BDSC-Based Automatic Task Parallelization: Experiments

Dounia KHALDI

CRI, Mathématiques et systèmes
MINES ParisTech

Septièmes rencontres de la communauté française de compilation,
Dammarie-les-Lys, France
December 04, 2013
Context and Motivation

- Anyone can build a fast CPU. The trick is to build a fast system. Attributed to Seymour Cray
- Parallelism handling:
  - Parallel software developed by converting sequential programs by hand
  - Automatic task parallelism extraction: **Scheduling** problem
  - Resource constraints: memory requirements, processor features...
  - Scientific, signal and image processing benchmarks

**Automatic Resource-Constrained Static Task Parallelization**

- **BDSC**: a memory-constrained, number of processor-bounded extension of DSC
- Experimentation on shared and distributed memory systems
1 BDSC: A Memory-Constrained, Number of Processor-Bounded Extension of DSC

2 Experimental Evaluations with PIPS

3 Conclusion and Future Work
Parallelization Process

- C Source Code
- Parsing
- Sequential IR
- PIPS Analyses

Program DAG (sequences)

Execution Time, Communication Cost and Memory Size

Polynomial Estimation

Numerical Profile

BDSC

Scheduled Task Graph
1. BDSC: A Memory-Constrained, Number of Processor-Bounded Extension of DSC
   - List-Scheduling Heuristics
   - Dominant Sequence Clustering (DSC)
   - BDSC: A Resource-Constrained Extension of DSC

2. Experimental Evaluations with PIPS

3. Conclusion and Future Work
Priorities are computed for all unscheduled vertices using:
- Top level \((tlevel(\tau))\): length of the longest path from entry to \(\tau\)
- Bottom level \((blevel(\tau))\): length of the longest path from \(\tau\) to exit

<table>
<thead>
<tr>
<th>task</th>
<th>tlevel</th>
<th>blevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>(C)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>(A)</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>(B)</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Vertex \(\tau\) with the highest priority is selected for scheduling
\(\tau\) is added to the cluster (logical process) with the earliest start-time
A List-Scheduling Heuristic: Dominant Sequence Clustering (DSC)

- DSC (Dominant Sequence Clustering) [Yang and Gerasoulis 1994]
- Task list-scheduling heuristic for an unbounded number of clusters
- priority(\(\tau\)) = tlevel(\(\tau\)) + blevel(\(\tau\))
- zeroing(\(\tau_p, \tau\)) puts \(\tau\) in the cluster of a predecessor \(\tau_p \Rightarrow\)
  reduces tlevel(\(\tau\)) by setting to zero the cost of the edge (\(\tau_p, \tau\))

Complexity: \(\mathcal{O}(n^2 \log(n))\)

<table>
<thead>
<tr>
<th>step</th>
<th>task</th>
<th>tlevel</th>
<th>blevel</th>
<th>prio</th>
<th>scheduled</th>
<th>tlevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>0*</td>
<td>(\kappa_0)</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>(\kappa_1)</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>3*</td>
<td>(\kappa_1)</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>2*</td>
<td>(\kappa_2)</td>
</tr>
</tbody>
</table>

\(\kappa_0, \kappa_1, \kappa_2\)
A List-Scheduling Heuristic: Dominant Sequence Clustering (DSC)

- DSC algorithm weaknesses for our purpose:
  - Unbounded number of clusters
  - Number of clusters is not predefined $\rightarrow$ blind clustering
  - Memory size is not predefined $\rightarrow$ blind clustering
  - Creates long idle slots in already existing clusters

Proposal

BDSC: A Memory-Constrained, Number of Processor-Bounded Extension of DSC
1. Memory Constraint Warranty (MCW):
   - Do not exceed a memory threshold $M$
   - Overapproximation of the amount of memory used in tasks
     
     $$\text{data}_\text{size}(\text{cluster}_\text{data} (\kappa) \cup \text{task}_\text{data} (\tau)) \leq M$$

2. Bounded number of clusters $P$:
   - Number of cluster allocations do not exceed Threshold $P$
   - Maintain the constraint MCW
     
     $$\arg \min_{k \in \text{clusters}} \text{cluster}_\text{time}(\kappa)$$

3. Efficient cluster allocation by exploiting idle slots

4. Complexity: $\mathcal{O}(n^3)$
<table>
<thead>
<tr>
<th>Related Work: Static Task Parallelization Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Dependence</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>blevel</td>
</tr>
<tr>
<td>BDSC Parallelization</td>
</tr>
<tr>
<td>Sarkar's work [Sarkar, 1989]</td>
</tr>
<tr>
<td>OSCAR [Kasahara et al., 1992]</td>
</tr>
<tr>
<td>Pedigree [Newburn and Shen, 1996]</td>
</tr>
<tr>
<td>SPIR [Choi et al., 2009]</td>
</tr>
</tbody>
</table>
Contents

1 BDSC: A Memory-Constrained, Number of Processor-Bounded Extension of DSC

2 Experimental Evaluations with PIPS
   - Experimental Setting
   - BDSC vs. DSC
   - Comparative Study with Faust Parallelizing Compiler

3 Conclusion and Future Work
Experimental Setting

1. **Benchmarks**
   - *Thales ABF* (Adaptive Beam Forming), with 1,065 lines
   - SPEC benchmark *equake*, with 1,432 lines
   - *Harris* corner detector, with 105 lines
   - NAS Parallel Benchmark *IS* (Integer Sort), with 1,076 lines

2. **Machines**
   - **Shared Memory**: host Linux (Ubuntu)
     - 2-socket AMD quadcore Opteron, 2.4 GHz
     - $M = 16$GB of RAM
     - gcc 4.6.3 -O3
     - OpenMP 3.0
     - Cluster ~ Thread
   - **Distributed Memory**: host Linux (RedHat)
     - 6 dual-core processors Intel® Xeon®, 2.5 GHz
     - $M = 32$GB of RAM per processor
     - gcc 4.4.6 -O3
     - Open MPI 1.6.2
     - Cluster ~ Process
ABF and equake
Speedups with OpenMP and MPI

Number of cores

Speedup

Number of processors

Speedup
Harris

Speedups with OpenMP and MPI: Impact of Tiling (P=3)

![Graph showing speedup comparisons between Harris-OpenMP, Harris-tiled-OpenMP, Harris-MPI, and Harris-tiled-MPI for different image sizes (1024x1024, 2048x1024, 2048x2048) and sequential execution times.]
NAS Parallel Benchmark IS
Speedups with OpenMP and MPI: Different Class Sizes

Speedup OpenMP vs. sequential

- Class A (sequential = 1.68 s)
- Class B (sequential = 4.55 s)
- Class C (sequential = 28 s)

Benchmark version:
- OMP (2 tasks)
- OMP-tiled-2
- OMP-tiled-4
- OMP-tiled-6
- OMP-tiled-8

Speedup MPI vs. sequential

- Class A (sequential = 0.26 s)
- Class B (sequential = 1.69 s)
- Class C (sequential = 13.57 s)

Benchmark version:
- MPI (2 tasks)
- MPI-tiled-2
- MPI-tiled-4
- MPI-tiled-6
Faust Parallel Scheduling vs. BDSC

- Faust (Functional AUdio STream) [Orlarey et al., 2009]
- DSL for real-time audio signal processing and synthesis
- Generation of C or C++ with or without OpenMP directives
- `omp task` (BDSC) vs. `omp section` (Faust Parallelizing Compiler)
- Scheduling: BDSC vs. Faust topological ordering
- Speedups for two programs: Karplus32 and Freeverb
1 BDSC: A Memory-Constrained, Number of Processor-Bounded Extension of DSC

2 Experimental Evaluations with PIPS

3 Conclusion and Future Work
Conclusion

1 BDSC-based hierarchical scheduling algorithm
   - Memory constraint, bounded number of clusters, efficient cluster allocation
   - BDSC-based task parallelization algorithm
   - Communication, data and time cost models

2 Experiments:
   - BDSC-based automatic parallelization in PIPS
   - Code generation in OpenMP and MPI
   - Good speedups for coarse-grained parallelism
Future Work

1. **BDSC Scheduling**
   - Handling of heterogeneous devices
   - More precise cost models

2. **Parallel Code Generation**
   - More experimentation needed
   - Solving communication generation problems (MPI)
   - Hybrid task + data parallelism
BDSC-Based Automatic Task Parallelization: Experiments

Dounia KHALDI

CRI, Mathématiques et systèmes
MINES ParisTech

Septièmes rencontres de la communauté française de compilation,
Dammarie-les-Lys, France
December 04, 2013
Stream Compilation for Real-Time Embedded Multicore Systems.

A Multi-Grain Parallelizing Compilation Scheme for OSCAR (Optimally Scheduled Advanced Multiprocessor).

A Transformation Framework for Optimizing Task-Parallel Programs.

Automatic Partitioning of Signal Processing Programs for Symmetric Multiprocessors.

Partitioning and Scheduling Parallel Programs for Multiprocessors.
MIT Press, Cambridge, MA, USA.