Predictive Thread-to-Core Assignment on a Heterogeneous Multi-core Processor*

Tyler Sondag, Viswanath Krishnamurthy, and Hridesh Rajan

Laboratory for Software Design
Department of Computer Science
Iowa State University

*This work is supported in part by the National Science Foundation under grant CT-ISG: 0627354
Overview

- Multi-core processors becoming commodity
  - Performance-power ratio: heterogeneity among cores
  - Efficient utilization: match thread’s needs → core’s resources

- Existing assignment methods
  - Require execution traces: Time consuming to obtain, requires expertise
  - Dynamically monitor performance throughout the entire execution: High dynamic overhead

- Contribution: automatic thread-to-core assignment
  - Phases i,j,k: i~j~k → sched(i) ~ sched(j) ~ sched(k)
  - Determine i~j~k statically, followed by sched(i) to determine sched(j) and sched(k) at low overhead
Heterogeneous Multi-core

- Chip multiprocessor in which cores have different characteristics (clock speed, cache size, etc.)
Heterogeneous Multi-core (cont.)

Advantages[1]

- More efficient space utilization
- Reduced heat dissipation and power consumption
- Often better performance-power ratio than their homogeneous counterparts
- High performance when thread requirements are matched with core's characteristics
- High throughput for workloads with high thread parallelism

Problem Statement

- We know that workloads will run more efficiently if sections of processes are mapped to cores which execute them most efficiently.
- How can we automatically determine such a mapping while incurring a minimal overhead?

Example Program

```
A → Core Type 1
B → Core Type 0
C → Core Type 1
```
Proposed Solution - Overview

- Apply the same thread-to-core mapping to all approximately similar blocks of code
  - Statically break the program into segments of code
  - Statically determine approximate similarity between these segments
  - Dynamically monitor runtime performance of a segment and use this information to make mapping decisions about similar segments
Approach

Break Program Into Segments

Static

Program → Static Analyser → Binary Instrumentation → Program'

Dynamic

Program' → System Scheduler → Performance Monitoring
Approach

Determine Approximately Similar Blocks

A B A C B A ...
Approach

Insert Code Into Application

Monitoring Code and Core Switching Code

Static

Program

Static Analyser

Binary Instrumentation

Program'

Dynamic

System Scheduler

Performance Monitoring

IOWA STATE UNIVERSITY
Approach

Monitor Block

Store Result

Monitor Block

Store Result

Monitor Block

Determine Core Mapping For Block Type A

Static

Program

Static Analyser

Program'

Binary Instrumentation

Dynamic

System Scheduler

Performance Monitoring

A
B
A
C
B
A
...
Benefits

- No expertise required by programmer or user
  - No sample inputs needed
  - No knowledge of architecture required
  - No knowledge of program characteristics needed
- Entire process can be carried out automatically
Evaluation Plan

Current system
- Quad core processor with 2 cores under-clocked
- Use hardware performance counters to gather runtime statistics (instructions retired, cycles)

Method performance evaluation
- Create workloads from randomly selected benchmarks from the SPEC2000 benchmark suite
- Compare our method with standard assignment on the system
- Additionally, compare our method with methods from related works
Preliminary Results

- With no scheduler modification
- Median decrease in workload runtime

![Box plot showing runtime comparison between Standard Linux Assignment and Our Approach for a 6 Process Workload.](image-url)
Conclusion

- Apply the same thread-to-core mapping for all approximately similar blocks
- No knowledge of the architecture or program behaviour is required by the programmer
- No input sets are required for the programs
- Low dynamic overhead
- Entire process can be carried out automatically
Questions
Proposed Solution - Details

- Static Analysis
  - Group instructions into groups of similar instructions (ex: arithmetic, data transfer, etc.)
  - Find all basic blocks longer than some number of instructions
  - For each block, find the percentage of each instruction type

<table>
<thead>
<tr>
<th>load</th>
<th>add</th>
<th>load</th>
<th>add</th>
<th>store</th>
</tr>
</thead>
</table>

Data Transfer - 60%
Arithmetic - 40%
Proposed Solution - Details (cont.)

• Put these percentages into a vector

\[
(0.6, 0.4, 0, \ldots)
\]

• Use the k-means method[1] to group blocks into \( k \) clusters of similar blocks
  • Randomly select \( k \) cluster centers
  • Put vectors into closest clusters and recalculate centers
  • Repeat until cluster centers no longer change

• This groups basic blocks based on instruction type

Proposed Solution - Details (cont.)

- Dynamic Assignment
  - At the start of a basic block
    - If a core mapping has not been decided, monitor the performance
  - At the end of a block that is being monitored
    - Stop monitoring the performance
    - Store the result
    - If we have results from all processor types for this block type, determine the mapping
Proposed Solution - Details (cont.)

A | B | A | C | B | A | …

Monitor Block

Store Result

Monitor Block

Store Result

Monitor Block

Determine Mapping For Type A
Related Work

**Becchi et al. [1]**
- Proposes a dynamic mapping approach
- Looks at “fast”-”slow” core IPC ratio to determine thread migration
- This data is gathered from execution traces
- Our method does not require these traces

**Kumar et al. [2]**
- Another dynamic mapping approach
- After a certain amount of time a sampling phase gathers statistics from performance counters and determines mapping of all threads
- Our approach attempts to reduce the dynamic overhead


Related Work

Sherwood et al. [1]

- An approach to determine similar sections of execution
- Instructions during execution are grouped into large groups
- Similarity between these groups is computed based on the number of times each basic block is executed
- Our approach determines approximate similarity between basic blocks (rather than sections of execution) based on instruction type (rather than basic blocks)

Future work

- Currently similarity is based only on instruction type. This can be expanded to consider instruction order, estimated cache hits/misses, etc.

- Instead of using basic blocks, look at $n^{th}$ order intervals or other ways to group basic blocks.
Overview

- Problem: How do we make full use of the architectural capabilities of this class of processors

- One solution: Determine an effective thread-to-core mapping

- Our approach: Dynamically determine mapping of a block of code, then use the same mapping for all approximately similar sections