)
- \((2,16)\)
- \((4,8)\)
- \((8,4)\)
- \((16,2)\)
- \((32,1)\)

**Total of 48 cores**

4 SMP nodes with 2 AMD six-core (Istanbul) CPUs each
- \((1,48)\)
- \((2,24)\)
- \((3,16)\)
- \((4,12)\)
- \((6,8)\)
- \((8,6)\)
- \((12,4)\)
- \((16,3)\)
- \((24,2)\)
- \((48,1)\)

**Total of 64 threads**

4 SMP nodes with 2 Intel quad-core (Nehalem) CPUs each (2-fold SMT)
- \((1,64)\)
- \((2,32)\)
- \((4,16)\)
- \((8,8)\)
- \((16,4)\)
- \((32,2)\)
- \((64,1)\)

TU Munich 2011-03-30
ADIMPI Message Strip Mining

Message strip mining enables computation pipelining for increased parallelism.

**Phase 1**
- Tile size
- $T$
- $P_0$, $P_1$, $P_2$, $P_3$

**Phase 2**
- $T$
- $P_0$, $P_1$, $P_2$, $P_3$

**Question**
What is the *optimal* tile size $T$?
The optimal tile size for a “good” data layout depends on underlying architecture, program, problem size, etc.
The Insieme System

- A multi-parameter optimizing Compiler for MPI, OpenMP and OpenCL
  - Optimization across multi-parameters:
    - performance, cost, energy consumption, reliability, etc.
  - Sources of optimization
    - program structure (transformations)
    - runtime environment parameters
  - Analysis and optimization
    - static and dynamic analysis for entire program and code regions
    - based on historic date: executions of training kernels and applications
    - uses machine learning to deal with huge search space for combinations of optimizations

- Insieme is currently under development at the University of Innsbruck
Machine Learning based Optimization

We propose the empirical model:

- acquire optimization knowledge by learning from examples
- apply a large number of transformations to benchmark suites to generate code versions
- measure performance, energy consumption, cost, reliability, etc. for each code version and store in repository
- describe programs and its regions through program features
- Use machine learning to accurately model the system
- Deliver the final “trained machine”

Source: I. Guyon “Introduction to ML”
Machine Learning based Optimization

- For each input program, the trained machine is queried to determine effective
  - transformation sequence for each program region
  - parameter setting for runtime environment for a given machine and system status - depends on input data

- Advantages
  - works for changing platforms
  - no hard-wired heuristics that are soon out of date
  - always based on evidence

Source: I. Guyon “Introduction to ML“
Performance Models to Drive Optimization

- How to describe a parallel programs in a way which is useful for machine learning?
- We need to describe programs in terms of characteristics (program features) that define similarity, e.g.: control and data flow information, number of operations, cache misses, communication patterns, volume of data exchanged, ...
- Programs with similar features are likely to have a similar behavior
k-nearest neighbors algorithm (k-NN):

• We need to match our new unseen program to previously seen and recorded programs to determine how to optimize

• Nearest neighbors determines the classification of our new program by measuring the distance in the feature space between the new program and all others

• We predict the new program shares the characteristics of its nearest neighbor
Insieme Training Phase

- **OpenMP, MPI, OpenCL**
  - **Training programs**
    - 1
    - 2
    - 3
  - **Transformations (Static Optimizations)**
  - **Runtime Optimizations**
  - **Program Versions**
    - 1
    - 1.1
    - 1.2
    - 2
    - 2.1

- **Feature Extraction**

- **Training Data**
  - Program Features
  - Transformation Sequence
  - Input Data Features
  - Runtime Parameters
  - Architecture Features
  - Exogenous Variables
  - Execution State
  - Performance Metrics
  - Cost Metrics
  - Energy Metrics

Profiling on different architectures

TU Munich 2011-03-30
Insieeme Optimization Phase

Training Data
- Program Features
- Transformation Sequence
- Input Data Features
- Runtime Parameters
- Architecture Features
- Exogenous Variables
- Execution State
- Performance Metrics
- Cost Metrics
- Energy Metrics

OpenMP, MPI, OpenCL
Target architecture

External load, system load, etc.

Features

INSIEME Compiler
- Optimal Transformation Sequence
- Source-to-source Translation
- Optimal Runtime Parameter Settings

INSIEME Runtime

Optimized program
Insieme Architecture Overview
Insieme Parallel Intermediate Representation - InsPIRe

- Unified Representation of Parallel Programs
  - structural type system
  - closed set of generic types and operators
- Minimal language core
- Explicit Parallelism
- Language level synchronization / communication
- Extendable through composability
- Core module offers
  - data structures to represent programs and annotations
  - manipulation tools
C Input:

```c
int main(int argc, char* argv[]) {
    int a;
    for(int i=0; i<10; i++) {
        a += i;
    }
}
```

InsPIRe:

```c
fun(int<4> v1, array<ref<array<ref<char>,1>>,1>,1> v2) {
    decl ref<int<4>> v3 = var(0);
    for(decl ref<int<4>> v4 = var(0) .. 10 : 1) {
        v3 := v3+v4;
    }
}
```
InsPIRe Abstract Syntax Tree

Multiple references: 90% memory reduction

XML export/import
Frontend

- Translates input program into InsPIRe - AST
- Capable of supporting hybrid code
- Two steps
  - Step1: C/C++ => IR (syntax)
  - Step2: eliminate MPI / OMP/ OpenCL (semantics)
- `clang` for parsing input (step 1)
- InsPIRe module for manipulations (step 2)
Optimizer

• High Level Transformations
• Pattern recognition
• High-level semantic optimizations
  – e.g. optimized use of arrays/sets/lists exploiting operator semantics
• Loop transformations
• Parallelization / Vectorization
• Integration of high-level knobs
  – e.g. selection of algorithms, data representation
Synthesizer

- „Simple“ Backend (first prototype)
- Pure MPI Backend
- Insieme Runtime Backend
- Target specific synthesizers
  - shared memory
  - distributed memory
  - accelerators
  - integration of target specific knobs
  - e.g. scheduling policies, communication protocols, group sizes, thresholds for parallelism
Insieme Runtime

• Runtime Library
  – called by target code
  – target specific extensions (MPI, OpenCL,...)

• Runtime Environment
  – tuning of runtime parameters (knobs)
  – resource management (cores, nodes, accelerators, ...)

TU Munich 2011-03-30
Case Study: OpenMP Benchmarks

Achievable speedup is limited

Machine: 8 quadcore AMD CPUs (Sun X4600 M2)
Multiple OpenMP applications with different job scheduling strategies

different strategies of reducing the number of threads assigned to each application

TU Munich 2011-03-30
Insieme OpenMP job scheduling

- For each region, optimal thread count is dynamically determined
- Optimization options:
  - locality: increase locality of threads assigned to the same application
  - clustering: clusters of cores should be used by single applications

![Bar chart showing total time (s) for different scheduling options: Sequential, OS parallel scheduling, maximum 2, Insieme scheduling, Insieme scheduling with locality, Insieme scheduling with locality plus clustering. 67% of sequential execution time highlighted.]

TU Munich 2011-03-30
Automatic Tuning of MPI Runtime Parameters

- MPI implementations allow for tuning the runtime environment to better fit the underlying architecture, such as:
  - eager/rendezvous send threshold:
    - use eager or the rendezvous protocol depending on messages size
  - processor affinity flag:
    - bind an MPI process rank to a physical core
- Open MPI's Modular Component Architecture (MCA) provides 100’s of parameters
Effects of MPI Runtime Parameter Tuning

FT, CG, IS and EP from NAS Parallel Benchmarks running on a cluster of SMPs nodes, using 8 vs. 32 nodes

wrt. Open MPI default settings
Using Machine Learning to Predict Optimal Parameter Settings

- Performance of predicted parameter setting, relative to best performance found during exploration, using two learning algorithms:
  - Artificial Neural Network (ANN)
  - K Nearest Neighbors (k-NN)
Summary

- Mult-Language support – MPI, OpenMP, OpenGL - for heterogenous multicore systems
  - Unified parallel intermediate representation
- Analytical approach not feasible due to complexity
  - Explore optimization space via experiments and machine learning
- Static and Runtime Optimizations
  - Program transformation
  - Tuning of runtime parameters